Corpus-based Connectionist Parsing

By

Jonathan Andrew Tepper

Dissertation submitted to the Faculty of Engineering and Computing
of The Nottingham Trent University in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy

February 2000

Supervisors

Dr Heather M. Powell
Dr Dominic Palmer-Brown

Abstract

The syntactic analysis or *parsing* of realistic subsets of natural language is the greatest obstacle to achieving practical natural language processing (NLP) systems. Classical symbolic models of syntactic parsing are typically constrained by large sets of grammar rules that attempt to capture numerous linguistic exceptions and generalisations that are prevalent in the everyday use of language. However, the grammar rules are constantly open to amendment and are unable to encode the necessary level of abstraction required to gain a reasonable coverage of the language. The greater the coverage of language, the more difficult it is to implement the rules in a parser due to the increased complexity. A further computational inadequacy of current symbolic parsing systems is that processing is serial in the majority of implementations and therefore cannot simultaneously make use of multiple sources of constraints and information.

This dissertation explores the use of connectionist networks for parsing realistic natural language domains. Connectionist natural language processing (CNLP) is a relatively new approach to language processing compared with classical symbolic methods. A decade of connectionist research has provided new methods of representation and approaches to parsing. Connectionist networks' inherently distributed knowledge representation and their parallel processing behaviour has enabled linguistic information to be represented as soft-rules or constraints rather than by 'hard-wired' symbolic rules. Connectionist networks therefore possess numerous attributes that are well-suited to language modelling.

The parsing model proposed in this dissertation integrates both connectionist and symbolic techniques to formulate a hybrid data-orientated parsing system that is trainable and able to acquire linguistic knowledge directly from pre-parsed sentence examples extracted from a large parsed corpus. The connectionist modules of the system enable the automatic learning of linguistic structure and provide an inherently distributed representation of linguistic information that exhibits tolerance to unfamiliar input data and that is able to generalise from previous sentence examples. The Lancaster Parsed Corpus (LPC) is used as the source of the training and test data. Three connectionist architectures are used. A Temporal Auto-Associative Simple Recurrent Network (TASRN) is required to discover the beginning of a phrase; another TASRN is required to discover the end of a phrase; and a feed-forward Multi-Layer Perceptron (MLP) network is required to recognise the phrase that has been extracted by the TASRN networks. This method of phrase segmenting and recognition provides a powerful technique for processing arbitrarily long and complex sentences. The symbolic components allow information to be stored in an easily interpretable and manipulable manner and provides the basis for organising the parse. The connectionist and symbolic components interact to form a deterministic shift-reduce parser that parses sentences from right-to-left. A modification of the Back-propagation learning algorithm that enables MLP networks' to dynamically focus on training patterns that have high errors is also presented. As learning takes place, the learning rate coefficient is adjusted in response to each individual pattern error. The results obtained from experiments with artificial and natural language domains are encouraging. It improved training times in most instances and in some cases allowed the removal of pattern replication used to balance the training data. Also, the evaluation method used to test the performance of the connectionist networks is based upon the natural and pure generalisation levels produced by the networks in response to unique linguistic input.

In contrast with previous approaches to syntactic parsing with connectionist networks, the corpus-based model proposed is able to process large and varied samples of naturally occurring English text and sentences that are of arbitrary length and complexity. The system exhibits high levels of syntactic generalisations at both the module level and the sentence level which provides the system with a realistic coverage of the language, a feature lacking in previous hybrid parsers. Crucially, the model is adaptable to the grammatical framework of the training corpus used and is not predisposed to a particular grammatical formalism thus widening the scope and reusability of the parser.

Acknowledgements

I would like to say a very big thank you to my supervisors, *Heather Powell* and *Dominic Palmer-Brown*, whose prolific understanding, encouragement, and patience enabled me to gain direction and the most out of post-graduate study.

Many thanks to *William Hallidie Smith* and *Richard Broadley* from Student Support Services for helping me throughout the duration of my studies.

Much love and a very special thanks to my partner, *Jane Carter* and her son, *Sam*, who have endured my anxieties over the years and have been a pillar of support for me. *Jane* and *Sam* your patience has been admirable and you are my inspiration.

Also, a big thanks to *Matt Hawkins, Darren Matthews, Shirley Massey* and *Jay Bridges* for their much valued friendship and support throughout the years - and to all my other friends in Burton-On-Trent and Nottingham.

Finally, a loving thanks to my mother, *Celia*, brother *Matthew* and to my 'Nottingham family' *Irene* and *Alan Poxon* for their care and support throughout my time of study.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Introduction

Automatic processing and understanding of natural language is an important area for computing research. If automatic natural language understanding (NLU) were achieved it would enable people to converse more naturally with a computer system without the restrictions imposed by artificial languages. Research in this area may also increase our understanding of human languages and minds. As human linguistic communication is via speech the problem of natural language processing (NLP) can be broken down into two fundamental steps. The first step is the processing of spoken language by analysing and interpreting phonetic information extracted from speech signals in order to produce the corresponding text - this process is known as speech recognition. The second step is to understand the meaning and linguistic relationships of the above text by using lexical, syntactic and semantic knowledge.

This dissertation concentrates on the *syntactic analysis* of text. Syntactic analysis or *parsing* plays a central role in NLU systems and is used to recognise the structural relationships between words within a sentence. The ability to syntactically parse unconstrained natural language is the greatest obstacle to achieving practical NLU systems. The dissertation explores the use of connectionist networks (CNs) [1, 2a, 3, 5, 16, 18, 21, 27, 68, 70] to perform large-scale parsing tasks. Connectionist networks for language parsing (a process referred to as *connectionist parsing*) is a relatively new approach to language parsing compared with symbolic computational methods. Just over a decade of connectionist research has provided new methods of structure representation and approaches to parsing. The inherently distributed knowledge representation and parallel processing behaviour of connectionist networks has enabled linguistic information to be represented as soft-rules or constraints rather than as symbolic rules. The ability of connectionist networks to learn by example make them suitable candidates for use within a trainable system that automatically extracts linguistic constraints from sentence examples.

Advancements in hardware and software technology together with the consequential emergence of large collections of annotated natural language corpora has also made it feasible to investigate the use of complex connectionist architectures using large data sets. This enables the development of trainable parsers that are able to make significant contributions to NLU systems whose aim is to process realistic subsets of natural language.

1.2. Motivations

Traditional or *classical* implementations of syntactic parsers are symbolic in that they consist of explicit and discrete symbolic items representing atomic tokens, concatenative data structures such as trees, lists, and stacks for structural representation and predicate calculus for symbolic computation. Symbolic representations and computation provide a well understood method of implementing formal linguistic theories of language and automata and have thus become the traditional method of modelling human and artificial language processing. A major appeal of symbolic representations is that information and recursive data structures can be represented and manipulated in an easily interpretable way.

Classical symbolic parsers are typically constrained by large sets of grammar rules that attempt to capture numerous linguistic constraints, exceptions and generalisations that are prevalent in the everyday use of language. However, when processing unconstrained natural language text, the need for explicit and comprehensive grammar rules within symbolic parsers have severely limited their success. They have difficulty in processing input sentences that do not strictly adhere to predefined rules of the grammar and lack effective recovery mechanisms in the event of parse failures. The use of grammar rules to provide the linguistic framework for a computational parser also risks low coverage because the rule-based constraints limit the number of acceptable grammatical structures and thus the ability of the parser to cope with numerous rule exceptions present in natural language. The serial search mechanisms involved in symbolic parsing are also computationally intensive and this increases with grammar size.

Connectionist networks consist of simple interacting processing units that are usually arranged in layers with variable patterns of interconnectedness. Knowledge is represented as connection strengths (or *weights*) between connected units. Learning occurs when general recursive rules are applied to adapt these connection strengths in order to produce desired output responses to particular input stimuli. This provides a method of machine learning that inherently supports parallel computation, associative retrieval of memories, and a distributed representation of information that gives CNs an ability to combine multiple sources of knowledge during processing. The distributed representations of CNs provide robustness with respect to incomplete input data and can learn complex and powerful transformations directly from training examples. Their inherent ability to generalise in order to process unfamiliar input data also make them suitable for processing slight grammatical variations. CNs therefore provide a mechanism for overcoming the limitations associated with using strict rule-based grammars and present a plausible framework for parsing systems that can automatically acquire linguistic knowledge by example, and can perform fast parallel computation.

However, there appears to be no clear consensus concerning the most suitable role CNs can play in the parsing process. There are a number of research questions which must be addressed in order to develop a trainable system to perform parsing of unconstrained natural language :

i. Can a single CN perform all the basic functions of a parser or is more than one CN required? If more than one is required, what will be the function of each connectionist network?

ii. What type of CN architecture is most suited to the task?

iii. What symbolic components and structures are required and how will they interact with the connectionist components? Likewise, how will each CN interact with one another if more than one CN is required?

iv. How will the parser establish temporal relationships between the sequential input words of a sentence to form a hierarchical structure that represents the necessary syntactic relationships?

v. Is it possible to totally remove the rule-base associated with symbolic parsers by using CNs?

Current connectionist parsing models have been unable to process realistic subsets of naturally occurring language. This implies that an unrealistic or incomplete set of grammar rules have been defined for the parser's grammatical framework or that the architecture has been inadequate to acquire and apply the general principles of the language from the training data.

The ultimate aim of this work is therefore to produce a hybrid parser that is trainable, robust and able to learn realistic grammatical constraints without the explicit use of grammar rules. Secondly, unlike previous connectionist/symbolic parsing systems, another aim of this work is to develop a parsing architecture that is not ultimately predisposed to a particular grammatical framework thus widening the scope and reusability of the parser.

1.3. Organisation of the Thesis

This chapter has briefly discussed syntactic parsing within NLU systems, and the motivations for the approach to parsing taken here. Chapter 2 reviews previous approaches to natural language parsing including traditional symbolic models, connectionist parsing models and statistical corpus-based parsing models. Since a key feature of the work presented in this thesis is the use of modular connectionist architectures, connectionist structure representations will also be covered. Chapter 3 addresses the issues regarding the type of connectionist architectures to use and what parsing roles the architecture can adopt. This is achieved through preliminary investigations that use simple context-free languages as the grammatical framework for two experimental parsers. The first parser uses a feed-forward MLP network to act as an arithmetic expression recogniser within a hybrid shift-reduce parser. The second parser extends the technique to examine whether a single feed-forward MLP network can simultaneously perform all of the main tasks of a parser for a simple context-free natural language grammar. Chapter 4 then follows with a high-level overview of the hybrid syntactic parsing model that embodies the main research effort. The modular architecture developed is motivated by the results of the preliminary investigations conducted in Chapter 3. Chapter 4 describes the linguistic properties of the LPC and then presents the input representation, parsing algorithm and architecture. A parsing example is then given to illustrate the full shift-reduce behaviour of the parser. The composition of the training and test samples is described.

The next two chapters describe each component of the parser. Chapter 5 provides a detailed description of the phrase segmentation aspects of the parser, including the look-back and look-ahead mechanisms that are responsible for the deterministic behaviour of the system. This is followed by the associated training and test results. Chapter 6 then presents the context-sensitive phrase recognition module of the parser, demonstrating the adequacy of feed-forward MLP networks for simple phrase invariant recognition. Chapters 5 and 6 have presented each module of the parser and the module-level testing. It is left until Chapter 7 to present the parser's behaviour as a whole using an LPC test sample of novel sentences and a number of sentences composed to test the response to specific grammatical constructions. Tests are performed to assess the parser's structural preferences and its behaviour when presented with sentences containing common syntactic constructions and problems. A comparison is also made with other parsers. Chapter 8 presents a summary of the main findings, and points to some future directions for the research.

Chapter 2

Literature Review

2.1. Introduction

This chapter provides a thorough review of the research into traditional symbolic parsing models, connectionist parsing models and statistical corpus-based parsing models. First, Section 2.2 presents the basic concepts of syntax analysis and parsing, and reviews the advantages of contemporary syntactic theories. The use and limitations of rule-based grammars are contrasted with that of data-orientated grammars. An argument is also made for the use of syntax within natural language modelling. Section 2.3 critically reviews the traditional symbolic approaches to parsing and efficient deterministic-based parsers are favoured over non-deterministic models. Section 2.4 explains the fundamental connectionist approaches to parsing and the associated structural representations. Localist and distributed parsing models are critically analysed and an argument is given for the preference of distributed-based methods over localist techniques for large-scale language parsing. Section 2.5 critically explores statistical corpus-based techniques and the avoidance of fixed grammar rules in favour of data-orientated frameworks. Finally, Section 2.6 concludes with a brief summary of the chapter.

2.2. Syntax Analysis And Parsing

2.2.1 Introduction

Syntax analysis plays a central role in linguistic theory, and is used to transform linear sequences of words into hierarchical structures that represent how the words within a sentence relate to one another. The linguistic input presented to a syntactic analyser has usually undergone some morphological analysis (or part-of-speech tagging) which uses a lexicon to associate input words with atomic or structure-based lexical tokens that describe the words linguistic properties and meanings. It is these lexical tokens and structures which are then passed on to the syntactic analyser for processing.

A syntactic analyser traditionally consists of two components, a *grammar* and a procedure called a *parser*. The grammar is a declarative representation of the syntactic facts about the language and explicitly states how linguistic input (i.e., words and phrases) can be combined into larger phrases and sentences in order to form structurally correct sentences. The words or syntactic roles (i.e., noun, verb, preposition etc) of the language are commonly atomic symbols and are thus referred to as *terminal* symbols. Phrases are higher level linguistic structures that consist of groups of words (or syntactic roles) and/or phrases. Phrases are commonly referred to as *constituent structures* and *non-terminal* symbols are used to represent them within a grammar i.e., the non-terminal symbol, VP, could represent a verb phrase which may consist of the terminal symbols verb, preposition and noun. Linguistic knowledge of a syntactic analyser is therefore encoded within the grammar, commonly as a set of pre-defined rules or constraints. The parser is a procedure or algorithm that decomposes a sentence into its constituent parts via applying the rules of the grammar to the input sentence. A parse tree is the resulting structure formed by the parser, which represents the structural relationships between input words in a hierarchical manner. The terms *syntactic parser* and *parser* will be used interchangeably to refer to a syntactic analyser throughout this thesis.

There are numerous problems that are encountered when constructing a syntactic parser for natural language and must be considered before designing such a system. The most common problems current parsing systems face are lexical ambiguity, local ambiguity, global ambiguity, embedded phrases and unbounded/cross-serial dependency (or filler-gap) constructions.

Lexical ambiguity refers to the ambiguity relating to each word and its syntactic role. Each word can be assigned one or more syntactic role depending upon its linguistic context. However, in some cases lexical ambiguity may have been resolved at the morphological phase.

Local ambiguity presents a different problem in that phrases could be assigned different structures and meanings when processed out of context, e.g., *Jonathan who uses the train arrives quickly*, the phrase *the train arrives quickly* has a different meaning to the whole sentence. When structural ambiguity cannot be resolved at the level of the whole sentence i.e., there is more than one possible structural and semantic interpretation of the sentence, then the problem of global ambiguity arises. In this situation, the syntactic analyser could output both interpretations or settle on one using some form of statistical measure.

Embedded phrases pose another serious problem for syntactic analysers The embedded phrase can be the same kind of phrase in which it is embedded or the same kind which dominates the phrase in which it is embedded. A *clause* is an embedded phrase e.g., *The dogs [that chased the cat] are hungry* - the embedded phrase, *that chased the cat*, must be processed before agreement between the subject and verb can be established. Formal grammars can handle embedding by having recursive rules. Generally, people can only understand sentences with a limited number of embedding. The parser would require a memory to store intermediate embedding so that previous constituents are not forgotten. Chomsky[6] explained that differences in acceptability of the amount of embedding is part of the study of performance and does not affect grammaticalness of sentences.

Unbounded dependency or filler-gap constructions are a common problem facing large-scale natural language syntactic analysers. Consider the WH-question, *I wonder who Jill loves.* The problem here, is that *love* is a transitive verb and does not have an immediately following direct object. These constructs are unbounded in that the dependency may extend across arbitrarily many clause boundaries. Syntactic analysers need some separate rule, principle or category to cope with a phenomenon of this type.

The rule- and principle-based theories of syntax provide us with rules and constraints that can be used to tackle the above problems associated with natural language processing. However, before considering grammars and their computational properties, it is first necessary to make the distinction between syntactic theories and grammars.

Syntactic theories are concepts and ideas that state how sentences can be judged to be structurally correct and well-formed according to a particular language. Grammars can be viewed as the main component within a computational system implementing this theory in order to empirically investigate the validity and adequacy of the syntactic theory. The grammar encodes the linguistic features and properties of the language and determines the competency of the parsing system. Grammatical framework is the most important consideration to be made when constructing a successful system, therefore Section 2.2.2 discusses the grammar types and their descriptive power. Section 2.2.3 critically considers the grammatical frameworks in use today and their importance. Section 2.2.4 considers the essential properties for developing efficient search procedures in natural language parsing. Finally, section 2.2.5 argues that the contribution of syntax cannot be ignored in favour of pure semantic- and pragmatic-based processing if large-scale natural language processing is to be modelled successfully, and systems that do omit syntactic processing are limited to incomplete performance.

2.2.2. Grammars And Computational Adequacy

The theories of grammar and in particular of generative or phrase structure grammar proposed by Chomsky in his book, Syntactic Structures[6] form the basis of the grammatical framework employed in the majority of syntactic parsers currently available and is therefore worthy of consideration.

The composition of a phrase structure grammar forms a system of rules that describes the valid sentences allowed by a language. A phrase structure grammar is defined by four things : a set of terminal symbols; a set of non-terminal symbols; a particular symbol designated as the start symbol; and a set of rules involving the terminal and non-terminal symbols. Grammar rules within a phrase structure grammar are termed *rewrite* or *production rules* and are typically of the form : *NP* $\rightarrow$ *determiner noun*[1]. The aim is to produce a grammar that generates all of those sentences, and only those sentences, that a native speaker judges to be grammatically correct. The expressive and descriptive power of a particular type of generative grammar is vital if natural language processing is to be successful, therefore the ability to measure the computational adequacy of the grammar being used is important. Chomsky[6] addressed this and developed a hierarchy of grammar types, known as the *Chomsky Hierarchy,* and provided the mathematical proof of the computational adequacy for the various types of generative grammar.

The Chomsky hierarchy of languages identifies four types of phrase structure grammar in order of representational

[1] The concept behind rewrite rules is that of being able to change one string of symbols into another. Non-terminal symbols can be changed by rewriting i.e., *NP*, for example is considered a non-terminal symbol since it can be decomposed into further linguistic tokens, i.e., a *determiner* and a *noun*. The atomic or terminal symbols, *determiner* and *noun* cannot be changed by rewriting and thus terminate the rewriting.

power : *enumerable set* (type 0), *context-sensitive* (type 1), *context-free* (type 2) and finite-state or *regular* grammars (type 3), the most powerful being type 0, and the weakest being type 3. A type 0 language is one in which a program could be written that would either specify whether any given string or sentence belonged to the language or not. Type 0 grammars have Turing Machine (TM) power and are characterised by unrestricted grammar production rules. Type 1 grammars describe the type of languages generated by context-sensitive grammars where the only restriction placed upon the production rules is that the right hand side (RHS) of each rule is at least as long as the left hand side (LHS). i.e., all productions are defined so that each rule defines the replacement of only one non-terminal in its LHS. Type 1 grammars are said to be context-sensitive because different productions may rewrite a non-terminal to different values dependant upon the surrounding symbols. Context-sensitive grammars correspond in computational power to linear bounded automata (LBA). LBA are restricted TMs which assume left and right end markers on the input, and moves beyond those end markers are disallowed.

Type 2 is the class of context-free languages generated by grammars whose productions are restricted such that the LHS of each is a single non-terminal symbol, and each RHS is a sequence of terminals and non-terminals. Context-free grammars have equivalent computational power to push-down automata (PDA), which are basically finite state machines with a stack-like memory. Artificial languages, such as compilers, are described with context-free phrase structure grammars. However, it is generally accepted that context-free phrase structure grammars alone are insufficient to process all syntactic phenomena associated with natural language. Context-free grammars that are augmented with complex linguistic categories can successfully process all the syntactic problems mentioned previously, provided an unbounded number of categories are allowed [75]. The term used to describe this extension to context-free grammars is *indexed grammars*. Indexed grammars correspond in computational power to nested stack automata (NSA) as defined by Aho [76] and have greater descriptive power than context-free grammars but less descriptive power than context-sensitive grammars, however this level representational and computational power is adequate for natural language syntax.

Finally, regular or type 3 languages are those produced by regular grammars which are defined by rules that have a single non-terminal on the LHS and on the right hand side, either a terminal symbol or a terminal symbol and a single non-terminal symbol. Regular grammars have the computational power of finite state automata (FSA) which are effectively non-recursive state transition networks, and are inadequate for describing natural languages and will not be considered further.

2.2.3. Grammatical Formalisms

There are a diversity of grammatical formalisms that have been investigated in parsing and language modelling, depending upon the nature of the application and on specific insights into language structure. When considering a suitable grammatical formalism for a natural language parsing system two important but conflicting requirements must be considered : *precision* and *coverage*. Precision refers to how well the grammar encodes constraints on possible sentences and possible meaningful relationships carried by those sentences. A precise grammar is important as that the more precise the grammar the better able it will be to rule out incorrect sentence structures. Coverage is an important consideration as it refers to the proportion of sentences that can be described by the grammar.

The three main types of grammatical formalisms employed in contemporary symbolic and connectionist parsing systems and that will be considered are *linguistic grammars*, *semantic-based grammars* and *data-orientated grammars.* Linguistic and semantic-based grammars are largely based on Chomsky's theory that language is governed by an innate system of rules that are 'hard-wired' into the human brain, and represents the rationalists view of human language understanding. Data-orientated grammars relate to the opposing view that language is actually governed by a system of statistical regularities, associations and constructions derived by learning. The statistical studies of language by Markov Models [77,78] form the basis of predominant data-orientated grammars.

Linguistic Grammars

Linguistic grammars are the most traditionally used grammatical framework implemented in current natural language processing systems. Due to their popularity in modern parsing systems, the well established linguistic grammars that will be briefly reviewed are Chomsky's transformation-based grammars, such as transformational grammar (TG) [45] and government binding (GB) theory [80,81,82], and extended phrase structured grammars such as generalised phrase structure grammar (GPSG) [47].

TG was developed by Chomsky and generally takes a lexicon and a set of context-free phrase structure rules and augments the system with transformations, which take structures created by the context-free rules and transforms them into new structures. The idea is that the structure of the active form is produced by the context-free grammar (i.e., termed the base component). Then a passive transformation changes the structure to produce the corresponding passive form (i.e., a transformation component or rule for converting an active phrase into a corresponding passive one). The *deep structure* of a sentence is the parse tree produced by the context-free grammar rules and contains the syntactic information necessary for interpretation by the semantic analyser. The *surface structure* is derived by the transformation rules. Chomsky's theory is purely syntactic in that it allows no interaction between the syntactic and semantic modules. Transformational grammars are procedural and inherently used for sentence generation and not analysis. Grishman[46] explains that there are a number of problems associated with reversing this procedure so that transformational grammars can be used for the recognition process. There is also no predefined order in which the transformations should be applied.

GB theory was developed by Chomsky and is the immediate descendant of TG. As with TG, GB theory consists of deep structure and surface structure, however there is only one rule - the *movement rule* or *move-α,* where α is a variable over syntactic categories. The central concept is that of the 'head' of a phrase, which is the linguistic unit that gives the structure its essential character e.g., the head of a verb phrase is the actual verb. GB theory uses a generic phrase template (i.e., X-bar theory [83]), which consists of general phrase structure rules that serve as templates for well-formed local phrase structure trees. The types of principles used in GB theory are that of *government* and *binding*. Government principles concern the relationship between the head and the categories dependant upon it, and the binding principles concern phrases being required to have some reference e.g., *John* and *himself* have the same reference in *John believes himself too late*. GB theory assumes that there are no construction-specific rules and this is the main departure from TG. Instead of rules for passive transformations, i.e., move the object to make it the subject, GB uses principles or constraints which state when such a transformation *cannot* take place rather than having a specific transformation rule. The 'head' of a phrase determines what other type of phrase can attach to it or become its argument. So although in theory, any phrase can be a head's argument, it is the head which filters out incompatible phrases. This process of filtering is referred to as subcategorisation (see Sells [83]).

Generalised phrase structure grammar is a theory of syntax that attempts to use only phrase structure rules. GPSG uses only one level of syntactic representation , surface structure and only one kind of syntactic object - the phrase structure rule. It augments a phrase structure grammar in certain ways that still leave you with a phrase structure grammar, but one that can handle constructions previously thought to be describable only with the aid of transformation rules; most common being the analysis of unbounded dependency constructions. GPSG's are considered a main rival of transformation grammars. Although GB theory placed constraints on TG transformation rules, GPSG totally removes the use of such rules. GPSG provides various constraints on what informational properties parts of the parse tree must encode. Phrase structure rules are not directly related to the parse trees. Rules of *immediate dominance* (*ID*) are directly related to trees by a rule-to-tree definition, and specifies which rules dominates others in a parse tree, but unlike standard phrase structure rules dominated nodes are unordered. Rules of *linear precedence* are used to govern left-to-right ordering among sister nodes dominated by a mother node. *Meta-rules* operate on ID rules and state generalisations across large sets of ID rules. It is the meta-rules that perform similar tasks to transformation rules in TG. However, rather than building new trees, meta-rules expand the set of phrase structure rules and therefore expand the set of parse trees it deems well-formed and grammatical.

Linguistic grammars are powerful theories of linguistic competence. However, it is difficult to incorporate discourse and pragmatic analysis into such techniques. For example, Hindle [99] states that whether or not a specific prepositional phrase modifies a direct object noun phrase is dependent upon the actual noun and prepositional object. However, common phrase structure grammars fail to make such information available in the syntactic categories of the noun phrase, and prepositional phrase. It therefore becomes the task of the search procedure to form the appropriate relationships that span constituents. The deficiencies of traditional phrase structure grammars have lead to a growing interest into *lexicalised* frameworks such as lexical functional grammar (LFG) [83, 84, 85], dependency grammar [86], categorial grammar [90,100], lexicalised tree-adjoining grammar (LTAG) [87] and more recently head-driven phrase structure grammar (HPSG)[82], all of which encode syntactic and semantic information in the lexicon. Lexicalised frameworks also form parse trees consisting of lexical items and direct relationships between them.

However, contemporary linguistic grammar alone has been unable to fully process all the idiosyncrasy associated with language. Grammar rules are always open to amendment to capture linguistic exceptions, and such alterations may unintentionally change the function of other rules in the grammar. Also, sentences are only termed grammatical if and only if the input sentence adheres strictly to the grammar. This level of precision constitutes a major disadvantage to the sole use of rules and their inability to cope with mild grammatical variations which are prevalent in naturally occurring language. The type of linguistic grammar used appears to have little correlation with computational efficiency. For instance, modern implementations of government-binding theory and standard transformational grammar involve either very complex search procedures or very complex compilation procedures into grammatical formalisms that use better search properties [101]. It appears that the success of a particular linguistic grammar is dependant on the use of an effective search or parsing procedure. The inadequacies associated with purely rule-based systems strengthens the empiricist view of language processing.

Semantic-based Grammars

The alternative approach to using linguistic grammars that are purely syntactic is to devise grammars that specify directly how relationships relevant to the problem domain may be expressed in natural language. Several approaches to the problem of creating a semantic representation of a sentence have been developed, and the most successful approaches are semantic grammars [89,91] and case grammars [92,140]. Semantic grammars are context-free grammars in which the choice of non-terminals and production rules are governed by semantic as well as syntactic function. In addition, there is usually a semantic action associated with each grammar rule. The result of parsing and applying all the associated semantic actions is the meaning of the sentence. The success of augmenting grammar rules with semantic actions is that actual the grammar rules are designed around key semantic concepts. The major advantages of using semantic grammars are that the output can be directly interpreted without further processing and many syntactic-based ambiguities can be avoided as some of the interpretations do not semantically make sense, and thus cannot be generated by a semantic grammar. Case grammars provide a more general approach to how syntactic and semantic interpretation can be combined within a task-orientated formalism. Grammar rules are written to describe syntactic rather than semantic regularities. But the structures the rules produce correspond to semantic relations rather than to strictly syntactic ones. The case used by a case grammar describe relationships between verbs and their arguments. This contrasts with the grammatical notion of surface case rather than surface structure. A given surface case can describe a variety of semantic cases. There appears to be no clear consensus on what the correct set of semantic cases should be, but some common cases are : agent - instigator of the action; instrument - cause of the event or object used in causing the event; and dative - entity affected by the action. The process of parsing into a case representation is heavily directed by the lexical entries with each verb.

The advantages of using a case grammar is the inherent ability to interpret the meaning of the sentence and to rule out syntactic-based ambiguities. However, the result of parsing in a case representation is not a complete semantic description of a sentence. For example, the constituents that fill the case slots may still be English words rather than true semantic descriptions stated in the target representations. Further post-processing is therefore required to build meaningful representations.

The major drawback of using semantic-based grammars rather than a linguistic grammar is that the number of rules required can become very large since syntactic generalisations are omitted, this makes the parsing process computationally intensive. Semantic-based grammars provide very strong guidance to a parser, but that guidance is bought at the expense of generality and coverage, since the detailed specifications they rely on may often fail to capture naturally-occurring linguistic generalisations. The parsing algorithms of semantic grammars are therefore allowed to relax the grammar and ignore portions of the input that cannot be described by the given grammar [93]. Semantic grammars are very useful for restricted tasks and domain specific applications, but as an overall solution to the problem of natural language understanding, they will inevitably fail due to the inability to capture important linguistic generalisations [7].

Data-Orientated Grammars

Linguistic- and semantic-based grammars inherently provide tight constraints for language modelling and parsing, they risk low coverage because the rule-based constraints limit the number of acceptable grammatical structures. Semantic-based grammars overcome this problem but is only effective for highly-constrained tasks. An alternative approach of increasing coverage is to start with a grammar that has minimal constraints, and to use statistical search techniques to rule-out poor derivations and to select the most probable derivation from the complex search space that results from the less constraining grammar. The derivations chosen by the system would be based on some prior statistical processing phase. Within the past several years numerous researchers have begun to investigate data-orientated frameworks for the use of *trainable* systems in natural language processing [94, 95, 96, 97, 98, 103, 142, 164, 165]. In a data-orientated formalism, a learning or training procedure tries to determine the search procedure that produces best results on an appropriate training corpus[2].

Systems based on the data-orientated framework use either hand-crafted linguistic grammars or large text corpora containing a variety of naturally occurring sentences as the data source from which to extract structural information. The problem with training the system with a hand-crafted grammar is that either the grammar will allow too many structures that rarely naturally occur, or the opposite in that it is too restrictive and does not allow structures that do naturally occur. This has lead to the investigation of grammatical frameworks and learning algorithms that will concurrently learn a grammar and an appropriate search procedure.

The use of n-gram[3] statistics for example, rule out bad word sequences by assigning probabilities to transitions from one n-gram state to another. These are probabilities derived from how often states were reached and transitions crossed running the grammar over a training corpus [95,102].

[2] A training corpus is a large body of sentence data that contains the necessary syntactic-semantic information to be automatically acquired by some data-orientated training procedure.

[3] It is worth noting that if there are *n* words presented to the parser per time step , the input is considered an *n-gram*. For example, if two words are presented to the parser per time step, then the input is considered a *bigram*; similarly, if three words are presented to the parser, the input is considered a *trigram*.

The success of such systems have allowed the emergence of a new generation of systems that both extract statistical information from and summarise pre-existing text from real-world domains. The aim of data-orientated techniques is to move closer to the reality of fully automatic syntactic analysis of unconstrained text and also towards the automatic acquisition of grammatical structure from both annotated and unannotated text corpora via learning.

2.2.4. Search Procedures And Parsing

Many natural language parsing systems explicitly use the grammar rules to describe a way of generating every possible legal sentence in the language defined by the grammar. The tree of all partially parsed and completely parsed sentences allowed by the grammar, i.e., a hierarchical tree structure whose root is some *S* or *start symbol*, and whose leaves are all the sentences that can be generated by the grammar, can be considered as a state space. The parser is therefore a search procedure that searches the state space in an attempt to discover a path from the *S* node to a desired sentence (referred to as *top down parsing*) or from a desired sentence to the *S* node (*bottom-up parsing*). There will also be more than one path through the tree for ambiguous sentences. Each path within the tree is referred to as a *derivation*. In parsing a natural language like English such a tree would possibly be infinite if it could be defined. The grammars that are created generally describe sub-trees or subsets of the English language but vary in terms of precision, coverage and generality.

Parsing systems that do not use fixed grammar rules and are probabilistic [94, 96,103,104] are described as inherently deducing grammatical constraints by statistically analysing the current input data and making decisions or linguistic predictions based upon statistical probabilities formed from previous analysis. Transitional probabilities are assigned to the possible next parse states allowed, given the current input symbol and the current parse state. These probabilities strongly constrain what the next state can be and thus form a localised state space, which is constrained further when further input symbols are processed. These parsing systems are said to form an implicit grammar directly from the data. A supervisory module is required to organise the control and flow of input symbols and statistical computations on the input data i.e., some controlling automaton or scheduler. An efficient and effective automaton is the shift-reduce parser proposed by Shieber [105,106], where the grammar corresponds to the possible transitions between the stack and input combinations, while the automaton's finite-state control determines which combinations are actually used in a derivation.

The task of the parser, regardless of the grammatical formalism, is therefore to apply transitions specified by the grammar, which is either explicit or implicit, in order to construct derivations for the given sentence. Top-down parsing procedures are non-deterministic[4] and tend to be inefficient due to the back-tracking control strategy they implement when grammar rules fail to match the input sentence. Also, the lack of effective heuristic techniques to select the correct grammar rule limits their use and can be excessively exhaustive and time consuming due to many dead-end paths being 'blindly' reached throughout the search.

[4] Strategies that are non-deterministic have more than one search path to explore in order to find the correct solution. Heuristics are required to select the optimum search path that would allow the least back-tracking.

Bottom-up parsers allow deterministic[5] approaches to be employed where constituents of the sentence are read into the parser's input buffer until the correct grammar rule can be selected from the grammar thus implementing a look ahead mechanism. The main disadvantage of deterministic techniques is that if the correct grammar rule or valid next state cannot be determined before the parser's input buffer or memory is exceeded then the parse fails and the input sentence is rejected. However, such parsing failures have been proven to model failures in human syntactic processing [14, 51, 53, 99]. Bottom-up parsers remove the need for back-tracking via using look aheads to reduce the number of possible rules or states to consider and are consequently more efficient than top-down parsers.

An efficient search procedure must therefore avoid searching the entire grammar search space as it is not computationally feasible due to the effect of grammar size on search space size. The deterministic-based parsers help reduce the search space by disregarding possible state configurations as more input information is used as look aheads, and are also suitable for modelling human syntactic processing. If there is more than one possible derivation within the search space then some method, mainly statistical, must be employed to find a single plausible interpretation. If the resulting interpretation is later rejected for semantic or pragmatic reasons, then a new attempt to find a more suitable interpretation can be made.

2.2.5. The Need For Syntax

There has been much debate as to the need for syntax analysis in natural language modelling. Although syntax analysis aids the extraction of semantic information from sentences, there is not a one-to-one correspondence between the syntactic components and the semantic components. For example, in the sentence *I saw the man on the hill with a telescope*, it is easy to extract the two prepositional phrases, but it is unclear as to whether the man on the hill had a telescope, or a telescope was used to see him. Thus, while structure contains meaning, it alone does not determine meaning. This instance of global ambiguity is a typical example of the difficulty in distinguishing whether to separate or integrate syntactic and semantic processing and raises numerous questions concerning the need for syntax. The main issue being whether a syntactic tree showing grammatical relations between words is necessary for deciding on the semantic representations of a sentence.

Chomsky's transformational theories of syntax are the most common grammatical framework used in natural language processing systems even though they do not allow for semantic interaction i.e., the syntactic module must complete its processing before the semantic module can process the resulting syntactic structures. This has lead to numerous criticisms of syntactic parsing and the psychological plausibility of Chomsky's theory of linguistic competence. For example, psycholinguists undertook experiments to test whether people's ability to memorise and evaluate meanings of sentences is affected by the number and complexity of transformations needed to generate the sentences, and whether syntax analysis is performed independently of semantic processing [107].

[5] Deterministic in the sense that enough information is obtained so that it is possible to determine the correct category of every word and phrase as it appears.

The experiments demonstrated that people do not complete full syntactic analysis of sentences before starting to interpret meanings, and in some cases only consider semantic interpretations thus bypassing the syntactic stage altogether. However, deterministic methods of language parsing have provided valuable insight into human language parsing failures, such as failures in garden-path sentences, and also form the basis of efficient syntactic parsing models [14,51,53].

The refutation of Chomsky's theory of language has prompted numerous researchers to totally by-pass syntactic processing in favour of pure semantic and pragmatic-based computational models of human language processing [108,109,110]. However, such systems have been unable to process large-scale natural language and have been largely constrained to small subsets of natural language or fixed task-orientated domains. The limited performance of non-syntactic-based systems is due to the omission of syntactic generalisations and this forces the rules of the system to rely on detailed semantic grammar rules that reduce generality and coverage. Although it is possible to process linguistic input without using a syntactic analyser, it has not yet been proven that a general-purpose processor can be built on this basis. The performance of semantic-based systems prove that processing language without using syntax is restricted to incomplete performance. The more successful approaches to language modelling allow a syntactic parser to refer to the semantic or statistical processing of a sentence in order to decide between competing syntactic structures [111].

A significant justification for the use of syntax in modelling human sentence processing is demonstrated by the language processing capabilities of individuals with Broca's aphasia which renders them incapable of processing syntactic structure but are who fully able to recover the lexical representation of content words and use pragmatic information [7]. It is argued by structural and empirical linguists that natural language processing systems that omit syntactic processing may be doing exactly what people with aphasia do - partially processing the input without using syntactic information, and attaining results which are successful up to a point and in restricted domains. The contribution of syntactic processing cannot be disregarded if natural language systems of a *general* application are to be developed and particularly systems that attempt to model human sentence processing.

2.2.6. Discussion

The use of linguistic grammars alone has been unable to fully process all the idiosyncrasy and apparent statistical regularities associated with language. Systems that rely completely on the use of grammar rules are always open to amendment and by nature are unable to encode the necessary level of abstraction required to capture realistic or even acceptable levels of natural language. This level of precision constitutes a major disadvantage to the use of rules and their inability to cope with mild grammatical variations which are prevalent in naturally occurring language. Semantic- and pragmatic-based language, processing systems have reported varying degrees of success. However, such success has been limited to unrealistic and task-orientated language domains, and at present are unable to realise large-scale language domains of a general nature due to the omission of the general linguistic principles provided by syntactic theory. It is evident that the contribution of syntax analysis must be recognised if a natural language system is to adequately process language, and any system that omits syntactic processing will be limited to incomplete performance.

In contrast to 'hard-wired' grammar rules, the emergence of data-orientated frameworks have allowed systems to statistically learn linguistic knowledge from naturally occurring sentence examples. This allows computational models of language to automatically form less constraining linguistic rules and to add further constraints as more complex language structures are learnt, thus providing a wide and realistic coverage of the language being described.

Linguistic theory has provided a fundamental basis for developing syntactic analysers that are powerful enough to process natural language. However, the majority of language models use purely linguistic grammars that lack flexibility and this has limited their success. It is envisaged that by integrating the principles employed in current linguistic grammars with a data-orientated framework will allow the development of a syntactic parsing system that can learn language from sentence examples without a prior knowledge of the language, and provide a system that is able to automatically capture syntactic generalisations. Also, a deterministic approach to syntactic parsing will help reduce the search space by disregarding possible state configurations as more input information is processed as look aheads, and remove the need for back-tracking mechanisms employed by top-down parsers.

The following sections critically discuss the type of computational techniques employed to implement linguistic theory.

2.3. Traditional Models of Language Parsing

2.3.1. Introduction

Traditional implementations of linguistic theory are symbolic in that they consist of explicit and discrete symbolic items to represent atomic tokens, concatenative data structures such as trees, lists, and stacks for structural representation and predicate calculus for symbolic computation. Predicate calculus uses variables as abstract representations of entities, predicates[6] as abstract representation of properties, and formulae as abstract representations of generalisations. Symbolic representations and computation provides a well understood method of implementing formal linguistic theories of language and automata and have thus become the traditional or *classical* method of modelling human and artificial language processing.

The appeal of symbolic representations is that information and recursive data structures can be represented and manipulated in an easily interpretable way.

[6] Predicates are functions that make relations between incoming variables/entities. For example, the predicate *parent(NP,x)* could yield the proposition 'NP is x's parent', where *parent* is the predicate and *NP*, *x* are the entities.

The most common models of parsing are based on non-deterministic top-down processing with context-free grammars augmented with heuristics. The heuristic processing is used to examine the search space and guide its syntactic decisions when presented with an ambiguity. Transition networks have been common symbolic implementation devices for such parsers. Another approach has been to employ deterministic bottom-up processing with a set of well-motivated computational restrictions to delay syntactic decisions until the most suitable structures can be selected. This section will provide a brief overview of key research in each of these areas.

2.3.2. Transition Networks And Non-Deterministic Processing

Researchers have developed numerous non-deterministic syntactic parsers over the last twenty five years and one of the earlier successful implementations of a non-deterministic syntactic parser was based on the transition network (TN) developed by Woods [8]. A transition network consists of nodes that represent parse states and links between parse states to represent words or phrases. TNs focus on transitions between each of the words or phrases of a sentence, starting at the beginning and working through the sentence from left to right, attempting to construct the syntactic structure of a sentence by identifying its constituents. However, the standard TNs have severe difficulty in processing ambiguity e.g., sentences such as *the old man the boats*, and there is no definite way of obtaining just one canonical parse. Also, context-free grammar rules used to describe natural languages are largely recursive and standard TNs were unable to model recursion. This lead to the development of recursive transition networks (RTNs) which have the ability to implement recursion in TNs via use of a stack where destinations are pushed onto the stack and control is then passed to a sub-network, say for a noun phrase, which is then traversed and control passed back to the pushed destination. RTNs provided system designers a mechanism for representing context-free grammars that include recursive rules. A RTN parser must search through the list of context-free rules at each computational step to find the correct rule, however if there is more than one possible transition that can be made at one particular point, then a back-tracking strategy must be employed. In this case, each competing path must be explored and the starting state must be stored in memory so that if the path chosen is incorrect and does not match the input sentence, the network can 'roll-back' to the designated recovery point and the other competing path can then be explored.

The disadvantage of RTNs is that the current structure of the input sentence is not stored at each time step, only roll-back points were stored rather than sentence structure. RTNs were therefore extended further to enable structural information to be available at each state in the transition network. These symbolic implementation devices are termed augmented transition networks (ATNs) [48] and allowed transition networks to implement back-tracking and look aheads. Even though ATNs are widely used to implement top-down parsing, ATNs are able to implement determinism as they have an augmented facility to suspend judgement about a group of words by holding them in a temporary store called a register while they look ahead or consult other transitions as necessary. Registers allow ATNs to take context into account before deciding which transitions to follow. The structure that is formed by an ATN is dependant upon the grammar writer. Usually, the structure is built from the information stored in the registers at the end of the parse.

The use of ATNs as an implementation device for parsing theories has been widespread [48,49,50,114], and the majority of academic and commercial natural language processing systems currently in use are based on the ATN formalism. Early work by Kaplan [121] developed a parsing strategy using ATNs that attempted to model human syntactic preferences when parsing ambiguous or temporarily ambiguous linguistic input. However, ATNs are often used as front-ends or interfaces to database systems for question answering, and can successfully integrate syntactic and semantic processing for task-orientated domains. The disadvantage of using ATNs is their inability to model complex grammars, as the more complicated the grammar the larger and more opaque the ATN to model it and thus reduces its suitability for realistic subsets of natural language.

Early work by Kimball [112] and Fodor et al [113] avoided using ATN formalisms and postulated explicit heuristics that would apply to specific syntactic constructions to guide the parser to the preferred structural analysis. Frazier [114,115,116] was inspired by Kimball's work and extended it further with her colleagues to develop an influential model that includes heuristics such as *Minimal Attachment* and *Late Closure*. The Minimal Attachment heuristic states that incoming linguistic input will be attached into the parse tree using the fewest nodes consistent with the well-formedness rules of the language. The Late Closure heuristic states that when possible, attach incoming linguistic input into the phrase or clause currently being parsed. These heuristics have been extremely influential on other research projects attempting to tackle ambiguity during parsing and to model human sentence processing [117, 118]. Frazier compared her parsing model with the ATN-based formalisms stating that the inability to access registers in order to make further syntactic decisions limited the application of ATNs. However, Frazier admitted that ATNs could be implemented in a way that could overcome this and heuristics such as Minimal Attachment and Late Closure could be implemented with ATNs. The major limitations of Frazier's work is that the structural preference and recovery mechanisms of the system are construction specific and therefore lack generality.

Ford et al [119] resolved ambiguity using a back-tracking parser by including lexical and structural information rather than purely syntactic information to resolve ambiguity. Fodor and Inoue [120] developed a more general non-deterministic parser that used heuristics to guide the parser in a way that would yield a universal mechanism that matches human behaviour in processing both English and Japanese. The parser attempts to determine the best structure that is compatible with the available evidence derived from the input, and a record of choice points is maintained to ease any necessary revisions. The major disadvantage of these approaches was the inefficient back-tracking control strategy the systems employed

Although the non-deterministic techniques with heuristics are powerful with regards to its ability to back-track and to process ambiguity, the dependence on construction-specific processing heuristics that lack generality make it unclear how successful this and similar systems would be in processing large-scale natural language. Another disadvantage of using non-deterministic parsing systems is that processing is inherently sequential and does not allow for information to combined and processed in parallel. The lack of parallel processing and the use of inflexible grammar rules that are commonly unrealistic in size or are task-orientated make it difficult to establish whether such systems are adequate for large-scale natural language parsing of general domains.

2.3.3. Deterministic Processing Models

The models previously mentioned use non-deterministic parsing strategies with built-in heuristics to deal with such problems as ambiguity resolution. An alternative and successful line of research has instead focussed on constraints that could lead to parsing systems that can remove the need to back-track. Deterministic processing techniques are employed in bottom-up parsing models which are inherently data-orientated. Constituents of the input sentence are read into the parser's input buffer until the correct grammar rule can be selected from the grammar. Deterministic parsers employ efficient search procedures as deterministic processing helps reduce the search space by disregarding possible state configurations as more input information is used as look aheads.

A predominant and influential deterministic parsing model is that proposed by Marcus [51]. Marcus termed his parser PARSIFAL and it exhibited similarities to human behaviour when processing complex sentences such garden path sentences. Garden path sentences are those which mislead the reader as to their correct syntactic structure and leave them in an impasse. This is because people are usually unwilling to reconsider earlier commitments. For example, in the sentence *The boat sailed down the river sank*, the verb *sailed* is assumed to be the main verb whereas the main verb is actually *sank* and *sailed down the river* is a relative clause as in *the boat that was sailed down the river sank.* Garden path sentences are informative as they seem to indicate that people make very firm commitments as to the syntactic structure of early parts of the sentence before they reach the end of a sentence, and that they are very reluctant to backtrack when they reach an impasse. Marcus designed PARSIFAL based on this idea. Grammatical rules within PARSIFAL are assigned a priority and each rule has multiple parsing actions and conditions similar to a programming language. It explicitly activates and deactivates rule packets, executes rules, creates new phrase structure nodes and tests for complexities contained within the input. Marcus stated that a maximum of five words are required at any one time to determine the syntactic category of a particular word within the input buffer. However, if after five words the system could not determine a correct syntactic category for a particular word then it would fail. The time it took to process a sentence was proportional to the length of the sentence. Milne [53] extends Marcus's parser, in an attempt to provide a more extensive account of human behaviour in resolving lexical ambiguities and of human inability to parse well-known grammatical constructions. However, the extensions made by Milne rely on very specific processing rules and assumptions concerning the amount of allowable look ahead.

Since the emergence of PARSIFAL, other parsers have been developed based upon the determinism hypothesis [52,54,122] . The more recent work has integrated constraints imposed by a deterministic mechanism with assumptions about the necessity of fast, incremental interpretation. The minimally specified parse tree representations defined in Marcus, Hindle and Fleck's D-Theory [123], have provided a principled form of limited parallelism in maintaining multiple structure analyses. Gorrell [124] uses similar D-theoretic motivations, relying on restrictions on the re-analysis of precedence relations to extend the account of human behaviour. Shieber's [105] shift-reduce parsing model is suitable for implementing deterministic parsing mechanisms. It uses heuristics that are built-in conflict resolution strategies which guide the parser when there is more than one action it can perform in response to a new input token. The shift-reduce strategy where shifts are preferred over reductions, provide an effective implementation of the Minimal Attachment and Late Closure preferences. However, it is uncertain how difficult it is to revise the initially preferred structures with such parsers.

The disadvantage of the more modern deterministic approaches is the lack of effective recovery mechanisms for recovering from parse failures.

2.3.4. Discussion

The grammar rules that form the basis of most non-deterministic parsers have a high level of precision and an intractable number of additional rules and heuristics would be required for such systems to provide an acceptable level of coverage. Also, the inability of ATNs' to model complex grammars has limited their use to task-orientated domains and other mechanisms must be used for modelling realistic subsets of natural language. The non-deterministic parsers that are augmented with heuristics are usually specific to modelling a particular type of phenomena such as ambiguity, rather than attempting large-scale natural language processing.

Syntactic parsers based on the determinism hypothesis provide a more accurate account of human syntactic processing and also provide an efficient computational framework that rejects the back-tracking control strategy implemented by non-deterministic parsers. However, there has been much debate to the optimal amount of look ahead required before making correct syntactic judgements. Also the lack of effective recovery mechanisms restricts the performance of deterministic models.

The general limitations of the previously mentioned parsing models are that they are strictly rule-based, and have difficulty in processing input sentences that do not adhere to the predefined rules of a grammar and lack effective recovery mechanisms to recover from parse failures. This limits such parsers ability to cope with numerous rule exceptions present in natural language, and provides no definite solution to sentence completion, lexical ambiguity and acquisition of exceptions. The use of grammar rules are constantly open to amendment and are unable to encode the necessary level of abstraction required to capture realistic subsets of natural language.

Perhaps the major computational inadequacy that current parsing models have is that they are implemented symbolically and processing is inherently performed sequentially and therefore cannot simultaneously share multiple sources of constraints and information. Sequential search mechanisms are also computationally exhaustive which is increased with respect to the grammar size. However, numeric computation is being introduced into symbolic parsers to address such problems. Gibson [125] developed a parallel parsing model in which grammatical constraints based on linguistic theory plays a central role in defining the computational restrictions on syntactic processing operations. The parser pursues all possible parses of the input sentence in parallel, and performs a calculation that determines the cost of maintaining each structural alternative. This numerical cost is based upon how well the structure satisfies its grammatical constraints. The costs are then used as the basis for pruning the search space. This provides an impressive account of human syntactic preference and recovery mechanisms and an adequate level of coverage. Gibson's model is an example of the move from sequential and discrete models to parallel and continuous modelling of language within symbolic frameworks.

2.4. Connectionist Models of Language Parsing

2.4.1. Introduction

Connectionist networks (CNs) [1,2a,3,5] consist of simple interacting processing units that are arranged in arbitrary layers with variable patterns of interconnectedness. Knowledge is represented as connection strengths (or *weights*) between connected units. Each unit performs a product summation of the output activation flow from other processing units and associated connection strengths. A linear or non-linear threshold-based computation is then performed to determine its own activation value, which will then be spread to other interconnected units. Learning occurs when general recursive rules are applied to adapt these connection strengths in order to produce desired output responses to particular input stimuli. This provides a method of machine learning that inherently supports parallel computation, associative retrieval of memories, and a distributed representation of information which gives CNs an ability to combine multiple sources of knowledge during processing. CNs distributed representations (Rumelhart et al [2a]) provide robustness with respect to incomplete input data and can learn complex and powerful transformations from training examples.

However, there has been much criticism made of connectionist representations to encode temporal structure, structured knowledge and variable binding [15,16]. Classical symbolic techniques are well suited to performing dynamic variable bindings and representing complex data structures required for language processing. This raises questions about the suitability of connectionist techniques for structural analysis and processing. Section 2.4.2 therefore briefly contrasts symbolic representations with connectionist representations and argues that connectionist distributed processing and representations are adequate for processing and representing linguistic information. Section 2.4.3 critically evaluates localist parsing models, and Section 2.4.4 critically evaluates distributed parsing models. Finally, Section 2.4.5 provides a brief discussion highlighting the major points made in the previous sections.

2.4.2. Symbolic Vs Connectionist Representations

One of the main problems of applying connectionism to high-level cognitive tasks, such as language processing, is their inadequacy to form structural representations. This was controversially highlighted by Fodor and Pylyshyn[15] in their criticisms of connectionism. It was their view that connectionists numerical representations had been unsuitable for capturing the dynamically-allocated, variable-sized symbolic data structures traditionally used in classical AI techniques. Fodor and Pylyshyn state that connectionist theories acknowledge only causal connectedness as a primitive relation amongst nodes; when you know how activation and inhibition flow among them, you know everything there is to know about how the nodes in a network are related. By contrast, they state that Classical theories acknowledge not only causal relations among the semantically evaluable objects that they posit, but also a range of structural relations, of which constituency is pragmatic. Fodor and Pylyshyn finally argue why the usual reasons given for preferring a connectionist architecture are invalid and conclude that connectionist implementations are in fact simply re-implementations of symbolic techniques and offer nothing new.

Since the critique by Fodor and Pylyshyn connectionists have been developing methods to enable connectionist networks to possess compositional structure with operations that are sensitive to this structure. There have been a variety of successful implementations devised by Elman [19], Pollack [20], and Smolensky [21] amongst others and some of these will be discussed in subsequent sections. Connectionists have also attempted to refute symbolists claims that all connectionist implementations equate to classical implementations.

It is necessary to first distinguish between the different types of connectionist representations before stating why connectionism offers a more powerful and psychologically plausible method of cognitive modelling. There are two types of connectionist representations, localist representations and distributed representations. A localist representation refers to a connectionist network whose individual units act as semantically interpretable symbols. The advantage of implementing a localist representation is that each unit is clearly labelled and so it is easy to see what its function is in the network. Therefore it is a popular method for researchers from the AI tradition. However, the major limitation of using such representations is that they provide no opportunity to share information amongst different entities and hence the representation is inefficient in terms of connections and units.

Distributed representations are connectionist representations in its purest form, where concepts are represented as patterns of activation distributed across a large number of units in a network. Distributed representations are commonly those representations automatically formed across hidden units within a feed-forward multi-layered perceptron (MLP) networks trained with the Back-propagation learning algorithm (see Appendix B). Back-propagation is a systematic method for training MLPs. Rumelhart et al[2a] presented a clear and concise description of the Back-propagation algorithm. The theory behind this solution is that the errors for the units of the hidden layer are determined by back-propagating the errors of the units of the output layer. Back-propagation can also be considered as a generalisation of the delta rule [70] for non-linear activation functions and MLPs. Distributed representations are not only more efficient in terms of resources but they also provide potential for presenting a novel object in terms of primitive features of other objects. For example, distributed representation can encode up to $2^n - (n+1)$ items whereas localist representations can encode up to n items i.e., one unit per item or concept. Distributed representations therefore require less memory than localist ones, and more distributed items can be represented per element vector.

Symbolic representations are disjoint and discrete, highly grammatical and concatenatively compositional. For example, it is always possible to tell whether a symbolic token belongs to a certain class, and it is possible to change the tokens in a composite symbolic representation without affecting the rest of the representation (thus these representations are concatenatively compositional). In contrast to this, distributed representations are typically continuous, non-grammatical and non-concatenative. The representation of a particular pattern class belongs to the representation of each and every other pattern class to a certain degree, depending on their similarities to other representations. This implies that the addition of another pattern/item into a distributed representation affects the representation of all other items. Therefore items can only be represented in the context of all other items. These properties makes connectionist networks very different from their classical symbolic counterparts, and also from localist connectionist representations. It is this property that provides connectionist networks with psychologically plausible behaviour and provides an account of human performance errors i.e., forgetting, memory confusions etc [23]. How can such human performance errors be modelled within a symbolic framework?

Smolensky[21] suggests that connectionism exists at a level between symbolic explanation and neural explanation (subconceptual level). He uses the term *subsymbolic* to refer to all connectionist distributed *microfeatural* representations.

The term micro-feature[7] refers to atomic elements in a distributed connectionist representation. Since the advent of algorithms such as the generalised delta rule or Back-propagation [2a], micro-features can automatically be formed within layers of hidden units.

This form of representation differs greatly from classical representations. Van Gelder[18] termed this superpositional representation in that many items are stored at once over the units by degree of activation. He also stated that connectionist representations possess functional compositionality rather than concatenative compositionality. A representation is functionally compositional when there are general processes for producing an expression given its constituents and for decomposing the expression back into those constituents. These composition and extraction functions, which operate on distributed representations, are a clear divergence from the classical theme that employs the use of functions like '*cons*' and '*car*' in LISP to compose and extract data from symbolic list structures.

Much connectionist research into the representation of variable binding and complex recursive data structures has been fuelled by the symbolic/connectionist debate. The following subsections further considers the key research that provides successful connectionist implementations of variable binding and complex data structures.

2.4.2.1 Connectionist Variable Binding

Pinker and Prince [32] state that rules, modular structures and variables are essential for modelling the human language faculty. Symbolic systems are easily capable of binding variables to values at run time. The values being bound may range from simple entities to complex recursive structures of arbitrary depth. For example, supposing we

[7] Although M^{c}Clelland and Kawamoto [4] use the term micro-feature to refer to individual elements which are semantically interpretable on their own (e.g is-soft,non-human,compact,rounded to represent the word *ball*). This type of microfeatural representation is symbolic in that each item refers to a property in the real world. The term microfeatural more commonly refers to individual units that cannot be semantically interpretable without further processing. Individual micro-features are non-symbolic.

have two entities, a red circle and a black square, we have to have a method of associating red with the circle and black with the square (e.g., COLOUR(obj1,red), SHAPE(obj, circle), etc); and this binding has to occur and cease dynamically when a system is reasoning with input data. Linguistic theory requires such bindings and complex data structures to be represented when implementing the theory, and this is easily achieved with symbolic techniques. So, how can this be achieved within a connectionist framework? Many leading connectionists [33,41,43] have addressed the problem of achieving this, but with limited success. It is important to review some of the key research in this area to assess the adequacy of integrating connectionist, rather than symbolic, variable binding techniques into a hybrid symbolic-connectionist architecture suitable for large-scale natural language parsing.

For explanation purposes, consider the following symbolic representation of knowledge regarding we disown what we sell (modified from Dyer [158]) :

SELLS(person1,object,person2) $\rightarrow$ DISOWNS(person1,object)

This rule is applicable for any object or person, as a result of the variables person1, object and person2 being bound to their appropriate values at execution time.

Localist Techniques

In localist CNs, each unit represents a given syntactic or semantic entity (e.g., a predicate such as DISOWNS, or a role such as SELLER) and the amount of activation on the unit represents how committed the network is to a given unit (or path of units) as the correct interpretation of the input. However, without some type of variable binding mechanism, localist CNs would have to represent, before execution, all possible binding combinations, which would lead to combinatorial explosion. To avoid this problem, current localist CNs use either *signatures* [34] or *phase synchronisation* [43] to propagate simple bindings.

A signature is a unique activation value assigned to a given entity and that serves as the value in binding. For example, the unit representing the entity Jack (person1) could be assigned the activation value of 7, which would be Jacks identity, while the unit representing the entity Jill (person2) is given the activation of 9 as its signature, and a unit representing some object, like a computer, given an activation of 11. To represent the above rule of selling and disowning, units are assigned to each predicate (SELLS, DISOWNS) and to each role (i.e.,SELLS:SELLER, SELLS:OBJECT, DISOWNS:OBJECT, DISOWNS:DISOWNER etc) and connections are setup between roles (e.g., from SELLS:SELLER, to DISOWN:DISOWNER) to specify how bindings should be propagated between predicates. Normal activation is propagated from SELLS to DISOWNS, while signatures are propagated along role-to-role pathways. If the signature 7 spreads from SELLS:SELLER to DISOWNS:DISOWNER this represents that Jack has sold a computer and now disowns it. Sun [33] and Lange et al [34] have successfully implemented signatures to propagate simple role-bindings.

Phrase synchronisation or *temporal synchrony variable binding* requires that the unit of time for each basic spread-of-activation step is broken down into a few, smaller subunits of time, termed phases. For instance, if there are five distinct phases within each spread-of-activation step, then a total of five distinct bindings can be propagated through a localist network. Each entity is assigned a unit of time or phase. The units in such networks are designed so that, when they receive activation within a given phase, they propagate it along their connections within the same phase of the next spread-of-activation cycle thus forming a pulse train of activation. So, the time phases represent the entities, activation levels represent the probability of a predicate being true and the temporal synchrony of a unit activation that is used to represent the bindings between predications. If two predicate units of the network are pulsing synchronously within the same time phase, then they are representing predications about the same entity, otherwise it is possible that they are representing predications about different entities. This temporal dimension is used to provide the third dimension which is needed for symbolic representations. Shastri and Ajanagadde [43] make use of temporal synchrony variable binding to propagate bindings in a localist CN to perform deductive reasoning, and Henderson [14] has developed a successful syntactic parser, based on this architecture, which will be discussed later in this chapter.

The advantage of using signatures is that an unbounded number of signatures can be propagated simultaneously along multiple pathways while temporal synchrony variable binding is limited to a maximum of ten simultaneous bindings. However, this limitation associated with phase locking is analogous to human performance in that humans can only remember sentences of a given complexity [43]. Also, for n phases there are *n* subcycles required for each spread-of-activation cycle, which slows down propagation rates by a factor of *n*.

However, localist networks that propagate phases or signatures have difficulty with propagating multiple instances of the same type. For example, consider the sentence, '*Jack asked Jill if Simon asked Sarah if Bob paid*'. In localist networks, each predicate is assigned a unit in the network, thus the clause *Jack asked Jill* can be represented by passing phases (or signatures) over the ASK:ASKER and ASK:RECEIVER units. A problem arises when the network is required to represent the embedded clause *Simon asked Sarah* which must be represented as another ASK instance that could be *dynamically* bound to the ASK:QUESTION unit of the top-level ASK. However, connectionists who construct localist representations overcome such problems by instantiating *n* units (copies) for each predicate and associated roles, i.e., if n=2, then the network could represent one ASK instance embedded within another ASK. Such representations are inadequate if the level of embedding is greater than *n*, but if the depth of recursion is greater than which a human could understand then the localist system is adequate for modelling human parsing.

Although the localist methods of variable binding reduce the problem of combinatorial explosion associated with representing rule-based and inductive reasoning such systems are hand-crafted. The major disadvantage associated with this is that amendments to the system is cumbersome and the more complicated the domain being modelled the more opaque the localist system to model it will be and thus reduces its suitability for realistic subsets of natural language. As there is no learning in such systems it is difficult to perceive how localist approaches can capture the linguistic generalisations and exceptions that occur in naturally occurring language.

Distributed Techniques

Distributed connectionist systems do allow learning and thus allow structured bindings to be created dynamically either by modification of connection weights or by changes in patterns of activations over ensembles of connectionist units. The use of distributed patterns supports the dynamic creation of a potentially exponential number of possible values. At present, a method that has been developed for representing and propagating distributed patterns of activation as role bindings is tensor products.

Tensor product variable binding was defined by Smolensky [41] and is a formulisation of the idea that a set of value/variable pairs can be represented by accumulating activity in a collection of units each of which computes the product of a feature of a variable and a feature of its value. The method allows the fully distributed representation of bindings and symbolic structures. It organises the space dimension into multiple representational dimensions, one for properties, and the rest for entities. In the context of linguistic structure, it combines lexical items (words) with their syntactic roles (e.g., noun) and is mathematically equivalent to outer product learning (Sharkey[42]). A vector representing an element (or role filler), e, is bound to a vector representing a role, r by the outer product er^T. This is a tensor of rank two and results in a square matrix of activations. However the formalism goes beyond the simpler outer product in that it enables the construction of recursive representations for syntactic trees by using 3rd, 4th or nth order tensors.

A third order tensor is a cube of unit activation and orders beyond the third are hypercubes. Representing properties (words) which have multiple roles (noun, verb) requires as many additional representational dimensions as there are roles. It therefore uses as many dimensions for entities as the maximum number of roles for any property.

The disadvantages of using tensor product representations is that the network could grow unfeasibly large and also, when the input vectors (the fillers for the roles) are not orthogonal the tensorial representations have to be constructed by more complex incremental learning methods (e.g., Back-propagation). It is for these reasons that Smolensky advises that further analysis is required to examine the consequences of discarding binding units, and other means of controlling the potentially prohibitive growth of the tensor representations. He also stated that the relations between tensor product binding units and connection weights need to be pursued.

The development of connectionist variable binding techniques have provided valuable insight into how symbolic computation can be performed within localist and distributed architectures. However, it is unclear as to whether the success achieved will be as successful with realistic natural language domains and further research must be undertaken to examine the computational power and generalisation capabilities of such techniques. At present, well-established symbolic techniques provide easily manipulative and interpretative methods of dynamic variable binding.

2.4.2.2 Connectionist Representation of Structure

Representation of recursive and compositional structure is important if connectionist networks are to successfully model high-level cognitive tasks such as language processing. For example, sentences can theoretically contain an unlimited number of embedded clauses which are represented as recursive rules in a grammar, and can easily be symbolically represented as a deep-nested tree structure.

There have been numerous distributed approaches to explicitly encoding compositional structure within connectionist networks. Distributed approaches are largely based on multi-layered perceptrons trained with Back-propagation. The methods employed in BoltzCONS [35] explores how the manipulation of distributed representations can be functionally analogous to symbolic lists and trees, but BoltzCONS provides little advantage over symbolic techniques, Tensor Products [41] require exponential growth in the number of units used to represent structure as the depth increases. The more powerful distributed approaches are based on recurrent connectionist architectures and convolution-based processing, such as the method employed in Pollock's RAAM [20], Kwansy and Kalman's SRAAM [36], and Plate's Holographic Reduced Representations (HRRs) [37] which all provide methods of representing recursive and compositional structure within fixed-width vectors.

Recursive Auto-Associative Memory (RAAM) was developed by Jordan Pollack [20] to address the problem of structural representation and manipulation within connectionist networks. RAAM represents variable-sized recursive data structures in fixed width patterns and provide a vehicle for representing symbolic structures that supports compositionality. The RAAM architecture is based on the standard feed-forward network trained with Back-propagation and consists of an encoder and decoder. The encoder encodes two patterns into one, and is recursively applied. For example, for the binary tree ((AB)(CD)), A and B would be compressed into R_1 via auto-association. R_1 is the hidden unit representation or response vector for A and B and is stored on a symbolic stack. C and D would then be compressed into R_2 in the same manner, then R_1 and R_2 into R_3. R_3 is now the hidden unit representation for the whole tree. These hidden unit vectors could be termed *reduced descriptions* of the current input data. The decoder decodes compressed patterns and determines whether the decoded parts should be decoded further. For example, R_3 would be decoded into R_1' and R_2', R_1' would be decoded into A' and B', and R_2' into C' and D'. The system uses a feed-forward auto-associative network with three layers: an input, one hidden and an output layer.

Pollack demonstrated the ability of RAAM to learn well-formed syntactic parse trees using a simple context-free grammar, for example a sentence such as *The beautiful girl picked up the flower* would have the structure ((*determiner* (*adjective noun*)) (*verb preposition* (*determiner noun*)))). Once the network had successfully trained, representations for all subtrees were accessible. The hidden layer representation formed in the RAAM architecture from the training set was able to distinguish from grammatical and ungrammatical sentences. The cluster analysis of the hidden layer representations identified further similarities, such as the identification of phrase types and tree depths. Pollack reported that poor generalisation performance but he discussed how improving the training environment, and reconsideration of the mathematical principles of connectionism would improve the results.

Chalmers[44] further analysed the RAAM architecture and its distributed representations in order to examine its sensitivity to structural changes and syntactic transformations. Chalmers claimed that to get beyond an re-implementations of symbolic processing he stated that operations other than composition and extraction must be used. He illustrated how RAAMs can be used to model syntactic transformations by operating *directly* on the compressed distributed representations of sentences, without emerging to the symbolic surface via going through composition and extraction stages.

Although the experiments with RAAM have displayed encouraging results, it is not clear what the range of applications and limitations of RAAM are. RAAMs can learn distributed representations of recursive compositional structures and have already been predominantly used to encode parse trees in some connectionist parsers [62, 63, 126, 127]. Plate [37] states that there are numerous problems with RAAMs and the most serious are that : they require long periods of training; the types of structures that RAAMs can represent are inflexible as the number of fixed roles is limited and there is no way of representing similarity between different roles; the depth of trees that can be encoded are severely limited; and the generalisation and scaling properties of RAAMs are largely unknown.

Kwansy and Kalman [36] addressed the most serious problems of RAAMs i.e., the long training periods and the fixed valence of RAAM structures. They provide a technique for mapping any ordered collection (which they term *forest*) of hierarchical structures (*trees*) into a set of training patterns which can be used effectively in training an Elmans SRN [19] to develop RAAM-style distributed representations. This architecture is referred to as *Sequential RAAM* or *SRAAM.* SRAAM eliminates the constraint of fixed-valence structures as symbols can be presented to the system sequentially. Also, the representations resulting from training correspond to ordered forests of labelled trees thereby extending what can be represented in this way. The third advantage of using SRAAM over RAAM is that training is a straightforward process since training is accomplished with an auto-associative SRN. In their experiments, Kwansy and Kalman also reported a higher generalisation rate than RAAMs. Callan and Palmer-Brown [158] also addressed the problems of training and poor generalisation associated with RAAM with their *Simplified RAAM* or *(S)RAAM* model. (S)RAAM uses a fast analytical training process based on principle component analyses (PCA) and once trained exhibits higher levels of generalisation than standard RAAM. However, RAAM and its variants have not yet been shown to scale-up effectively for large complex natural language training sets.

Tony Plate [37] devised Holographic Reduced Representations (HRRs) for representing associations between items (i.e., variable binding), and complex recursive structures within fixed-width vectors in a similar manner to Smolensky's tensor-products representation for prepositions, but Plate uses *circular convolution* to associate roles and fillers, rather than the tensor product. Circular convolution can be viewed as a compression of the outer (or tensor) product of the two vectors i.e., reduces two vectors to a single vector. The advantages of this approach is that the dimensionality of representations remains constant rather than increasing exponentially with the depth of composition as with tensor products. *Circular correlation* is used to decode the convoluted vectors i.e., decode a single vector into two separate vectors.

There are numerous disadvantages of using HRRs the most important being the large size and random nature of base vectors as this makes it practically impossible to use them as inputs or outputs to networks trained with Back-propagation. Connectionist networks based on Back-propagation learning normally use binary vectors for input and output representations, but HRRs will need to learn mappings between real numbered vectors that range between -1 and 1, thus the random nature of base vectors will make it difficult for the networks to capture statistical regularities within them. Also, the size of the vectors are normally large and this extends memory usage and training times.

HRRs are an important advancement in techniques for representing compositional structure, but it is a complex method which is not easily integrated with common connectionist networks trained with Back-propagation. Also, due to the nature of circular convolution it is questionable as to whether this is truly a connectionist approach to structural representations as there is no learning involved in the processing. HRRs are distributed models of representation but it is arguable as to the need of a connectionist network when using HRRs, why not just use circular convolution and circular correlations? It appears to the author that further work needs to be undertaken to assess HRRs applicability to connectionism and its integration into true connectionist architectures.

It is evident that at present the development of connectionist representations of complex structure is in its infancy, and has yet to be tested on large-scale natural language domains. The problems of intractable training sets and unrealistic computation times will be faced when such experiments are being undertaken by researchers. However, as the development of VLSI technology advances and faster computational devices are available it is envisaged that further successful developments will be made in this line of research.

2.4.3. Localist Parsing Models

The first attempts at parsing with connectionist networks were based on localist techniques. The parser developed by Small [38] and then extended by Waltz and Pollack [128] assigned concepts and lexical items to units within the network and used inhibitory and excitatory links to allow syntactic and lexical-semantic constraints to interact in order to determine the result. A symbolic chart parser is also used to aid this task. Fanty[9] also integrated localist representation techniques with the chart parsing method. He used a localist representation of a fixed chart. A fixed chart that includes all possible parses for all possible sentences could be specified if a fixed length restriction was made on the input sentences. A combination of top down and bottom up activations are used to determine which constituents in the chart are parts of the possible parse. To select among the possible parses a winner-takes-all network is used. However, a very large network would be required for sentences of reasonable length and hence also for complex grammars.

Cottrell [24] investigated the use of connectionism for word sense (lexical) disambiguation in 1985. The parser's grammatical framework uses context-free grammars which include both syntactic category and semantic role information. The network uses a localist representation of this information. There are dedicated units for each possible category-role pair, and dedicated units for each list of roles which can form a constituent of a given category.

There are multiple copies of each of these types of units, to allow for multiple constituents of the same type. While this representation specifies each of the context-free productions used in the parse tree of the sentence, it is possible for this representation to be ambiguous as to how these pieces fit together in the complete structure. Cottrell attempts to incorporate temporal order in which these units become active to represent which constituents have already been parsed and bound to a position in the phrase structure tree. He reported little success and referenced Fanty's work for possible resolutions to such problems. Also, the parser does not represent features of constituents other than their main syntactic category. The majority of the tests he performed were on simple sentences.

Selman and Hirst[10] employed the Boltzmann machine for syntactic parsing. A localist representation was used and a restriction placed on the input sentence length. The inherent limitation of this method is similar to that of the previous two in that the network size increases dramatically in proportion to the grammar complexity, and the input length is restricted.

More recently, localist parsers developed by Henderson [14] and Stevenson [111] have illustrated that localist techniques are powerful enough to model human syntactic processing and to some degree process realistic natural language domains without learning.

Henderson developed a syntactic parser, termed the Neural-Network Node Equating Parser (NNEP), based on Shastri and Ajanagadde's [43] temporal synchrony variable binding architecture. The NNEP has three basic parts, input units, predicates units, and grammar units. The temporal pattern of activation over the predicate units represent information the parser needs to know about its parser state. The phases in this pattern of activation represent variables that refer to non-terminal units in the phrase structure of the sentence, and the predicates represent properties of these units and of the phrase structure tree as a whole. Henderson states that since these predicate units represent a feature decomposition (e.g., *has_parent, has_verb*) of the types of units, the predicate units are analogous to hidden units within Back-propagation trained MLPs. There is one input unit per word, but there are also input units for another word to aid lexical disambiguation. When the next word is input, that word's unit becomes active and stays active until a grammar entry for that word is combined with the parser state. Connections from these units go to units for the grammar entries of the word. These primary connections are inhibited by connections from the predicate units. Henderson explains that these inhibitory connections filter out all the phases for phrase structure units which cannot be combined with the grammar entry. Therefore, connections between the input and predicate units, and inhibitory connections from the predicate units implement the patterns for pattern-action rules that calculate what action the parser should take given the parser state and the next input word. The parser outputs incremental descriptions of the sentence structure (see Marcus et al's D-Theory [123]). Henderson demonstrated that the NNEP was able to parse sentences taken from the Brown corpus at random.

The NNEP inherits the limitations found with phase synchronisation i.e., a maximum of ten simultaneous bindings. Henderson defends this by stating such limitations are analogous to human memory degradation when the number of embeddings within a sentence becomes incomprehensible. However, during natural language understanding, many new bindings can arise dynamically that are easily understandable by humans even though there are more than ten bindings. For example, Henderson's NNEP would not be able to parse sentences such as, *The rich bespectacled man took an expensive, greedy risk for the web browser market*, as over ten bindings are required. Another limitation is that there is no explicit representation of coordinating conjunctions. Henderson also uses space more than once for representing relations between variables thus it requires more units and it adds complexity in modifying and accessing the stored relations. Processing with relations between variables is difficult because such rules involve more than one entity. Since rules only have access to information about one word or constituent at a time, information that the rule needs about other word or constituents must be represented as properties of the structure as a whole. Recently, Henderson and Lane [161, 162] have integrated SRNs and thus distributed techniques with phase synchronisation to allow learning within the NNEP model. This represents a shift towards the integration of localist and distributed representations to improve the computational adequacy of such systems. Although improvements in language coverage have been reported, the experiments have been based on simple context-free grammars and small subsets of the Brown corpus [144].

Stevenson [111] developed a localist parser based upon spreading activation to represent parse trees, and to establish syntactic dependencies between units through local communication. Stevenson's parser, CAPERS, is based on Government-Binding theory and encodes GB principles as a set of simultaneous local constraints. CAPERS was developed to model human processing of long-distance dependencies and integrates simple symbolic processing with numeric spreading activation, within a competition-based spreading activation (CBSA) [129] network that directly represents a parse tree. Layers of syntactically labelled units are interconnected, and must actively determine their structure by trying to attach themselves together to form a valid parse tree. Valid syntactic relations among the phrasal units are established incrementally through feature-passing. The communication method used relies on an integration of symbolic and numeric (activation) constraints on passing features through the parsing network. It is the feature-passing algorithm that enables the system to establish long-distance syntactic relations using only local communication within the network. The communication of symbolic GB-based features through the network determines all the valid attachment structures; numeric competition is necessary to focus activation onto a winning subset of the attachments that form a legitimate parse state. Stevenson reports adequate modelling of human processing of long-distance dependencies, however little information was provided as to the failings of the model and the scope of language the model could capture.

The inherent problems associated with localist representations will always be a significant obstacle for localist parsing models when posed with a realistic subset of natural language. Localist models are inefficient in terms of the number of units and connections required to model a system as no opportunity to share information amongst different entities is allowed. For example, multiple copies of units are required to represent multiple instances of the same type. Although the localist methods of parsing are easily interpretable, the rules of the language being modelled are hand-crafted, therefore amendments to the system is highly coupled with the grammatical formalism used and the more complicated the domain being modelled the more opaque the localist system to model it. Also, as there is no learning in such systems it is difficult to perceive how localist approaches can capture the linguistic generalisations and exceptions that occur in naturally occurring language.

2.4.4. Distributed Parsing Models

Early distributed models of language processing inspired much research into connectionist parsing using strictly feed-forward MLP architectures trained with Back-propagation. In 1986, Rumelhart and McClelland's [2b] connectionist model of learning the past tense of verbs demonstrated that the MLPs trained with Back-propagation could exhibit rule-like behaviour without being presented with any explicit symbolic rules during training. A verb was presented to the MLP as input and the past form of the verb as the target output. An important property of the network, once it had been trained on a set of verb/past-tense verbs, was its ability to over-generalise in similar ways to children e.g., the network stating that the past form of the verb, *come*, is *comed* rather than *came*. In the same year, McClelland and Kawamoto [4] developed an MLP model that could learn Case-role [92,140] assignment in sentences and utilises the ideas of semantic grammars to perform semantic processing. Each act can be described by the main verb and a set of semantic cases such as agent, patient, instrument, and recipient. Semantic micro-features were defined to represent semantic information about words. The MLP was required to associate the words of the sentence with these roles.

However, a significant limitation of both models was the manner in which the input was presented to the networks. The 'Wickelphones' in Rumelhart and McClelland's model and the input sentence in McClelland and Kawamoto's model was presented to the network as a whole therefore a pre-determined length was placed upon the input. This is mapping time onto space in that rather than processing each word sequentially each word or utterance of the sentence is presented together in a parallel fashion. The maximum sentence length must therefore be pre-determined so that the number of input units can be established. Sentences above the maximum length cannot be processed. These limitations were reflected in early distributed connectionist parsers with fixed length restrictions on the input sentence (Hanson and Kegl [11], Howells [12]). It is evident that when reading text and in speech understanding, the input is structured in time and hence linguistic input is commonly processed sequentially, i.e., one word or constituent at a time [19]. How to perform temporal processing with CNs therefore became a major focus of subsequent distributed connectionist parsers.

Temporal Processing with Sliding Input Windows

To overcome the problem of fixed input constraints, there have been further developments with distributed models which represent a restricted part of a sequence spatially. For example, sliding input windows or *temporal input windows* could be used to represent sequentiality in feed-forward connectionist networks, in that rather than presenting the whole sentence to the network, the network can process a fixed number of constituents per time step. This enables such networks to perform a type of n-gram statistics. By moving the window across the whole sequence, sequences of an unrestricted length could be processed using an MLP architecture. This also allowed context in MLP systems to be taken into account. Numerous distributed connectionist parsers, for a variety of natural languages, have been developed using this technique (Archambault [130], Apolloni [131], Kwansy [55]). Such parsing architectures have been closely coupled with some grammatical formalism. However, the major limitation of using temporal input windows is that temporal context is limited to the size of the input window, thus if the contents of the input window cannot be disambiguated then the parse will fail on the current input.

Temporal Processing with Recurrent Networks

When applying the feed-forward MLP architecture with Back-propagation learning to the language domain the problem of adequately representing temporal sequences has therefore been a major obstacle. Connectionists [19,28,29,132,133,134,135,136] have thus extended the MLP architecture to incorporate some form of memory system in order to cope with temporal sequences and relations. These connections put cycles in the spread of activation through the network, thereby allowing the network to store state information. Words can then be input into the network sequentially, and the network's state is used to represent how the previous words constrain the processing of the current word. In some cases this should provide connectionist networks the ability to process or represent arbitrarily long sentences. The most powerful and influential recurrent MLP is Elman's Simple Recurrent Network (SRN) [19].

The SRN was developed by Elman to represent time by the effect it has on processing rather than additional dimensions of the input. This means that the system has dynamical properties that are responsive to temporal sequences i.e., it has a memory. The network architecture is a feed-forward MLP augmented at the input level by additional units called *context units,* and input symbols can be presented sequentially to the network. The context units are hidden in that they do not explicitly interact with the outside world. Context units are initially set to 0.5 and both the input and context units activate the hidden units; and hidden units then activate the output units. The hidden unit activations are also fed back to activate the context units and represents an internal reduced description of the input representation [132]. Depending on the task there may or may not be a learning phase in a time cycle. If there is a learning phase then the output activations are compared with the target and Back-propagation is used to adjust the connection strengths. Recurrent connections are fixed at 1 with no adjustment. At time *t+1* the training sequence is repeated but the context units now contain values which are exactly the hidden unit values at time *t*. Context units serve to remember previous internal states and thus provide a memory. Hidden units develop representations which are useful encodings of temporal properties of the sequential input. This internal representation is sensitive to temporal context. Elman demonstrated that SRNs could learn to predict language sequences based on simple context-free grammars. Once an SRN has learnt valid sequences from a grammar the network implicitly encodes syntactic and semantic structure within its weights. This type of implicit and internal distributed representation of hierarchical structure provides a powerful architecture that is well suited to natural language processing tasks and is able to learn linguistic knowledge by example.

Since the advent of the SRN, numerous distributed connectionist parsers have been presented by researchers with varying degrees of success. Cleeremans, Servan-Shreiber and McClelland [137] demonstrated that SRNs can learn to be a perfect finite-state recogniser for a particular finite-state grammar via exposure to valid strings of variable length from the grammar. It was also proved that SRNs could carry information regarding cross-serial dependencies, although this information is lost if intervening constituents do not depend upon it. However, Cleeremans et al defend this by stating that embeddings in natural language are not completely independent of earlier information. Moisl [138] also trained an SRN to act as a finite-state automaton. In his parsing model, three networks are used : an MLP to compress word vectors via auto-association; an SRN to act as a deterministic finite state transducer (FST) in order to associate compressed output symbols with the compressed representation of the input symbols; and another MLP net to decompress the output of the SRN back to its symbolic form (i.e., from hidden unit vector to back to original bit vector). The MLP net is termed the input net. Each input symbol (i.e., a word from grammar, or non-terminal symbol or bracket) is a bit-map representation consisting of 15 bits per letter/symbol. A maximum of 5 letters per word is allowed. Auto-association is used to reduce this 75-bit input representation. Initially, this net is trained and thus the hidden unit activations produced in response to the current word is passed as input to the SRN. The SRN is the FST and takes as input the MLP's hidden unit vector for the current input word and a look ahead word. The output of the SRN is the relevant output symbol associated with the current input and a letter denoting some parser action like shift or stay put. These output symbols are in their reduced form therefore another MLP network, called the output net, is used to decompress this back to its original bit form. So the output net, during training, uses auto-association to reduce the representation of an output symbol and a parser action. Once trained, only the second half of the network is used i.e., hidden to output. It is obvious that the input net and output net must be trained before the SRN. Training and test data is based on a simple context-free grammar. The network was tolerant to mildly ungrammatical sentences i.e., number disagreement, and was able to process corrupted strings e.g., misspelt words. However, Moisl reported poor rates of generalisation, and novel sentences which the system was able to process were closely related to the training set, thus it is questionable as the type of linguistic generalisations that were acquired by the network.

Berg [63] augmented an SRN with a RAAM architecture to formulate Xeric, a single connectionist architecture capable of processing recursive sentence structure and lexical disambiguation. The input layer contained a word and the previous hidden context, this information was then fed into the RAAM architecture, whose hidden state is copied back to the context units, and the output layer represents an X-bar template in order to use GB-related principles. When parsing, Xeric begins with all of the context units reset to a predetermined value. The lexical encoding of a word is presented to the network and the activation feeds forward through the network. For each subsequent word in the sentence, the lexical encoding of the word is presented as input along with the hidden unit pattern from the previous time step. After the last word has been processed, the hidden units represent the entire sentence. The sentence structure can be obtained explicitly by recursively using the RAAM decoder segment of the network. Although the parser impressively learnt sentence structure of a particular length, the overall parsing error increased in proportion to the depth of the embedding encoded. Berg explained that this was due to the difficulty Xeric had in reproducing the ID-code of each word. He suggested this was due to either the reduced description of the sentence being limited in the amount of information it could represent. It is evident that the limitations associated with RAAM is inherited by Xeric and until further advancements are made with RAAM-style representations then this will always be a fundamental problem with Xeric. Also, Berg did not provide an accurate account of the error rate on the output units i.e., how many words were reproduced incorrectly and it is difficult to assess its ability to scale up to large natural language domains. At present Berg has only used a simple grammar and only simple sentence structures were processed, even though it can in principle process arbitrarily long sentences which is an advancement from other connectionist parsers.

Perhaps a major limitation associated with the above distributed parsers is that the task of parsing is being modelled using only a single connectionist architecture. This restricts the size and complexity of the grammar being modelled. If the grammar used is large and complex then the size of the network and the resulting training sets required will become computationally unfeasible and intractable. It is therefore of no surprise that distributed connectionist parsers based on a single architecture are constrained to using 'toy' grammars[8], fixed length input constraints, and an inability to successfully deal with certain syntactic phenomena such as centre embedding, constituent coordination and long-distance dependencies. This has forced connectionists to concentrate on simplifying the parsing problem into separate modules, and assigning appropriate connectionist architectures to perform its own dedicated task.

Reilly [62] developed a distributed connectionist parser on this basis but with limited success. Reilly's parser trained an SRN to build RAAM representations as it receives each input item. Each input item is the syntactic category of the word being processed. The SRN processes each syntactic item until a phrase can be encoded within the RAAM i.e., once the SRN had received determiner and a noun, it would pass this input to the RAAM which could then encode it as a noun phrase. The RAAM hidden layer representation of the noun phrase can then be passed back as input to the SRN. Reilly reported that the system could not learn all of the training examples, and that generalisation levels of the RAAM was poor. A toy grammar was also used, and the RAAMs would have to be trained on a sufficiently large selection of possible parse trees for it to be capable of fully representing the parse trees of the grammar adequately, and then train the SRN to reproduce the parse trees corresponding to a selection of sentences. Even with the toy grammar this would prove computationally expensive, which is evident as the system could not learn all training examples and sufficiently generalise to unique sentences with the training data used.

A more promising modular distributed parsing model is that developed by Sharkey and Sharkey [126]. The parsing model consists of three networks : an SRN to predict the next word in the input sequence and implicitly encode the sentence in its weights; an MLP to pass the SRN hidden layer representations to the RAAM; and finally a RAAM to encode and decode parse trees for a range of sentence structures which included embedded clauses, relative clauses and prepositional phrases. The SRN is the encoder in that the input is some word representation. A limited vocabulary was used i.e., 9 actions, 22 inanimate objects, 7 prepositions and 34 verbs. The encoding scheme was not given. The

[8] Toy grammars refer to those grammars which are typically context-free and consist of a small number of rules which govern the formation of grammatical strings but whose expressive power is weak and unable to realistically represent the natural language. However, such grammars are useful for generating arbitrary length sentences which are a subset of the natural language and may contain specific syntactic problems such as centre embedding and long-distance dependencies in order to train or test a system for assessing its computational adequacy.

SRN reads in all input words sequentially until the end of sentence terminator is encountered. The output of the network is simply the next input word - i.e., its attempting to predict the next valid word. Once trained, the final hidden unit representation encodes the whole sentence structure implicitly. Sharkey demonstrated this by performing a hierarchical cluster analysis on the hidden unit activations. The analysis showed that words belonging to similar syntactic types were grouped together - similar to the Elman's results [19,74]. The MLP is like a transformer in that it takes the SRN's final hidden unit activations as input and transforms it to the RAAM representation of the appropriate root node for the sentence fed to the SRN. The RAAM has a fixed valence of three and all the training examples have been encoded within it. Each module is trained separately, the SRN first, then the RAAM and finally the MLP. The whole tree can be recursively decoded by the RAAM to obtain the whole parse tree. After the modules have trained, they are combined so that the final layer of the SRN is replaced by the hidden and final layers of the MLP, thus the resulting architecture directly takes input words and produces RAAM representations of the parse tree at the end of processing.

It is evident that the task of each network was made simpler by breaking the parsing process into different sub tasks, and the compound results of the distributed parsing system shows more promise than previously mentioned systems. Sharkey reported that the modular parser was able to parse the 1280 sentences for which the components had been trained, and it correctly disambiguated 75.62% of a set of 320 novelly structured ambiguous sentences, which is impressive considering the system developed its own lexical rules and had no access to semantic and pragmatic information other than that implicit in the sample, consider the sentence, *The policeman chased the boy with the limp.* The parser attaches the preposition to the noun phrase (*the boy*). However, when Sharkey replaced the word *limp* with *truncheon*, the parser was able to reattach the preposition phrase to the verb (*the policeman*) because of this lexical alteration. So the inherent representation of word groupings enabled structural changes to be made. However, although the modular approach is evidently more powerful and manageable than singular approaches, Sharkey still used a toy grammar, and the complexity of the sentences it can process will always be subject to the limitations of the memory capacity of the RAAM network, as is Reilly's parser.

Jain and Waibel [166] (also see Jain [167, 168]) developed a novel modular recurrent connectionist architecture, PARSEC, based upon Back-propagation learning to perform incremental parsing of spontaneous speech of a limited knowledge domain. The architecture is a composite of three separate recurrent connectionist networks each trained to perform a portion of the parsing task on a training set containing over 200 sentences extracted from a left-associative context-free grammar. The parsing model contains five processing levels : *Word*, *Phrase*, *Clause Structure*, *Clause Roles* and *Interclause*. A word is presented to the input layer of the network via activating a word unit. This produces a pattern of activations across the hidden (or feature) units which represents the meaning of the word. The connections between the word and feature units encode semantic and syntactic information about words. The Phrase level uses the sequences of word representations from the Word level to build syntactic phrases. Connections from the Word level to the Phrase level are modulated by "gating units" which learn the required structural assignment behaviour. The Clause Structure level maps phrases into constituent clauses of the input sentence. The Clause Role level labels the phrases of the clauses with semantic roles and attaches phrases to other phrases. For each clause there is a set of units that assign the role labels to phrases and further units to indicate phrasal modification. The final level, Interclause, represents the interrelationships among clauses. The Clause Structure and Interclause levels are trained simultaneously as a single module. This module contains *mapping*, *hidden* and *labelling* units. The mapping units assign phrase blocks to clauses; the labelling units indicate relative clause and a subordinate clause relationship; and the mapping and labelling units are recurrently connected to the hidden units which also have input connections from the phrase blocks of the Phrase level. The submodules were separately trained and combined into a singular connectionist architecture to perform the entire parsing task.

Although Jain and Waibel's model represents a tightly coupled modular connectionist architecture for parsing and assigning case-role assignment to real spoken language tasks, little information was provided about the grammatical formalism used and the number of case-roles allowed. It was unclear as to how many of the 200 training sentences were learnt and how many test sentences were used and also how similar the test sentences were to the training corpus. Although impressive results were claimed it is difficult to assess the nature of these results and adequacy of the architecture. Buo, Polzin and Waibel [169] extend PARSEC to enable the system to learn to parse sentences into case frames annotated with syntactic feature-values. Again, little information as to the nature of the training and test data was provided. The JANUS project [172] utilised PARSEC to parse limited speech dialects. More interestingly, a derivative of PARSEC was also presented by Buo et al [169] that attempted to use two Back-propagation trained architectures to approximate the behaviour of a symbolic bottom-up shift-reduce parser. The first network is responsible for identifying phrase boundaries and another is required to label these phrases. An X-bar based grammatical formalism was used to generate over 100 training sentences and the parsing model was reported to have 'generalised well'. Again, inadequate information was provided regarding the architecture and also the composition of the training and test sets. However, it is clear that this architecture uses a simple context-free grammar and is predisposed to the grammatical framework used and thus inherits the limitations associated with the use of symbolic grammar rules for training and test data.

A similar modular parsing system to PARSEC is the SCREEN system developed by Wermter and Weber [163]. The ultimate aim of SCREEN, however, is to move away from domain-dependent speech parsing and towards domain-independent speech parsing by maintaining a level of generality within the architecture such that it can be applied to unrestricted domains. SCREEN uses a hybrid connectionist architecture to exploit the generalisation properties of connectionist networks to process the typical complexities associated with spontaneous speech e.g., false-starts, hesitations and incomplete utterances. Although Wermter and Weber stress that the fundamental intention of SCREEN is to perform incremental fault-tolerant parsing of speech, speech input is not actually presented to the system. Real utterances extracted from a corpus are transcribed and therefore converted to text before being presented to SCREEN. However, the corpus is representative of spoken language and contains naturally spoken constructions and associated speech errors. SCREEN contains five core modules : *Speech Interface, Category, Correction, Subclause* and *Case-Frame*. Each module consists of a connectionist network and a symbolic interface.

The Speech Interface module receives input from a speech recogniser as a word and provides an analysis of the syntactic and semantic plausibility of the recognised words. The Category module receives words of an utterance and yields basic syntactic and semantic categories for each word. The Correction module receives knowledge about words and phrases as well as their categories and provides the knowledge about the occurrence of a certain speech error such as repair or repetition. The Subclause module is responsible for the detection of subclause borders in order to distinguish different subclauses. Finally, the Case-Frame module takes the syntactic and semantic categories of each utterance and assigns the appropriate case-frames to each phrase. This module is responsible for the overall interpretation of the sentence. Wermter and Weber demonstrate the category, correction and case-frame modules of SCREEN in [163], particularly focusing on syntactic analysis. These three modules used SRNs to learn their dedicated tasks from a training corpus containing 37 utterances with 394 words (including errors). The test corpus contained 58 utterances consisting of 823 words. Each network consisted of 13 input units. The category module learnt 99% of its training data and generalised to 93% of its test data with 14 hidden units. The correction and case-role modules learnt 91% and 93% of its training data respectively using 7 hidden units and generalised to only 85% and 89% of its test data respectively.

Although SCREEN demonstrates the ability of modular hybrid architectures to learn to parse spoken language without the use of rule-based grammars for training, the corpus used was limited in genre and domain thus resulting in a low coverage of spoken language being used to assess the adequacy of the system. It is therefore unclear as to the practicalities of successfully training such a highly modularised system with a realistic coverage of language extracted from a number of diverse domains. Also, is there a limit to the number of allowable case-frames and roles? The system-level generalisations made for an utterance is also considerably lower than the module-level generalisations. This could imply that the module architectures could be improved or that further training strategies are required to allow more complete learning of the training data. However, SCREEN makes a significant contribution to fault-tolerant speech processing systems that are trainable and able to sufficiently generalise to process unfamiliar input utterances.

Miikkulainen [127] developed a modular distributed connectionist parser that also used case-role assignment and that was able to provide higher levels of generalisation than Sharkey and Sharkey's parser. Miikkulainen's parser, SPEC, is a descendant of his previous attempt CLAUSES [139], which was unable to generalise sufficiently well to new sentence structures due to rigid case-role output representations and excessive context sensitivity. SPEC consists of three networks : an SRN, called the Parser; an MLP network, called the Segmenter; and a RAAM to store clauses. The input to the Parser is the current word and the output is a case-role vector which allows a maximum of the three roles e.g., agent, object, action etc. Each time it receives a valid clause it will fill each role with the appropriate word. It is the collection of these case-role vectors that is the parse for the sentence i.e., in the sentence, *the man who saw the dog that bit jack was tired*, the case-role output would be :'*man saw dog*','*dog bit jack*' and '*man was tired*'. To avoid the Parser processing unwanted context the task of the Segmenter is to take the Parser's previous hidden state and modify it if the Parser is entering a new clause. The Segmenter therefore takes the Parsers previous hidden state and the next word (as look ahead) as its input, and if the next word is part of the current phrase it will leave the Parsers hidden state unchanged, however if it is part of a new clause it will transform the hidden state into the Parsers hidden state for the previous words that relate to the current clause. For example, consider the example sentence above. When the Segmenter receives the word '*who*', we know that '*the man*' has been processed and this context is important for processing the '*who*' clause, therefore it will leave the Parsers hidden state unchanged. But when the Segmenter encounters '*that*' it will produce the Parsers hidden state for '*the dog*' as this is the only important context.

Miikkulainen trained SPEC with a toy context-free grammar which was able to generate a corpus of sentences with relatively complex clause structures. Miikkulainen reports higher levels of generalisation than Reilly and Sharkey, but again a toy grammar is used as the fundamental basis of the system. The major limitations of his parser are the inherent memory constraints imposed by RAAM and the fixed number of case-roles used as his output representations. SPEC appeared complex to train even with a toy semantic grammar and few case-roles and omitted general syntactic processing. It is therefore unclear as to whether this model, and indeed the other modular parsing models, will scale up and be trainable for realistic language domains.

Buo and Waibel [170, 171] developed a system, called FeasPar, which attempts to learn to parse spontaneous speech by integrating connectionist networks with a symbolic search. The connectionist networks split incoming sentences into chunks and label these chunks with feature values (based on GPSG principles) and uses a symbolic search to find the most probable and consistent feature structure. A minimum of hand-modelling is required compared to conventional LR-parser, and the connectionist networks are expected to learn the parsing task. The parser is trained and evaluated on a domain orientated language. The *Chunker* module splits an input sentence into chunks. It consists of three feed-forward MLP networks. The first MLP network finds and recognises the difference between ordinal and cardinal numbers and presents them as words to the following networks. The next MLP network groups words into phrases. The third MLP network groups phrases together into clauses. There are consequently four levels of chunks : word/numbers, phrases, clauses and sentence. The *Linguistic Feature Labeller* module attaches semantic feature labels and atomic feature values to these chunks. The connectionist architecture used is tailoured to classify a particular feature at a particular chunk level. This module is tightly coupled with a symbolic syntactic-semantic lexicon for lexical look-up. The *Chunk Relation Finder* determines how a chunk relates to its parent chunk and it has one network per chunk level and chunk relation element. It is the addition of these relations that forms the full parse for a given sentence. Although FeasPar faired favourably over a comparable symbolic GLR-parser, it is a complicated system that is tightly coupled with the grammatical formalism and difficult to train and maintain. It is envisaged that significant complexities in terms of network size and training set composition will be encountered if attempts were made to extend the system to domain independent language parsing.

The vast amount of research carried out with MLP-based parsing architectures have revealed their limitations. For example, the recursive representations developed within an SRN begin to degrade after about three levels of embedding. This indicates that the memory is bounded with an upper limit, after which the network will lose or 'forget' previous inputs in the sequence. However, the development of the SRN has resulted in a powerful architecture that can discover interesting representations that are formed from sequential input over time and that are hierarchical in nature. Also, the results illustrated that CNs with distributed representations are able to implicitly capture the temporal and compositional nature of language.

It is clear that modular connectionist architectures outperform singular connectionist architectures for language parsing. However, at present the combination of architectures used (i.e., SRN with RAAM) have proven limited even with simple context-free grammars, thus further architectural considerations need to be made when developing a modular connectionist architecture for large-scale natural language domains.

2.4.5. Discussion

It is now being realised that CNs and their associated distributed representations of information provide an alternative to symbolic strategies. Localist and distributed CNs were considered. Distributed representations offer novelty and can encode multitudes of different information across the same nodes; and allow transformations and structure manipulation operations to be performed upon them. Localist representations are suited to interpreting the distributed features formed within hidden layers of connection links and nodes; and possess many similarities with classical symbolic theories.

In contrast with symbolic representations, the distributed representations formed by Back-propagation learning with MLPs are inherently non-concatenative and extensions to the MLP architecture and Back-propagation learning allow such representations to encode temporal structure and lexical category structure if required. As a consequence of the learning process, CNs inherently distributed knowledge representation and parallel processing behaviour has enabled linguistic information to be represented as regions of state space rather than discrete context-free symbols. It is the connection strengths on the links between processing units that are interpreted as rules embedded within the dynamics of the system and which are represented in the context of all other stored information. Such representations allow the movement from certain regions within weight space to other regions while constraining other transitions, and thus the behaviour of language may be rule governed but not in the symbolic sense it is currently conceived to be [152]. Connectionist distributed representations therefore provide novel robust representations that are very different from symbolic representations, and that are also psychologically plausible in terms of behavioural characteristics.

Although the development of connectionist variable binding techniques and representation of complex structure, have provided valuable insight into how symbolic computation and storage can be performed with localist and distributed architectures much research still needs to done to improve memory capacity and generalisation capabilities. This must be achieved if connectionist techniques are to be computationally adequate for representing large-scale natural language domains. At present, well-established symbolic techniques still provide an easily manipulative and interpretative method of dynamic variable binding and storage for parsing purposes. Although the localist methods of parsing are easily interpretable, the rules of the language being modelled are hand-crafted, therefore amendments to the system is cumbersome and the more complicated the domain being modelled the more opaque the localist system to model it.

Distributed parsing models have achieved much success since the emergence of recurrent networks, such as the SRN, that are able to capture the temporal aspects of language. However, parsing models that attempt to perform the whole task within a single network are limited to using simple toy grammars. If the grammar used is large and complex then the size of the network required and the resulting training sets will become computationally unfeasible and intractable. A more successful approach has been to break the parsing problem down into separate modules and dedicate a suitable CN to each module. This has resulted in modular distributed architectures that can achieve impressive levels of generalisation. However, even modular distributed parsers have been based on toy grammars with reduced complexity due to the memory limitations of the RAAM architectures used to store phrase structure information.

Rather than a pure localist or distributed approach, recent advances in connectionist research [40, 161, 162, 163] has also seen the integration of symbolic techniques with localist *and* distributed representations to form hybrid systems that combine the advantages of all three representations during parsing. Wermter's Symbolic/Connectionist Approach for Natural language (SCAN) model [40] integrates all three representations to perform structural and semantic phrase analysis within a modular hybrid architecture. For example, in Wermter's Integrated Disambiguation model (ID model) he uses a localist network to perform structural phrase analysis and a distributed network to perform contextual phrase analysis (i.e., to determine the semantic plausibility of the syntactic structure). Both the localist and distributed networks have access to a lexicon that contains syntactic categories and semantic information about words. The localist network integrates multiple sources of constraints based upon the 'relaxation' of activation within the network. Such relaxation networks contain units that are connected via inhibitory and excitory weights that represent the influence one unit has on another within the network [5]. After the network has been initialised, activation spreads between each unit within the network according to some relaxation algorithm. To overcome the inherent problems with localist networks, Wermter's network shares units between different structural interpretations. In this model, Wermter concentrates on complex noun phrases. There is one unit per noun within a noun phrase, and a locality unit and semantic unit for each structural attachment. The locality unit represents the strength of the locality constraint (see Frazier [114, 115, 116]). The semantic unit represents the plausibility of the semantic relationship of a particular noun phrase attachment. The connections between the noun units and semantic units within the relaxation network reflect the different attachments within a noun phrase. The localist and distributed networks are integrated as the initial value of a semantic unit within the localist network is provided by the distributed network. The distributed network is an SRN that has learnt semantic relationships between nouns within a variety of contexts. Wermter also presents an Integrated Context model (IC model) that uses symbolic heuristics and SRNs to perform hybrid phrase analysis. However, although Wermter's work presents an interesting area for further research, it is difficult to assess how such architectures would process unrestricted domains of natural language. Also, the inherent limitations of localist representations are embedded within the system making it difficult to extend and maintain.

At present, there is no principled method of combining CNs to parse natural language, and previous approaches have been limited by the grammar and selected representations used.

2.5. Statistical Corpus-Based Models of Language Parsing

2.5.1. Introduction

The parsing systems discussed have traditionally based their grammatical framework on context-free grammars which are compliant with Chomsky's theory of linguistic competence rather than that of human linguistic performance i.e., how people use those rules in everyday conversation and communication. The fundamental problem with grammar rules are apparent when large-scale language is attempted. The level of precision provided by grammar rules constitutes a major disadvantage to the use of grammar rules as mentioned previously. Corpus linguistics is method of combining both linguistic theory and the observation of naturally occurring language by native speakers, commonly in the form of text.

A significant argument over using models of linguistic performance rather than Chomsky's model of competence is that made by Sampson [141], '*An automatic language-processing system which works adequately for competence but fails on performance is a futile system, because all there is to be processed is performance*'.

Corpus-based language models are data-orientated and fundamentally simple systems. The corpus commonly consists of text extracts from newspapers, magazines, novels, scientific papers, and text books in order to gain a wide coverage of the language and also how the language is used in varying contexts. Linguistic models can therefore be formed via using some training methodology to automatically extract statistical regularities relative to the task at hand e.g., word usage, tagging words with syntactic information, and syntactic phrase composition etc. Once trained, the system will then be able to process novel sentences via searching for statistical regularities that match those the system learnt from its training data. This presents a powerful approach that is able to process *unconstrained* large-scale natural language via automatically extracting statistical regularities from sentence examples, and overcome those obstacles faced by traditional competence-based parsers.

Large corpora are beginning to serve as an important tool for researchers in natural language processing, speech recognition, and integrated spoken language systems, as well as in theoretical linguistics. Annotated corpora are essential for the automatic construction of statistical models for the grammar of the written and colloquial spoken language and the comparison of the adequacy of parsing models.

The following subsections will therefore consider current corpora being used, and the statistical-based parsers built on this basis.

2.5.2. Available Large Corpora of English Texts

The available corpora that exist are either text-based or speech-based. Text-based corpora will only be considered for the work presented in this thesis, as the parsing systems developed assumes that adequate speech recognition has been performed upon the linguistic input before it enters the language processing system. Before discussing the available corpora it is important to distinguish between the different types of text corpora and consider the size of the corpora.

Corpora may be classified according to how much processing the corpus creators have performed upon the text. Leech [142] defined three formats of corpora : *raw corpora*, *tagged corpora* and *pre-parsed corpora*. Raw corpora consists of the 'pure' text in that no processing has been performed to annotate the text, and is the most popular form of corpora. Any body of text that has not undergone any type of analyses is considered raw. Tagged corpora contains the raw text and the syntactic categories for each word within the text. The development of automatic syntactic tagging systems such as CLAWS [143] has lead to much interest in tagged corpora, however tagged corpora is comparatively rare. In addition to the syntactic categories for each word in the corpus, parsed corpora contain further syntactic analysis such as parse trees in the form of labelled bracketing for each sentence in the corpora. Pre-parsed corpora are commonly produced by some automatic parsing system with manual post-editing by human linguistic experts. Pre-parsed corpora are rare which reflects the difficulty and expense of pre-parsing large bodies of text.

At present, the text corpora that are currently available or under construction for academic research are the Brown corpus, Penn Treebank, LOB corpus and the LPC corpus. The Brown corpus [144] contains approximately a million syntactically tagged words, and consists of American-English texts. The Penn Treebank, devised by Marcus et al [145], consists of over 4.5 million words of American English. The Penn Treebank at present, has been annotated for syntactic tagging, and over half of it has been annotated for skeletal syntactic structure. Work is still being undertaken on The Penn Treebank, and the resulting annotated treebank will prove extremely useful for future research projects. The Lancaster-Oslo/Bergen (LOB) corpus [146] consists of 500 English text samples from the 1960's. Each sample contains approximately 2,000 words and the complete corpus about one million words. Each word is tagged with its syntactic category. The LOB text samples provide a wide representation of varying text types to allow research on a broad range of aspects of the language. The syntactic tag set used in the LOB corpus consists of 132 tags, and each tag represents either classes of grammatically similar words, or for closed class words that have a special function.

The most accessible corpus which has been successfully annotated for syntactic structure, and the most suitable for forming the underlying basis for a system attempting to learn syntactic structure by example, is the Lancaster Parsed Corpus (LPC) [147]. The LPC is a corpus of English sentences excerpted from printed publications of the year 1961, and is a subset of the LOB corpus. Each sentence in the LPC has undergone syntactic analysis in the form of labelled bracketing. The LPC consists of 13.29% of the word-tagged LOB corpus. However, the sentences included in the LPC are considerably shorter, on average, than the average sentence length for the LOB corpus as a whole. The average for the LPC is 11.39 words per sentence, as compared with an average of 19 words per sentence for the entire LOB. The LPC therefore does not contain each LOB text sample in its entirety. However, the LPC is a valuable source and will be discussed further in chapter four when a modular connectionist parser for large-scale natural language parsing is proposed.

2.5.3. Hidden Markov Models

Hidden Markov models (HMM) are well known in speech recognition research and their formal properties and use in speech modelling is widely documented [77,78].

A HMM is a collection of states connected by transitions. Each transition carries two sets of probabilities: a transition probability, which provides the probability for taking this transition, and an output probability density function (pdf), which defines the conditional probability of emitting each output symbol from a finite alphabet, given that the transition is taken. There are numerous types of HMMs, the most common being discrete density HMMs which consists of a set of states, a set of transitions between states and the associated probabilities, and the output probability matrix, i.e., the probability of emitting some output symbol when taking a transition from state x to state y. A forward-backward algorithm is used to estimate the probability of making a transition between states, and to estimate the probability of emitting the output symbol. For each iteration of the algorithm, the estimates from the previous iteration are used to count how frequently each symbol is observed for each transition, and how frequently each transition is take from each state. It is these counts which are then normalised in to new parameters for subsequent processing.

After the huge success of applying HMMs to the speech recognition domain, researchers have inevitably incorporated HMMs in other language modelling tasks such as syntactic tagging and parsing. Early HMM statistical models attempted to syntactically tag large text corpora. To increase the usefulness of the LOB corpus, the Lancaster research team developed the Constituent Likelihood Automatic Wordtagging System (CLAWS) [143,148], which is a system for automatically tagging each word with the appropriate syntactic tags. CLAWS applies a probabilistic Markovian model to the grammatical analysis of corpora. The CLAWS input is a sequence of words; each word is assigned a set of possible syntactic tags, with the associated relative weights; the best sequence of tags is then selected using a Markov model of tag cooccurrences. CLAWS proved to be a very simple and successful statistical system for syntactic tagging, successfully tagging 96-97% of all words in the LOB corpus. The success of CLAWS lead to an attempt to extend the technique to perform automatic syntactic parsing of the LOB corpus through incorporating labelled bracketing to group syntactic tags into phrases.

Garside and Leech [142] developed a probabilistic parser that parsed the tagged output from the CLAWS system. Parsing involved the insertion of labelled brackets denoting constituent structure, and this in turn required a set of phrase- and clause-labels. The original intention of the parser was to automatically parse the entire tagged LOB corpus. However, the probabilistic parser developed proved inadequate to perform the task. A limited sample of 145 text extracts was chosen from the LOB corpus, which consists of 500 text extracts[9]. The parser was unable to achieve

[9] By extract I mean a collection of sentences of a particular type i.e., novel, editorial, scientific publication etc.

a parse of most sentences over 20-25 words in length and these unparsed sentences were therefore omitted from the resulting set of parse trees (represented with labelled bracketing). The remaining parsed sentences are based on a 'winning parse' where the parser assigns the highest probability to a particular parse. Each successful parse was then subject to manual post-processing to correct errors. It is these parsed sentences or 'treebank' which formed the Lancaster Parsed Corpus. A total of 11,827 sentences were eventually produced by the system with manual post processing. Therefore the LPC can be regarded as broadly representative of the syntax of written English across a great variety of styles and text types. In coverage of grammatical rules and structures, the LPC is more adequate than any other treebank publicly available to date.

An influential probabilistic parser is APRIL, developed by Sampson et al [94], which uses Hidden Markov models and simulated annealing. The objective of APRIL was to parse sentences from the LOB corpus after 'training' the system on a database of manually parsed English text, also extracted from the LOB corpus. In the APRIL system, there is no such notion as ungrammatical, instead APRIL seeks to assign a 'goodness' score to each sentence relating to the degree of grammaticalness. Simulated annealing is used to reduce the search space and settle on a probable parse. Beginning with an initial parse tree for the sentence, non-terminals are constantly being added, moved or deleted from the tree according to their effect on the tree-value : the decision to add or drop the non-terminal symbol is made probabilistically. As processing progresses and the 'temperature drops' certain structures are preferred over others and the surrounding developments are constrained by the preferred structures. It is this technique that enables the system to converge on a global optimum. The method in which Sampson compares APRIL's resulting parse with the target parse, is that for each input word he compares the chains of non-terminal symbols between the terminal nodes and the root (S) node in order to compute the number of non-terminals which match each other and occur in the same order. An average is then made over all input words. The consequence of averaging over words means that high-level non-terminals (those nearer the root node) dominate many words and thus contribute more to the overall cost than lower-level non-terminal symbols. The resulting cost is a percentage of how close the actual parse is to the target parse. Sampson reported that the average cost for 50 test sentences was 75.3% accuracy.

Atwell and Pocock [149] developed an HMM parser that used pre-parsed sentences from the Lancaster Treebank as training data. The smallest training set consisted of 30 pre-parsed sentences, and the largest 292 pre-parsed sentences. Once trained, attempts were made to parse the training data. Atwall reported disappointing results. It could only parse 19 of the 30 pre-parsed training sentences, and increasing the training set reduced the accuracy further. This was apparently due to further ambiguity being added to the training set. However, Atwall did report better levels of accuracy for sentences not in the training data (i.e., generalisation), and stated that this accuracy increased as the number of training samples increased. Many problems were encountered in terms of testing times, and little information was given as to how similar the test sentences were to the training data. The model itself was computationally inadequate and insufficient to process realistic subsets of natural language.

Attempts have been made to learn the correct grammar for a given corpus of data, through applying HMM maximum-likelihood re-estimation techniques (such as the 'inside-outside' algorithm rather than the traditional Vertibi techniques) to probabilistic context-free grammars [96,145,150,151,164,165]. This type of data-orientated approach form initially weak grammatical constraints from sentence examples, and add more constraint by using iterative re-estimations of the rule probabilities to converge on the appropriate subset of context-free rules.

More recently, Charniak [164] uses these HMM-based re-estimation techniques to assign probabilities to possible top-down parses for sentences extracted from the Penn Wall Street Journal Treebank [145]. The parser is trained using 1,000,000 words of the Penn Treebank and a context-free grammar is extracted from the model using the technique described by Charniak [165]. The parser was tested on 50,000 words. A top-down parse is performed using a chart-parsing technique to help find the most probable parse which is measured according to a simple probabilistic context-free grammar distribution. This rules out improbable parses and forms a chart containing constituents and associated information on how they help form parses. The parser then extracts the parse with the highest probability. Charniak reports a 1-2% increase over similar methods.

2.5.4. Discussion

The availability of large text corpora has provided a valuable training and test source for system designers who are attempting to build natural language parsing systems that learn statistical regularities about word and phrase usage in naturally occurring language, rather than constructing grammar rules that are open to constant amendment.

Although HMM-based models of language parsing have definitely provided a more robust framework there are problems parsing realistic language domains. The re-estimation techniques employed in some HMM language models have to consider the entire search space even for a small grammar or subset of a corpus, and so the likelihood of convergence on a useful subset of language is low. HMMs have similar problems to those encountered with localist connectionist models in that HMM-based parsing systems are hand-crafted and therefore amendments to the system are cumbersome and the more complicated the domain being modelled the more opaque the HMM system becomes. A significant limitations of applying HMM models or n-gram statistics to syntactic parsing is that such models cannot represent hierarchical structures found in natural language [155]. The success of an HMM-based parser is therefore dependant upon the system designer, as HMM models have no inherent notion of feature extraction or dynamic structural representation.

Recent advances in symbolic approaches to corpus-based parsing are also challenging current HMM-based methods. Brill's symbolic parser [173, 174, 175] defines a novel transformation-based error-driven learning algorithm that is able to automatically extract a set of transformation rules from a pre-parsed corpus of text to accurately parse text into binary-branching syntactic trees with unlabelled non-terminal symbols. The algorithm works by beginning in a very naive state of knowledge about phrase structure. By repeatedly comparing the results of bracketing in the current state to proper bracketing provided in the training corpus, the system learns a set of simple structural transformations that can be applied to reduce the error. Although the resulting parsing model is purely symbolic, it parses in linear time and since some tokens in a sentence are not even considered in parsing, the method is considerably more robust than conventional rule-based parsers when processing unfamiliar input. The results reported by Brill have been comparable to HMM-based parsers that employ re-estimation techniques such as [150].

2.6. Summary

The majority of traditional parsing models use linguistic grammars and processing heuristics that lack flexibility and this has limited their success. Systems that rely completely on the use of grammar rules are always open to amendment and by nature are unable to encode the necessary level of abstraction required to capture realistic or even acceptable levels of natural language. It is envisaged that by integrating the principles employed in current linguistic grammars with a data-orientated framework will allow the development of a syntactic parsing system that can learn language from sentence structure examples without a prior knowledge of the language, and provide a system that is able to automatically capture syntactic generalisations. Also, a deterministic approach to syntactic parsing will help reduce the search space by disregarding possible state configurations as more input information is processed as look aheads, and remove the need for the back-tracking mechanisms employed by top-down parsers.

At this point in the development of connectionist techniques, the ability to perform large-scale natural language parsing has not been demonstrated. The connectionist architectures most suitable for the particular functions of a parser has also not been fully established. The issues of dynamically encoding complex recursive structure and variable binding within connectionist networks remains open, and parsing systems based purely on connectionist techniques will always inherit the severe limitations associated with connectionist variable binding and structure representation. However, hybrid parsing systems that integrate the advantages of both symbolic and connectionist processing show their potential in modelling large-scale natural language parsing. A pure connectionist or symbolic approach will therefore not be addressed, as the most feasible strategy is to combine the strengths of both approaches, to counteract the weakness of each. By allowing each approach to work in parallel it is envisaged that an effective solution can be achieved.

Chapter 3

Preliminary Investigations with Context-free Languages

3.1. *Introduction*

When applying a single feed-forward MLP architecture with Back-propagation learning to language parsing [55,130, 131] temporal input windows are commonly used to overcome the problem of fixed input constraints. This enables such networks to perform a type of n-gram statistics of adjacent input symbols. By moving the window across the whole input sequence, sequences of an unrestricted length could be processed using a feed-forward MLP architecture. However, the temporal context is limited to the size of the input window, thus if the contents of the input window cannot be disambiguated then the parser will fail on the current input. Feed-forward MLPs have therefore been combined with recurrent MLP architectures to form modular connectionist parsers [62, 126, 127, 139] that sequentially process arbitrary length input sequences. The modular approach divides the parsing task into a number of simple modules.

However, there is no clear consensus as to what function feed-forward MLPs can and cannot play in the parsing process. Before constructing a modular connectionist parser, the following questions needs to be answered : Can a single feed-forward MLP simultaneously perform all the functions of a parser? At what point do we need to introduce further connectionist modules? This chapter attempts to address these questions using simple context-free languages as the grammatical framework for the experimental parsers. Section 3.2 uses a feed-forward MLP network to act as an arithmetic expression recogniser within a hybrid shift-reduce parser. The language of arithmetic expressions is chosen as it presents a simple well defined language domain that is strictly context-free and provides a simple recognition task for the MLP network. Section 3.3 then extends this technique to examine whether a single feed-forward MLP network can simultaneously perform all the functions of a parser for a simple context-free natural language grammar. Finally, Section 3.4 discusses the performance and adequacy of the MLP network in its given roles.

3.2. *A Hybrid Shift-Reduce Parser for Arithmetic Expressions*

This section presents a shift-reduce parser for arithmetic expressions. The shift-reduce parsing strategy used is similar to that proposed by Shieber [105,106]. The task of the MLP network is to determine what input combinations can be reduced and which input symbols must be stored on the parse stack. The manipulation of symbols and flow of information is controlled by 'supervisory' C++ code. The following sub-sections discuss the hybrid parser.

3.2.1. *Grammatical Framework*

Although there is no formal context-free grammar associated with the hybrid shift-reduce parser, a context-free grammar will be defined to guide the construction of the training data for the MLP network. The majority of formal and artificial languages can be described with context-free grammars. The context-free grammar defined for describing the arithmetic expressions can be seen in figure 3.1. An arithmetic expression (*E*) consists of one or more arithmetic terms (*T*). Each term contains a binary operator and two input symbols (operands). The terminal symbol, *e*, can be considered as the *empty set* in that the rule in which it appears cannot be matched against the input and terminates the recursion. The order in which arithmetic terms are reduced by the parser and thus the order in which arithmetic operations are performed, is determined by the precedence relation of the binary operator. Although the grammar in figure 3.1 allows the use of parentheses, it does not specifically encode precedence relations. The search procedure within a purely symbolic parser would have to implement some heuristic that would determine the precedence of arithmetic terms within the input expression.

$$
\begin{aligned}
&\langle E\rangle \longrightarrow \langle T\rangle\ \langle E^*\rangle\\
&\langle E^*\rangle \longrightarrow +\langle T\rangle\ \langle E^*\rangle \mid -\langle T\rangle\ \langle E^*\rangle \mid e\\
&\langle T\rangle \longrightarrow \langle P\rangle\ \langle T^*\rangle\\
&\langle T^*\rangle \longrightarrow *\langle P\rangle\ \langle T^*\rangle \mid /\ \langle P\rangle\langle T^*\rangle \mid e\\
&\langle P\rangle \longrightarrow A\,..\,Z \mid a\,..\,z \mid (\langle E\rangle)
\end{aligned}
$$

Where :

<E> - Arithmetic expression
<E*> - 0 or more arithmetic expressions using at least one + or –
<T> - Arithmetic term
<T*> - 0 or more arithmetic expressions using at least one * or /
<P> - Primary or expression in parenthesis

e – empty set

Figure 3.1 : A context-free grammar to describe simple arithmetic expressions.

Also, precedence relations for such an arithmetic expression parser could be considered a semantic issue and would thus be enforced by a semantic module. The training data constructed for the MLP network will therefore only be loosely based on the context-free grammar shown in figure 3.1, as the precedence relations amongst the binary operators will be explicitly encoded in the training data. The MLP network will consequently force 'reductions' on higher precedence arithmetic terms before lower precedence terms.

3.2.2. Language Constraints And Coverage

The coverage and complexity of the expressions to be parsed by the system will be limited by the following constraints :

- The operators allowed are limited to :

+	Addition
-	Subtraction

*	Multiplication
/	Division
%	Modulus

Operators with the same precedence will be represented identically. The operators + and - have the lowest and identical precedence levels, whereas / and % have the highest and identical precedence levels.

- The precedence of expressions or sub-expressions can be altered through the use of parentheses :

(	Open parenthesis
)	Closed parenthesis

- Variables are given identical representations. They can be strings of any alphanumeric character, where the first character must be a letter. For the purpose of these investigations the parser will not take into account the value of each variable and consequently the use of symbol tables.

- A reduction operation instantiates a unique non-terminal symbol. A non-terminal symbol is assigned the same bit representation as a variable.

- Each arithmetic expression must be terminated with a semi-colon.

- The operators are binary and therefore operate on two variables at a time. A maximum of four symbols will be processed by the parser at any one time so that the two variables (operands) and the operator can be reduced when presented with the fourth symbol for determining operator precedence. For example, in the expression *a* + *b* /, the fourth symbol indicates that the addition must not be performed as a division operator is present and must be processed first. This eliminates the need to back-track and 'undo' previous reductions.

- The direction of the parse will be from Left-to-Right starting with the first four input symbols.

It is envisaged that by using these constraints the architecture developed will be capable of parsing arithmetic expressions of arbitrary length and complexity.

3.2.3. The Parsing Architecture And Algorithm

A parsing algorithm is required to determine the derivation for a given input expression. It specifies how the input symbols are to be accessed, how the input is to be processed, and also how the symbolic and connectionist structures are integrated in order to produce the resulting symbolic parse tree derivations. The basic parsing algorithms (top-down or bottom-up) were discussed in the previous chapter. It was explained that the bottom-up method appears to be more efficient in that no back-tracking is required and hence no redundant structures are formed. The parsing method used is based on the shift-reduce parsing strategy defined by Shieber [105,106] as it provides an efficient method of grouping and reducing input constituents according to some grammatical constraint. This allows the parser to shift and store input symbols until an appropriate grouping, recognised by some finite state automaton, is recognised as matching a specific rule of the grammar. The matching input is then replaced or *reduced* to one symbol denoting the left-hand symbol of the appropriate grammar rule. However, if the current input to the parser is not a reducible term then the parser *shifts* along the input expression.

For some grammars there are problems associated with this parsing strategy as there may be more than one successful path through the grammar[10]. However, it is suitable for arithmetic expressions as alternative parses are equally valid and the use of the MLP network will force the parser to use one parse.

Parsing Architecture

The architecture can be seen in figure 3.2 and consists of the following components:

- A temporal input window that can store a maximum of four terminal/non-terminal symbols at any one time (i.e., a quadgram).

- A three-layered feed-forward MLP network paradigm trained using the Back-propagation learning algorithm (see Appendix B). The training data consists of numerous subsets of positive examples (e.g., $a + a +$) and negative examples (e.g., $a{+}{+}a$) to find the optimum number of input patterns required for the connectionist network to learn the grammar. The total number of input patterns is not fully used during training so that the generalisation capability of the network can be analysed. The training data contains input pairs consisting of a bit representation of the symbols in the temporal input window and the desired output response from the MLP network. Each input symbol is assigned a 3-bit code resulting in an input layer of 12 input units.

The units contained within the hidden layer automatically encode feature information about the current input as learning takes place. A single output unit is used to signify either a reduction, or a movement across the input expression.

[10] However, there are methods of resolving ambiguous grammar entries. The most common method is to use a *chart* that will store all of the hypotheses that have been applied to the input. Implementing such a mechanism avoids repetition of work that has already been performed by inhibiting any attempt to reapply the same rule in the same place.

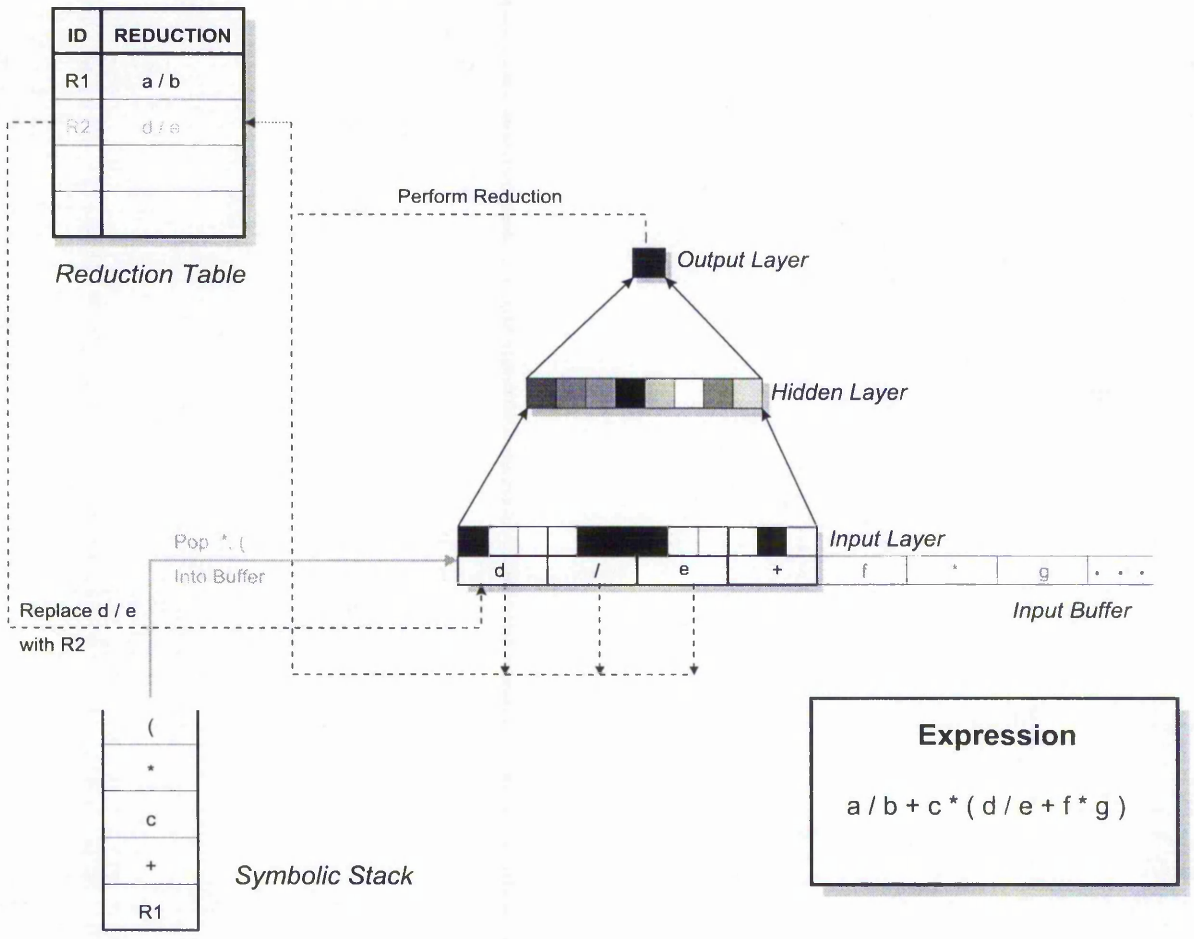


Figure 3.2 : The hybrid shift-reduce parser for arithmetic expressions.

The MLP network in figure 3.2 has been depicted as a block diagram with inter-connections omitted for clarity. The parser is shown parsing the expression *a/b+c*(d/e+f*g)*.

- A symbolic reduction table to store and uniquely label reductions.

- A symbolic Last-In-First-Out (LIFO) stack structure to store unprocessed input symbols and intermediate parse states[11], and is known as the primary stack.

The connectionist network acts as a recogniser, accepting or rejecting fragments of the expression at each time step. It can be seen that only the constituents contained within the temporal input window are analysed by the connectionist network. To process further tokens, the temporal input window shifts right or left along the input buffer until it has

[11] A secondary stack is required if the temporal input window is forced to shift left (against the direction of the parse) to process the remaining input symbols stored on the primary symbolic stack.

reached its capacity, or there are no more constituents remaining. The symbolic structures provide important temporary storage and unique labelling of reductions. The process of training the connectionist network is discussed in detail in Section 3.3.

Parsing Algorithm

The parsing algorithm governs how the components within the hybrid architecture will communicate with each other; and hence how the parse structures will be formed. It also determines the efficiency of the parser. The shift-reduce algorithm developed is implemented in the C++ programming language for efficiency and portability. It is executed on either a SUN SPARC 20 mini-computer or a standard Intel 80486-66MHZ based personal computer. The SUN SPARC is the preferred system due to the availability of a *flat* memory model which allows full use of its memory and disk resources.

```
Main_Loop
{
        output_threshold= 0.5; //set a threshold for the MLP network's output activation.
        Initialise current_symbol ;
        while ((current_symbol != terminator) && (input_buffer != empty))
        {
                Read a symbol from the input_buffer and store in current_symbol;
                Get_Constituents();         //fill up the temporal input window
                Process_Constituents();     //process the constituents in the temporal input window
        }

        //now interpret the final state of the parse
        if      (stack==empty     &&
                 temporal input window contains a non-terminal symbol from the reduction table &&
                a terminator has been encountered)
        then    a  valid expression has been encountered
        else    an invalid expression has been encountered

        //the resulting parse structure for the input expression is contained in the reduction table
}
```

Figure 3.3 : The basic outline of the shift-reduce parsing algorithm.

A summary of the algorithm is provided as pseudo/C code in figure 3.3. It can be seen from this that symbols are repeatedly extracted from the input buffer, placed into the temporal input window and processed until either the terminating symbol has been encountered or all input symbols have been processed. Finally, the contents of the symbolic structures are then interpreted.

Figure 3.4 lists the two main procedures of the algorithm which describe how constituents enter the temporal input window and how they are processed. The procedure *Get_Constituents* puts input patterns, representing the input symbols contained in the input buffer, into the temporal input window. These are the three bit representations of each input symbol e.g., *011* represents the division operator. This forms one *view* of the input expression. Constituents are read into the temporal input window until the terminating symbol has been encountered, the input buffer is empty or until the temporal input window has reached its maximum capacity of 12 bits i.e., 4 input symbols.

```
Get_Constituents()
{
        Place current_symbol into the next free position in the temporal_input_window.
        while   ((current_symbol != terminator) &&
                (temporal_input_window!= full) &&
                 (input_buffer != empty))
        {
                Read a symbol from the input_buffer and store it into current_symbol.
                Place current_symbol into the next free position in the temporal_input_window.
        }
}

Process_Constituents()
{
        Present the contents of the temporal_input_window to the connectionist network.

        if (MLP_Output_Value > output_threshold)
        {//The three left-most symbols in the temporal_input_window are reducible
                Store the 3 leftmost symbols from the temporal_input_window into the reduction table.
                Make an entry in the reduction table, assigning the reduction with a unique identifier.
                Assign the identifier the relevant bit pattern.

                Place the unique identifier into the leftmost position of the temporal_input_window.

                //process the stack
                while ((stack != empty) && (temporal_input_window != full))
                {
                  Shift the symbols of the temporal_ input_ window right one position
                  Pop a symbol from the top of the stack into the left-most position of the temporal_input_window
                }
                Process_Constituents(); //recursively call itself to process a modified temporal_input_window
        }
        else
        {//shift the temporal_input_window across the expression right one position and store unprocessed
         // input symbol
                Push the left-most symbol of the temporal_input_window onto the stack
                Shift the remaining input symbols left one position
        }
}
```

Figure 3.4 : Procedures to obtain, shift and reduce input constituents.

The procedure *Process_Constituents* describes what actions are performed given the output activation response of the MLP network. The range of the activation response is limited to a continuous value between 0.0 and 1.0. This is due to the use of the sigmoid activation function. If the connectionist network response exceeds a predetermined threshold, *output_threshold*, then the temporal input window contains a reducible expression or *term*. The *output_threshold* value is fixed at 0.5.

A valid or reducible term is indicated when the input consists of :

i. a binary operator with two operands or a reduced expression enclosed in parentheses. This will always form the first three symbols of the reduction.

and

ii. a fourth symbol that is either a closed parenthesis, a terminating symbol, or an operator with precedence less than or equal to the operator (if present) contained in i).

When a reducible term has been recognised, the three left-most symbols from the temporal input window are removed; and placed as an entry within the reduction table. The reduction table is a symbolic linked list structure of records. Each record consists of the three input constituents belonging to the reducible term. A unique symbol is then assigned to the entry. If a context-free grammar was used then the unique symbol represents the appropriate non-terminal symbol on the left-hand side of a rule. The non-terminal symbol denoting the reduction is assigned the three bit vector of a variable/operand. This is then placed in the leftmost position of the temporal input window, in effect replacing the reducible term. This results in a three to one reduction of input symbols. After a reduction has been processed, the temporal input window is able to accommodate further input constituents. However, unprocessed symbols may reside on the LIFO symbolic stack, and therefore an attempt must be made to process them. Symbols are *popped* off the LIFO stack and into the left-most position of the temporal input window; thus shifting the present contents of the temporal input window right. This process is repeated until the stack is empty or the temporal input window is full. If the stack is empty but the temporal input window is able to accommodate further input constituents, then further symbols are extracted from the input buffer. Finally, the temporal input window is again presented to the MLP network, and *Process_Constituents* processes the networks activation response by recursively calling itself.

If the response of the connectionist network activation is below the threshold limit then a reduction is not possible and further input constituents need to be processed. In this case, the left-most symbol of the temporal input window is *pushed* onto the stack; and the contents of the temporal input window are shifted by left one position. Further input constituents can now be extracted from the input buffer.

A successful parse would be reported if all of the following conditions are true :

i. A single symbol, denoting a the whole parse tree, is the only symbol remaining in the temporal input window.

ii. The terminating symbol has been encountered.

iii. The input buffer and stack are empty.

An unsuccessful parse would be reported under either of the following conditions :

i. The input buffer and stack are empty; and the input symbols remaining within the temporal input window cannot be reduced.

ii. The input buffer is empty and a terminating symbol has not been encountered.

Input expressions that possess two or more consecutive operators and/or operands, unmatching parentheses, or a missing operator will cause a failed parse.

It has been assumed so far, that the MLP network has learnt to accept valid terms and reject invalid terms of length four. The following section discusses in detail the MLP architecture, the input representation of symbols, training set construction and the training and test performances of the network.

3.2.4. The MLP Network for Arithmetic Expression Recognition

The task of the feed-forward MLP network is to simply classify arithmetic terms of length four as valid or invalid. The architecture of the MLP network and the learning algorithm used to train the network must first be established before any learning phase can take place.

The MLP network architecture consists of three layers of units and two layers of weights[12] and is referred to as a three-layered MLP network. The Back-propagation learning algorithm is selected to train the MLP network. The reason for this selection is that Back-propagation and its variants are universal approximators and can learn to compute anything computable [73]. It can learn any linear and non-linear function through exposure to examples of the function. Also, the execution speed of multi-layer feed-forward MLPs is among the fastest of all connectionist models in use today. Learning will take place in on-line mode[13], and the learning rate and momentum term will initially be set to 0.35 and 0.9 respectively.

The possible input symbols (e.g., *a*, +, -, /, %) must be converted into a representation suitable for a connectionist network to process. This may consist of binary or continuously valued input representations. The connectionist network must then be trained to perform the classification task required. Its performance can then be analysed and it can be retrained if required.

[12] The weights for the connectionist network are initialised to random values between –0.1 and 0.1 before a training session is undertaken.

[13] A single training epoch is when all training patterns have been presented to the network. Training performed in on-line mode forces the weights to be adapted per pattern presentation. Batch mode learning allows the weights to be adapted when a number of epochs have occurred and the accumulative pattern errors and deltas computed.

3.2.4.1. Input Representation

Although the hybrid architecture is expected to parse expressions of arbitrary length, only four input symbols are processed by the connectionist network at a time. This limits the size of the training set and consequently the size and complexity of the network architecture itself. Processing more than four input symbols at a time would be of no advantage when parsing simple arithmetic expressions as the operators used are binary in the sense that they consist of *operand-operator-operand*, so three symbols need to be considered for a valid operation. A fourth symbol is considered to inhibit reductions, in order to process operators with a higher precedence further on in the expression first. This symbol is a look-a-head symbol.

It can be seen from table 3.1 that there are eight unique input symbol representations, thus the network can be presented with a maximum of 8^4 expressions of length four, or 4096 unique expressions. The operators + and - have the same precedence and therefore identical input representations, as do / and %.

Symbol	Description	Input Representation
a	Variable/Operand	1 0 0
+ -	Addition & Minus Operators	0 1 0
*	Multiplication Operator	0 0 1
/ %	Division & Modulus Operators	0 1 1
(	Open Parenthesis	1 1 0
)	Closed Parenthesis	1 0 1
;	Terminator	1 1 1
^	Null	0 0 0

Table 3.1 : Input representation of allowed input symbols.

3.2.4.2. Training And Test Experiments

The connectionist network is expected to produce an output activation above 0.5 to indicate a reducible expression. Activations below this threshold will indicate an invalid expression is present on the input units. Therefore the training data will consist of two pattern classes, one to represent valid expressions and the other to represent invalid expressions. This requires the use of only one output unit. The threshold acts like a decision boundary, classing activations below it as 0 and those exceeding it as 1. However, the continuous-valued activation can also be viewed as a graded response from the network indicating the probability of an input pattern belonging to a particular class. Activations that are close to the threshold are known as *borderline cases*, and this implies that the network is unsure of which class the input pattern belongs to. This may imply that the connectionist network requires further training on such patterns.

If the MLP network is to be successful at recognising expressions, the training set must be complete enough to be representative of all pattern classes. The training set will contain *all* valid terms, and a percentage of the invalid expressions. It is imperative that all valid terms be learnt. *The fourth symbol provides the necessary precedence information and is based on the operator precedence relations defined in table 3.2.* This symbol must be an operator of lower than or equal precedence to the one in the term. For example, the expression a + a / would be termed invalid as the divide operator has a higher precedence than addition.

Operator	Precedence
(	1
)	2
+, -	3
*	4
/, %	5

Table 3.2 : Operator precedence relations.

A total of 17 expressions have been defined as valid and can be seen in table 3.3. Since there are only 17 valid terms, this leaves a possible 4079 invalid expressions. This produces a severe imbalance of patterns in the set of possible training patterns. Feed-forward MLP networks trained with Back-propagation cannot automatically adjust for unbalanced training sets. As such networks learn by minimising the mean error across the entire training set, the degree of representation of the pattern classes in the training set can have a profound influence on the network's performance. For example, if a particular pattern class is disproportionally highly represented, the connectionist network will attempt to optimise its performance for that pattern class, at the expense of the other pattern class.

This implies that the connectionist network will find it difficult to learn the valid terms as there are many more invalid terms than valid terms. As the network does learn the valid terms it will affect the representation of invalid terms and affect its ability to generalise[14] to the remaining invalid expressions. Ideally the training set should contain a subset of invalid terms that will allow the connectionist network to recognise all invalid expressions, whilst simultaneously allowing it to learn all valid terms.

One method of overcoming training set imbalance is pattern replication. Examples from the under-represented set are replicated so that they constitute a higher proportion of the entire training set. The remaining training examples would consist of a percentage of the over represented terms. The prominence of valid terms in the training set for arithmetic expressions would therefore be increased. Another method to overcome the problem of training set imbalance is to adapt the learning rate in a way that would force the connectionist network to become more sensitive to patterns that are disproportionally represented in the training set.

[14] Generalisation is a term commonly used to describe the ability of a connectionist network to correctly classify those input patterns that were not present in the training set by recognising features that are common to both or are general.

Valid Expression	Input Representation
a + a +	1 0 0 0 1 0 1 0 0 0 1 0
a + a)	1 0 0 0 1 0 1 0 0 1 0 1
a + a ;	1 0 0 0 1 0 1 0 0 1 1 1
a * a +	1 0 0 0 0 1 1 0 0 0 1 0
a * a *	1 0 0 0 0 1 1 0 0 0 0 1
a * a)	1 0 0 0 0 1 1 0 0 1 0 1
a * a ;	1 0 0 0 0 1 1 0 0 1 1 1
a / a +	1 0 0 0 1 1 1 0 0 0 1 0
a / a *	1 0 0 0 1 1 1 0 0 0 0 1
a / a /	1 0 0 0 1 1 1 0 0 0 1 1
a / a)	1 0 0 0 1 1 1 0 0 1 0 1
a / a ;	1 0 0 0 1 1 1 0 0 1 1 1
(a) +	1 1 0 1 0 0 1 0 1 0 1 0
(a) *	1 1 0 1 0 0 1 0 1 0 0 1
(a) /	1 1 0 1 0 0 1 0 1 0 1 1
(a))	1 1 0 1 0 0 1 0 1 1 0 1
(a) ;	1 1 0 1 0 0 1 0 1 1 1 1

Table 3.3 : Valid terms with their corresponding input representation.

An advantage of this technique is that it would not require an increase in training set size. A method for adapting the learning rate is proposed in detail later in this chapter.

When training the MLP network, a general error measure is computed to provide a value that indicates how well the network is learning the training examples. This value is called the *root mean squared error* (RMS). The RMS is computed by finding the square root of the sum of squared differences between the desired target outputs and the actual output values attained across all training patterns and taking the square root. If d_j^p is the desired value for output unit j in response to input pattern p, and o_j^p is the value obtained by the network for output unit j, the root mean squared error for p input patterns, and q output units is calculated as :

(E3.1)

$$RMS = \sqrt{\frac{1}{p}\sum_{i=0}^{p-1}\frac{1}{q}\sum_{j=0}^{q-1}(d_j^p - o_j^p)^2}$$

Experiments undertaken without any measures to overcome the training set imbalance resulted in a reasonably low RMS error, without all training patterns being learnt. For example, several training attempts were made with a training set consisting of all 17 valid terms, and 50% of the 4,079 invalid terms, so valid terms constituted only 0.8% of the training set. Typical training results attained with six hidden units were an RMS of 0.0696 after 500 epochs but the MLP network consistently failed to learn all valid expressions. Although a generalisation rate of 100% was attained, there are only invalid expressions in the test data. Results using the two methods to overcome the training set imbalance outlined earlier are presented in the next two sections.

3.2.4.2.1. Pattern Replication

The optimum level of pattern replication is dependant upon the problem domain, and has to be empirically established. In this problem domain there are only 17 possible valid terms, in contrast to 4,079 invalid expressions so only the valid term pattern class is replicated. All training set configurations initially contained 50% of the invalid terms (2,040 patterns) to ascertain whether the MLP network is able to generalise to the remaining 2,039 invalid terms. The invalid terms were randomly selected.

Experiments to Ascertain Optimum Replication

As the MLP network will be trained on all valid terms and 50% of invalid terms, the generalisation can only be to invalid terms. A number of training set configurations were used to obtain the optimum balance of valid and invalid terms. The number of valid terms in the training set is determined by the value of **R** e.g., if **R** is 30 then the number of valid terms in the training set is 30*17 = 510. Table 3.4 shows the number of valid and invalid terms contained within each training set used. A percentage indicating the balance of examples in the training set is also given.

Training Set	Total Patterns	R	Valid Terms	Invalid Terms	% Valid	% Invalid
1	4,114	122	2,074	2,040	50.4	49.6
2	3,060	60	1,020	2,040	33.3	66.7
3	2,550	30	510	2,040	20	80

Table 3.4 : Training sets defined to find the optimum level of replication, **R**, to enable the MLP network to learn all valid terms and 50% of invalid terms.

It can be seen from table 3.5 that segmenting the training set into an equal balance of valid and invalid terms, as described by training set one in table 3.4, does not produce acceptable generalisation results. Various numbers of hidden units were used, ranging from 6 to 20 units, in order to find the optimum level for feature extraction and compression. Although an average RMS value of 0.02 was attained, all of the networks were unable to learn all training patterns after 10,000 epochs. The experiments undertaken with 20 hidden units gave a slight increase in performance over the other configurations, achieving a generalisation rate of 47.25%. However, it is envisaged that increasing the number of hidden units to enable further learning would force the network to memorise training patterns rather than to learn the *general* features of each training pattern. It is therefore hypothesized that improved rates of generalisation will not be attained via increasing the number of hidden units.

Generalisation Results for MLP networks Trained with Training Set 1 (R=122)			
Hidden Units	**RMS for Training Data After 10,000 Epochs**	**Invalid Terms in Test Data**	**% Generalisation**
6	0.023435	2,039	31.42
7	0.019217	2,039	26.13
15	0.017774	2,039	47.16
20	0.009970	2,039	47.25

Table 3.5 : The training and test performance of a number of MLP network configurations for training set 1.

The replication level of valid terms, **R**, appears to be excessive as the MLP network is learning the valid expressions at the expense of the invalid terms. This training set is therefore not optimal for efficient learning and storage of information in order to perform the classification task. All network configurations converged within 637 epochs when trained with training set 2 (see table 3.4). The MLP network with 20 hidden units learnt all training patterns after 367 epochs, and the network with 6 hidden units converged after 406 epochs. It can be seen from figure 3.6 that a perfect generalisation rate of 100% was achieved by all networks. The minimum hidden units to learn the problem was 6. Similar results were achieved using training set three (see table 3.4) to train the MLP network configurations.

Generalisation Results for MLP networks Trained with Training Set 2 (R=60)			
Hidden Units	**Epochs taken to Learn Training Data**	**Invalid Terms in Test Data**	**% Generalisation**
6	406	2,039	100.00
7	410	2,039	100.00
15	636	2,039	100.00
20	367	2,039	100.00

Table 3.6 : The training and test performance of a number of MLP network configurations for training set 2, described in table 3.4.

Training set three consists of all valid terms which have been replicated so that they constitute a fifth of the entire training set (20%); and 50% of the possible invalid terms are included to constitute the remaining 80% of the training set. This provides a stronger bias towards invalid terms than training set two as well as being a smaller training set. The MLP network with 6 hidden units learnt all the training data after 200 epochs achieving an RMS error value of 0.000409. The MLP configuration with 6 hidden units was tested at various stages during the learning process. The analysis revealed that the MLP network incorrectly classified all input as invalid after 1 epoch and consequently produced a 100% generalisation rate when tested on the test data. However, the network gradually begins to learn the valid terms after a further 99 epochs, and has learnt all training patterns after 200 epochs. Training set three proved to contain the optimum level of replication. The MLP network with 6 hidden units was adequate to learn the simple recognition task.

Experiments to Maximise Generalisation

Experiments were undertaken to analyse the effect on the generalisation performance when the percentage of invalid terms is reduced. There were six training sets used. Each contained all 17 positive examples replicated to constitute a third of the training set. The rest of the set was composed of a percentage of the total population of invalid terms, ranging from 1% to 50%. Previously, just 50% of the invalid terms had been used for training experiments. All networks converged rapidly, achieving an RMS error of approximately 0.1 after just 100 epochs, and 0.001 after 10,000 epochs. However, the training set containing 50% of invalid terms converges further to an RMS error of 0.0001. The networks performed well, generalising to greater than 90%. When the MLP network is presented with 50% of invalid expressions, in the training set, a generalisation rate of 100% is attained. The generalisation for the other sets was less than this. However, the experiments proved that only a small proportion of invalid terms were required in the training set to produce acceptable generalisation rates. For example, after being exposed to 10% of a possible 4079 invalid expressions, the network managed to classify 96% of the remaining 3,672 expressions correctly. When the training no longer has an overall bias towards invalid cases, generalisation is expectedly reduced.

Conclusions

Replication of training patterns to improve the balance of the training set resulted in complete learning of training patterns, and 100% generalisation. However, this was at the expense of an increased training set and possibly consequent increased training time. Also, the success of the training in terms of the degree of generalisation depended upon the amount of replication. The effect of incomplete training data in more than one class or the effect of having more than two problem classes has not been investigated.

3.2.4.2.2. Pattern-error Sensitive Learning Rate

Although the Back-propagation MLP network can approximate any function to arbitrary accuracy there are problems concerned with learning the approximation. This limits the usefulness of the algorithm. The main problem is the long training process required. This can be the result of a non-optimal learning rate (or momentum term). The selection of the learning rate, η, is generally empirical, but a variety of rules for fixing the level have been developed. Previous research in this area has suggested setting the learning rate according to the training set size [58]. There have also been techniques that adaptively adjust η as learning takes place [59]. Both of these methods aim at speeding up convergence.

However, complete training failures with Back-propagation generally arise from the following two sources [73, 156, 157] :

- *Network paralysis.* As the network trains, the weights can become very large. The total input of a hidden or output unit can therefore reach very high magnitudes, and due to the sigmoid activation function the unit will have an activation very close to zero or very close to one. The weight adjustments which are proportional to $a_i^p\ (1-a_i^p)$ will be close to zero, and the training process can come to a virtual standstill.

- *Local minima.* The error surface of a complex network consists of hills and valleys. Because of gradient descent, the network can become trapped in a local minimum when there is a deeper minimum nearby. Probabilistic methods [73] can help to avoid this trap, but they tend to be slow. Another method of avoiding a local minimum is to increase the number of hidden units. Although this can work because the higher dimensionality of the error space results in the chance of getting trapped being smaller, it appears that there is some upper limit on the number of hidden units which, when exceeded, again results in the system being trapped in a local minimum. In addition, excessive numbers of hidden units results in poor generalisation performance as the network memorises the input patterns.

As explained earlier, the MLP network was unable to learn all valid terms without some form of pattern replication. This implies that the network continually reaches a local minimum due to a small subset of patterns not being learnt. A solution might be for the network to dynamically focus more on those patterns that have a high error. In order to achieve this, the learning rate is adapted for each pattern depending upon the associated error. In previous experiments, a constant learning rate of 0.35 was used.

The learning rate for each input pattern is calculated as follows :

(E3.2)

$$\eta = \frac{\sum_{i=0}^{q-1} |e_i|}{q} * c1 + c2$$

where e_i is the error for output unit *i*, *q* is the total number of output units and *c1* and *c2* are constants which allow adjustment of the range of η. *c1* = 0.6 and *c2* = 0.2 were chosen to give $0.2 \leq 0 \geq 0.8$. Therefore between 0.2 and 0.8, η is linearly dependent upon the average absolute output error. It is expected that using an adaptive learning rate will enable the MLP network to be sensitive to individual patterns, which may be of a minority type in the overall training set. Although the current emphasis here is on the average absolute output error, formula E3.2 could be adapted to refer to the error on an individual output unit or units and thus narrow the focus to specific output unit(s) with high errors.

The results obtained are encouraging. The training set consists of all 17 positive examples (no replication) and 50% of the remaining 4,079 negative examples as before. This time an RMS error of 0.001 is reached after 3,000 epochs and the MLP network had learnt all training patterns using 6 hidden units. The number of epochs is higher than when using replication indicating that the problem is harder to learn due to the lack of valid terms in the training data. The network generalises exceptionally well with an accuracy of 97.3%, which is a small decrease of 2.7% compared with the best produced from the replication technique. A number of experiments were performed and the generalisation performance was consistently lower by 2-3% than with replication. The plausibility of a universal use for the adaptive learning rate method will become apparent when applying it to other problem domains, notably the natural language domain.

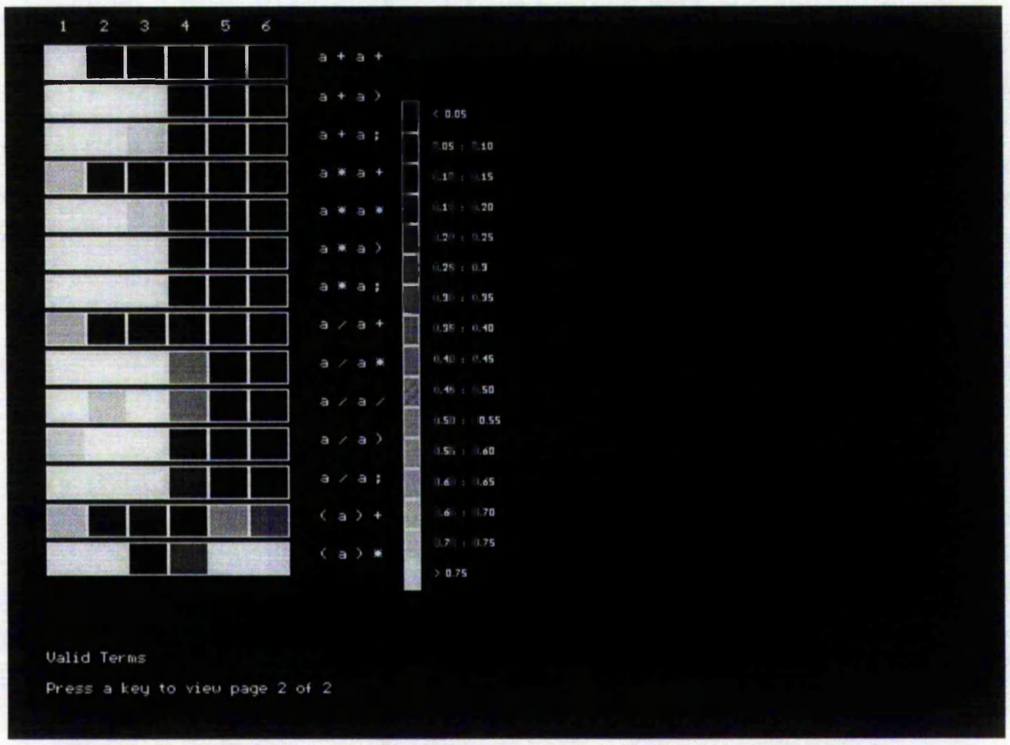
Well established second order methods exist for adapting the learning rate in accordance with how the error changes with respect to individual weights, e.g., delta-bar-delta [156]. However, the method proposed in this section is a second order method in terms of adapting the learning rate according to error changes on individual patterns.

3.2.5. Representation Analysis

The MLP network with 6 hidden units automatically acquired its knowledge about arithmetic expressions directly from the training examples. This knowledge is compactly represented by 85 floating point weight values. Each weight value within the network corresponds to a connection link between two units within the MLP network. During the training phase, these connection links were adapted in order to increase or decrease the influence each unit has on the overall network response to a particular pattern. This type of knowledge encoding provides an efficient, compact and robust representation which allows fast retrieval of information [23].

The weight vector associated with each hidden unit encodes some feature of the input data and the product summation of the input vector and the weight vector determines whether a hidden unit will activate, and to what degree it will activate. Through analysing the activation pattern across the hidden layer it is possible to detect which hidden units respond to which input pattern. It is expected that similar input patterns will activate the same hidden units, as they possess the features that the weight vector to those hidden units encode.

Figure 3.5 shows the pattern of hidden unit activations for 14 valid arithmetic terms produced by the MLP network with 6 hidden units. Each row corresponds to the hidden unit activations for an input pattern.



Valid Terms

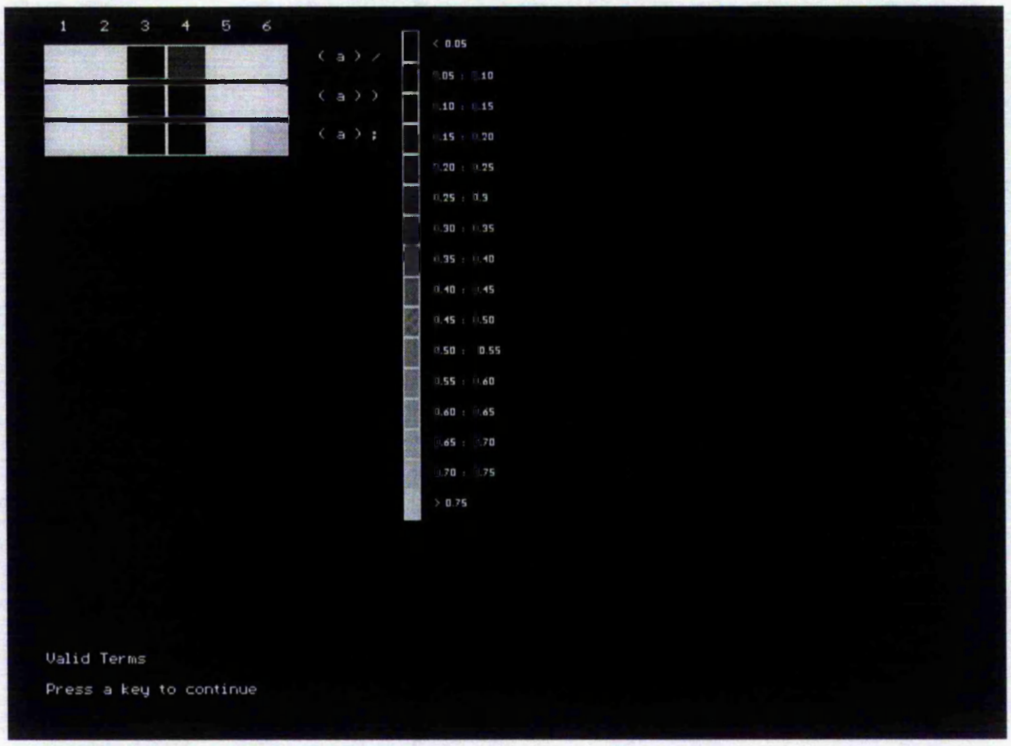
Press a key to view page 2 of 2

Figure 3.5 : A set of hidden unit activation patterns in response to 14 valid arithmetic terms.

Training set #2 (see tables 3.4 and 3.6) was used to train the MLP network in order to produce these hidden activation responses. Figure 3.5 illustrates the hidden unit activations for fourteen valid terms. It can be seen that hidden unit 1 activates strongly for all patterns. However, those patterns which possess a symbol with the lowest operator precedence, i.e., +/-, at the end of the term are strongly encoded within the connection links from the input units to hidden unit 1. An exception to the case is when a symbol is enclosed with parentheses, where it is apparent that units 5 and 6 are required to detect parentheses.

Those arithmetic terms that contain other input symbols at position 4 of the term, and a binary operator at position 3, are strongly encoded by hidden units 1,2 and 3. Arithmetic terms that contain symbols of a high precedence such as *%* and / or a closed parenthesis at position 4 of the term, are additionally represented by hidden unit 4.

The hidden unit response to the last pattern, *(a)**, in figure 3.5 together with the remaining valid arithmetic terms in figure 3.6 show that hidden units 1, 2, 5, and 6 are required to detect expressions encased within parentheses; hidden unit 4 is also slightly activated for each of these patterns.



Valid Terms

Press a key to continue

Figure 3.6 : The set of hidden unit activation patterns in response to the remaining 3 valid arithmetic terms.

It is apparent that *all* hidden units are involved in representing valid terms. A hidden unit activation can be viewed as the probability of an input pattern possessing the feature its weight vector encodes. Figure 3.7 illustrates how the MLP network has formed a representation for rejecting invalid terms.

It can be seen that for the sample of 14 invalid terms, hidden units 1 to 4 activate with varying degrees of intensity to reject those binary terms which have a binary operator with a higher precedence at position 4. For terms that contain two consecutive binary operators, encased by variable symbols, unit 2 activates intensely to provide the majority of the activation response required to correctly classify the terms as invalid. Hidden unit 2 responds to all of the invalid samples, except *+a)+*, which is detected by units 4,5, and 6. Figure 3.7 illustrates how all the hidden units, and consequently the features encoded within each hidden weight vector, combine to represent the knowledge required in order to perform the classification task. Euclidean distance[15] measures of the hidden unit activations were performed between each input pattern to obtain an accurate measurement of similarity between the response vectors. This analyses showed that similar hidden responses were produced for similar input patterns and that a particular feature of the input is being encoded.

[15] The Euclidean distance measure between two vectors *a* and *b* is calculated as follows : $\sqrt{\sum_{i=0}^{N-1}(a_i b_i)^2}$

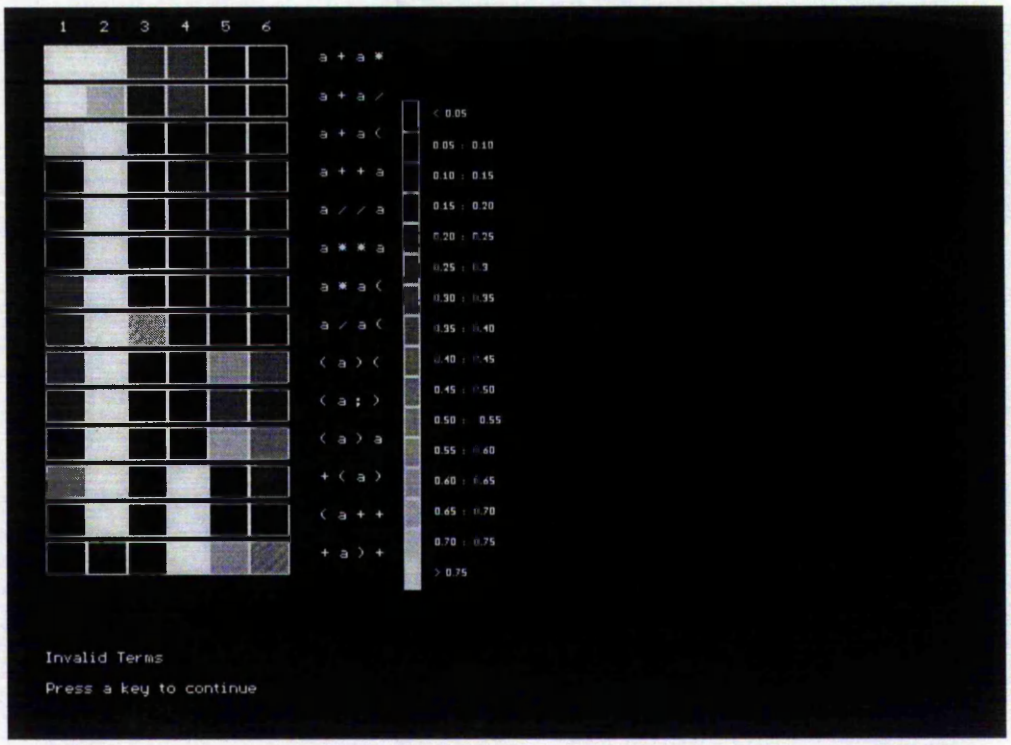


Figure 3.7 : A set of hidden unit activation patterns in response to 14 invalid arithmetic terms.

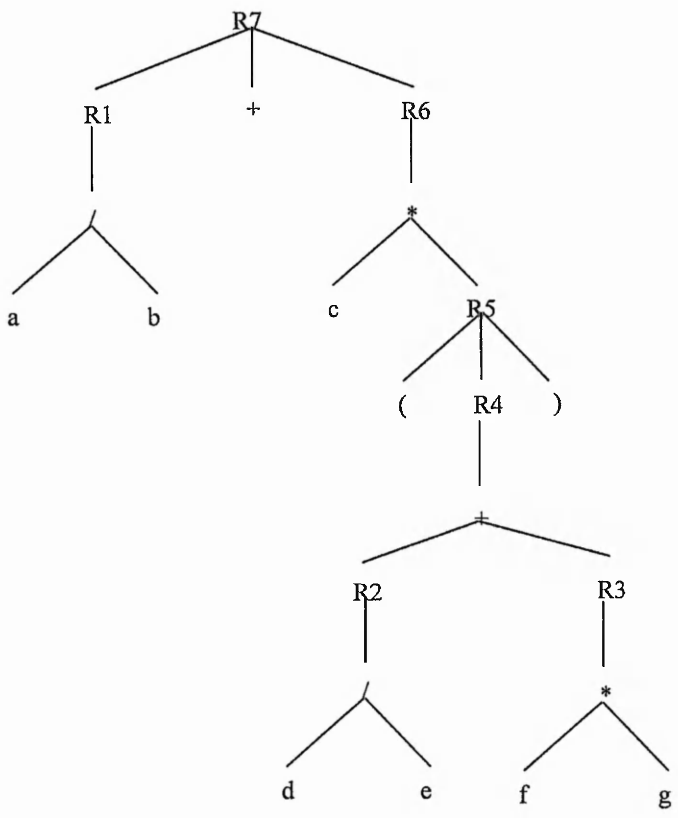
3.2.6. Parse Tree Derivation from The Hybrid Parser

During parsing, input symbols are constantly grouped and reduced in a manner dictated by the MLP network. Symbolic structures such as the parse stack are utilised to store intermediate parsing states, and the reductions performed at previous parse stages.

Figure 3.8 depicts how the parse tree is constructed from the symbolic reduction table for the expression *a/b+c*(d/e+f*g)*. The following can be seen from the reduction table : the order in which reductions were carried out i.e., all division operations were performed before addition; how each reduction relates to each other in the parse i.e., R4 is composed of an addition operation on R2 and R3; which in turn are descriptions of the previous arithmetic terms.

ID	REDUCTION
R1	a / b
R2	d /e
R3	f*g
R4	R2+R3
R5	(R4)
R6	c*R5
R7	R1+R6

Reduction Table



Expression : a/b+c*(d/e+f*g)

Figure 3.8 : A parse tree interpretation of the symbolic reduction table.

3.2.7. Discussion

A hybrid shift-reduce parser for arithmetic expressions was developed. A three-layered feed-forward MLP network was successfully trained with the Back-propagation learning algorithm to automatically learn fixed-length valid and invalid arithmetic terms. Reductions indicated by the MLP network were carried out symbolically, utilising a symbolic stack structure to store unprocessed input constituents and a symbolic reduction table to store grouped input symbols.

The standard Back-propagation learning method was initially unable to learn all of the valid terms as they were disproportionally represented in the training set. Two successful methods to overcome this were presented. One involved the replication of valid expressions to even the imbalance of different training patterns. This increased the training set size and the level of success was dependant upon the degree of replication. With an appropriate replication level the training set was fully learnt and the MLP network achieved a generalisation rate of 100%.

The second method involved adapting the learning rate for each pattern and its error. This had the following characteristics :

- Patterns from classes that are disproportionally represented in the training set are learned. This overcomes the problem of imbalance in the training set.

- There is no increase in the size of the training set.

- Generalisation performance degrades slightly. Generalisation rates of 2-3% lower than those with replication (e.g., 97.3%) were achieved.

- It is a systematic way of overcoming training set imbalance.

3.3. A Hybrid Shift-Reduce Parser for Natural Language

Section 3.2 demonstrated that the MLP network is able to act as a perfect recogniser for a simple context-free language describing arithmetic terms where no ambiguity is present. However, the role of the MLP architecture will now be extended to identify phrases from a simple natural language grammar within a similar shift-reduce parsing framework. However, the task for the MLP network is more difficult than that for arithmetic expressions as the network is expected to recognise the *position* of the phrase and the specific phrase type. As an additional experiment, another MLP network is also used for lexical disambiguation.

This section details the grammatical framework, parsing architecture and algorithm and the training and test performances of both MLP networks.

3.3.1. Grammatical Framework

The use of *toy grammars* is prevalent in the development of connectionist architectures for processing natural language (see Chapter 2). Toy grammars are typically context-free and consist of a small number of context-free rules which govern the formation of grammatical strings but whose language coverage is weak. However, numerous connectionists have encoded syntactic problems in the grammar, such as centre embedding and cross-serial dependencies [74, 137]. Architectures can thereby be developed to process complex syntactic problems. A toy grammar represents a small domain of natural language and provides a tractable training and test problem that is useful for establishing the computational adequacy of a system. The feasibility of using the resulting architecture to process realistic subsets of natural language can then be assessed.

The toy grammar implemented in this section is used to further establish the adequacy of the basic architecture developed for parsing arithmetic expressions, and to also assess the practicality of using rule-based grammars as the basis of training data for an adequate connectionist large-scale natural language parsing system.

The context-free grammar used by Fanty [9] in his localist parser will be used to form the training and test data sets for the MLP network developed in this section. The grammar used to describe a subset of the English language can be defined as a quadruple *G = (N,T,S,P)* where *N* is a finite set of non-terminal symbols; *T* is the finite set of terminal symbols; *S* is the starting symbol, which must be a member of the set *N*, and *P* is the set of production rules. The grammar used is defined in figure 3.9. Each rule can be used to derive a set of training patterns. These patterns consist of a binary encoding of terminal and non-terminal symbols in the grammar. In the context of natural language, terminal symbols are commonly referred to as *word tags* as they represent the syntactic category of a word. Non-terminal symbols are referred to as *constituent tags* as they represent a grouping of terminal and/or non-terminal symbols. These terms will be used interchangeably from hereon.

$$
\begin{array}{lll}
\mathrm{N} = & \mathrm{S, VP, PP, NP, NP2} & \\
\mathrm{T} = & \multicolumn{2}{l}{\textit{verb, noun, preposition, determiner, adjective}} \\
\mathrm{S} = & \mathrm{S} & \\
\mathrm{P} = & \mathrm{S} \longrightarrow & \mathrm{NP\ VP} \mid \mathrm{VP} \\
 & \mathrm{VP} \longrightarrow & \textit{verb} \mid \textit{verb}\ \mathrm{NP} \mid \mathrm{VP\ PP} \\
 & \mathrm{PP} \longrightarrow & \textit{preposition}\ \mathrm{NP} \\
 & \mathrm{NP} \longrightarrow & \text{determiner}\ \mathrm{NP2} \mid \mathrm{NP2} \mid \mathrm{NP\ PP} \\
 & \mathrm{NP2} \longrightarrow & \textit{noun} \mid \textit{adjective}\ \mathrm{NP2}
\end{array}
$$

Figure 3.9 : A simple context-free grammar to describe a subset of the English language.

3.3.2. The Parsing Architecture And Algorithm

The MLP architecture used to recognise valid and invalid arithmetic expressions is extended for the task of recognising the input constituents involved in a reduction and the grammar rule to apply to the input constituents according to the grammar defined in figure 3.9. It is also expected to reject invalid phrase structures and thus perform phrase validation. As an additional experiment, the problem of lexical ambiguity is also be tackled and a separate MLP network is developed to provide lexical alternatives to ambiguous input words. The resulting symbolic-connectionist hybrid architecture consists of the following components :

- A temporal input window of 12-bits. It is evident from the grammar illustrated in figure 3.9, that at most three input symbols (i.e., combinations of word tags and/or constituent tags) needs to be present in the temporal input window per time step. The input can therefore be referred to as a trigram of input combinations. Each symbol is represented by four bits.

- A three-layered MLP network trained with the Back-propagation learning algorithm whose task is to recognise valid and invalid input constituents and activate output units representing the appropriate grammar rule and inputs to group. This component is termed the *recogniser*. As input, it accepts a bit representation of the three symbols in temporal input window. It activates 8 output units in response to the contents of the temporal input window.

Five represent a rule of the context-free grammar. The other three represent positional information.

- ♦ A three-layered MLP network forms the *lexical disambiguation module*. This is a transformation network, where invalid input will be transformed into its most likely valid alternative. As well as the 12-bit temporal input window it has three additional input units, called *clamp units*. There are 12 output units, which represent an alternative to the input pattern. The connectionist recogniser and disambiguation module function in a synchronised manner.

- ♦ A symbolic reduction table to store and uniquely label reductions. This forms a symbolic representation of the parse tree for the sentence.

- ♦ A symbolic lexicon to provide an *initial* syntactic category for input words and validate alternatives. The symbolic lexicon does not recognise syntactic context and thus simply acts as a look-up table. The disambiguation module provides alternatives for words that possess an incorrect word tag (i.e., syntactic category). The lexicon then provides a true or false response to lexical alternatives for a word. The lexicon is implemented as a linked list of record structures.

Algorithm

The system parses sentences from Right-to-Left by iteratively processing word tags and constituent tags from the input buffer. The Right-to-Left strategy has been selected due to the nature of the context-free grammar being used. It removes the need for a symbolic stack, as constituents belonging to the right-most position of recursive rules are processed first. For example, in order to process an arbitrary number of adjectives without the use of a symbolic stack, a Right-to-Left parsing strategy always forces the noun to be reduced to an *NP2* before the adjectives are combined with the noun and reduced to a corresponding *NP2* structure.

The initial syntactic category of an input word is provided by the symbolic lexicon. The categories available are those defined in the grammar (i.e., *noun*, *preposition*, *verb*, *determiner* and *adjective*). The symbolic lexicon consists of a linked list which contains the words that can be recognised by the parser. The linked list consists of records that store : the word, its most probable syntactic category (limited by the grammar), and other syntactic roles that the word can be bound to. The conversion of word to syntactic category is performed symbolically. Each input symbol (syntactic category or phrase group) is represented by a four bit binary code.

The three right-most input constituents are extracted from the input buffer and placed into the temporal input window. Firstly, a forward-pass of the connectionist recogniser is computed and the output units of the recogniser are then analysed to detect whether a reduction can be performed. If a reduction is possible, then the recogniser will activate the output unit that corresponds to the appropriate grammar rule. The recogniser will also activate the output units that correspond to the position of the input symbols to be reduced. There will be at most a reduction of 2 input symbols to 1 non-terminal symbol. A further *n-1* sentence constituents can then enter the temporal input window from the input buffer, where *n* is the number of symbols in the reduction. Consider the example sentence, *Liam ran to the toy shop.* By analysing the grammar given in figure 3.9, it can be seen that the recogniser is initially expected to output an activation response which corresponds to an NP2 reduction at position 3 of the temporal input window. This example is illustrated in figure 3.10.

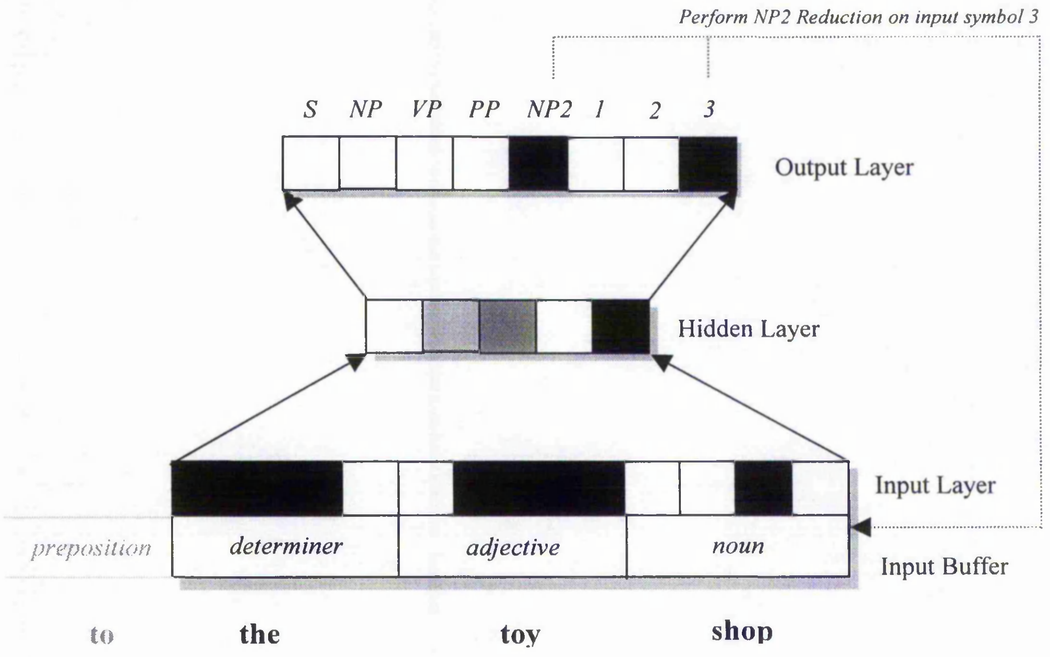


Figure 3.10 : The MLP network as a grammar rule recogniser for a shift-reduce parser. Noun at position 3 is reduced to NP2.

The input symbols belonging to the reduction are then placed in the symbolic reduction table. The reduction table is a symbolic linked list of records. Each record consists of the input constituents belonging to the reduction, the type of reduction, a unique label[16], and the bit representation of the type of reduction performed.

If the MLP network cannot recognise the three input symbols as belonging to a grammar rule, one of two states is assumed :

i. The initial syntactic category assigned by the symbolic lexicon for a word within the temporal input window was incorrect.

or

ii. The input is irreducible and therefore invalid.

[16] A unique label within the reduction table consists of the relevant grammar rule (e.g., *NP*) with an integer appended to it indicating its identity. For example, entry *NP2_3* $\rightarrow$ *adjective NP2_2* would indicate the third occurrence of the grammar rule *NP2*. This uniquely identifies parent nodes within the parse tree.

In either of these cases, the response from the connectionist disambiguation module is then analysed to detect whether it can provide lexical alternatives for the words associated with the contents of the temporal input window. This is performed using three additional input units that *clamp* the constituents in the temporal input window, see figure 3.11. Each clamp unit corresponds to a position in the temporal input window and if activated, prevents the disambiguation module from suggesting an alternative for that input. For example, if clamp unit one is activated then the syntactic category of the constituent in position one (leftmost) of the temporal input window cannot change.

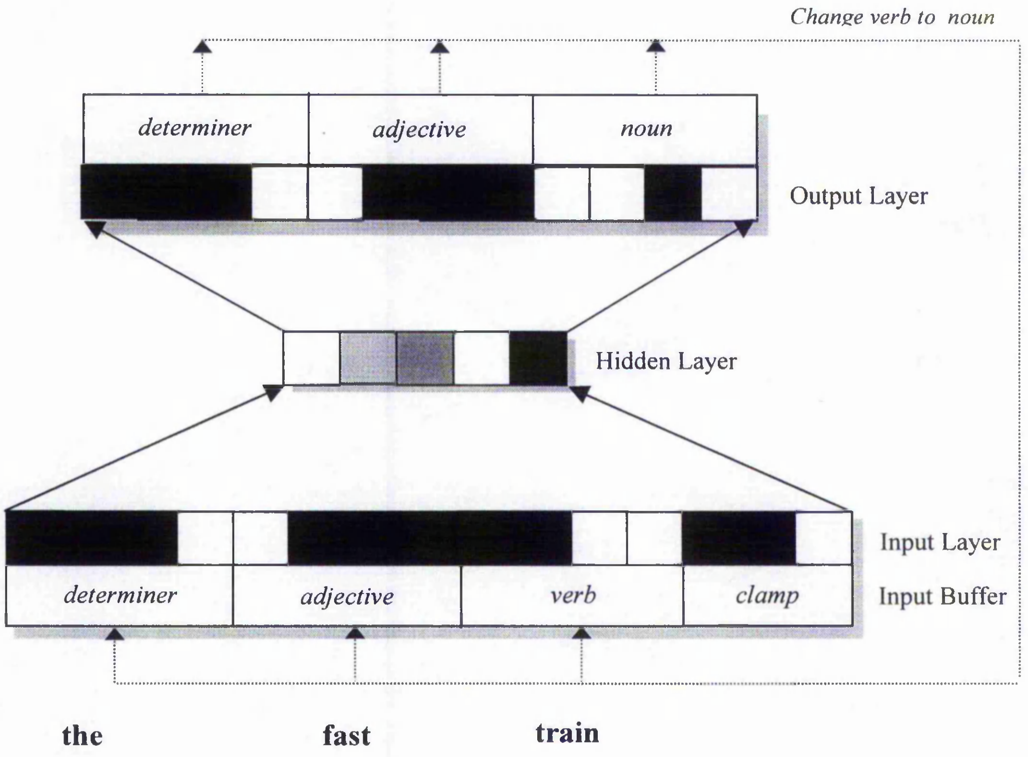


Figure 3.11 : Disambiguation module providing an alternative for position 3 (positions 1 and 2 are clamped).

In order to find a likely alternative to the syntactic categories on the input when the recogniser is unable to recognise the contents of the temporal input window, two clamp units are set to 1 (activated) to allow the third input symbol to change its syntactic category (word tag). The order of clamping is from Right-to-Left i.e., the input symbol in position three of the temporal input window is allowed to change first on the output layer as input symbols 1 and 2 are clamped. The disambiguation network will therefore perform a transformation that may provide an alternative syntactic category for the word at position three. Figure 3.11 shows a simple example for the sentence, *Jonathan caught the fast train.* The word *train* was initially assigned a *verb* category as the symbolic lexicon does not take context into account when providing the syntactic category for each word. Clamp unit three is then set to 0 and the MLP network is thus able to produce a *noun* category for position three due to the knowledge of syntactic context learnt during the training phase. When a lexical alternative has been provided for a particular word, the lexicon must then be referred to. In the example, the lexical look-up would confirm that *train* can be assigned a *noun* type.

However, if the syntactic category is rejected by the lexicon the output of the disambiguation module is ignored. The third clamp unit on the input layer is then activated to fix the third symbol in the temporal input window and the second clamp unit is set to 0 so that the MLP network is able to provide an alternative for input symbol 2 of the temporal input window and so on.

After a transformation has been successfully performed by the disambiguation module and the content of the temporal input window on the output layer is different to that on the input layer, the connectionist recogniser is presented with the modified temporal input window.

Essentially, the disambiguation process repeats until either :

i. a successful transformation has been performed in that the recogniser network has been able to activate the appropriate grammar rule to reduce the trigram in the temporal input window

or

ii. the disambiguation process has completed and a reduction is still not possible.

In the latter case the sentence is rejected by the parser on the basis that it is ungrammtical.

If after a reduction has been processed, the temporal input window is able to accommodate further input symbols, these are extracted from the input buffer.

A successful parse is indicated by the following conditions :

- The *S* output unit of the connectionist recogniser is activated.
- The terminating symbol, '.' at the beginning of the sentence, has been encountered.

An unsuccessful parse would be indicated by either of the following conditions :

- The input symbols in the temporal input window cannot be reduced or transformed into a valid alternative.
- The terminating symbol has not been encountered after all input symbols have been processed.

The above algorithm will only work if the recogniser and disambiguation module have learnt the grammar given in figure 3.9. The training and test performance of the connectionist recogniser and disambiguation module is discussed in sections 3.3.4 and 3.3.5 respectively.

3.3.3. Input Representation

Although the hybrid architecture is expected to parse sentences of arbitrary length, only three input symbols in the temporal input window are processed per time step. The temporal input window is a trigram which can consist of a mixture of terminal (e.g., *verb*) and non-terminal symbols (such as *NP*).

Symbol	**Description**	**Input Representation**
S	Start Symbol	1 0 0 0
NP	Noun Phrase Group	0 1 0 0
VP	Verb Phrase Group	0 0 0 1
PP	Preposition Phrase Group	0 0 1 1
NP2	Noun Phrase Sub-Group	1 0 1 0
noun	Noun	0 0 1 0
verb	Verb	1 1 0 0
prep	Preposition	1 0 1 1
adj	Adjective	0 1 1 1
det	Determiner	1 1 1 0
.	Terminator	1 1 1 1

Table 3.7 : Input representation of allowed input symbols.

It can be seen from table 3.7 that there are 11 unique input symbol representations, thus the network can be presented with a maximum of 11^3 trigrams, which equates to 1331 unique possible training patterns. The input pattern consists of a binary encoding of the contents of the temporal input window.

A similar training strategy to that taken with the arithmetic parser will be taken here. The training data for both of the MLP networks will mainly consist of *all* valid phrases of length three, and a subset of the invalid phrases. Each training pair will consist of a bit representation of the temporal input window and the desired output response.

Valid training pairs for the recogniser network consists of an input pattern that is reducible according to the grammar and an output pattern that corresponds to the appropriate grammar rule and input symbols (e.g., *det adj NP2* forces a *NP2* reduction for symbols *2* and *3*; . *NP VP* forces *S* reduction for symbols *2* and *3* and indicates a successful parse). Invalid training pairs for the recogniser network contain distracter patterns. Distracter patterns are input patterns that are not reducible and whose output pattern consists of 0's to inhibit activation and thus represent no reduction. Typically, a distracter pattern will be a sequence of the terminal and non-terminal grammar symbols which would not occur in a valid sentence according to the grammar (e.g., *n NP p*, *p . p* , or *p p p*). A total of 61 valid phrases have been manually defined as belonging to the grammar. The total number of invalid phrases (distracter patterns) is therefore 1,270.

3.3.4. The MLP Network for Simple Phrase Identification

The task of the feed-forward MLP network is to recognise whether a grammar rule can be applied to any of the input symbols and to determine which symbols the rule will be applied to. This MLP network is termed the connectionist recogniser and is expected to produce an output activation above 0.5 on the output unit that corresponds to the appropriate grammar rule and on the output units that correspond to the reducible input symbols. Activations below this threshold on every output unit, will indicate an invalid phrase on the input units. There are five possible rules that may be activated. So the training data will consist of six pattern classes, one to represent each of the possible rules, *S, VP, PP, NP* and *NP2*, and one to represent invalid input phrases. An additional three positional output units result in a total of eight output units.

As in Section 3.2.3, the MLP network architecture consists of three layers of units and two layers of weights. The Back-propagation learning algorithm is selected to train the MLP network. Learning will take place in on-line mode, and the learning rate and momentum term will initially be set to 0.35 and 0.9 respectively. The following sub-sections discuss the training and test performance of the connectionist recogniser.

3.3.4.1. Training And Test Experiments

As with the experiments on arithmetic expressions there is an imbalance between the number of valid and invalid patterns. The two strategies developed to overcome this are again used. These were : pattern replication and pattern-error sensitive learning rates. Five training strategies were tried. Strategy 1 and 5 used the pattern-error sensitive learning rate, 2 to 4 used pattern replication (as shown in table 3.8).

The first training set contains *all* 61 valid phrases (i.e., valid phrases taken from the grammar), and 49.6% of all invalid phrases (containing distracter patterns). As with training set 1 of the arithmetic expression parser, this produces a severe imbalance in training patterns, with 91.8% of the training set consisting of invalid phrases. This is reflected in training set #1 described in table 3.8. Table 3.8 details the training sets used in the experiments undertaken.

The pattern-error sensitive learning rate, *Adapt#1,* is identical to that used for learning arithmetic expressions (see equation E3.2). However, *Adapt#2* is an extension of *Adapt#1,* in that each weight layer possesses its own corresponding pattern-error sensitive learning rate. For example, if the output layer is the current layer then e_i is the error for output unit i , else e_i is the error for hidden unit i in hidden layer j. Therefore the learning rate for the weight matrix k is dependant upon the errors in the unit layer they are connected to. η is therefore linearly dependent on the average absolute *hidden or output error.*

Training Strategy	**Total Patterns**	**R**	**η**	**Valid Phrases**	**Invalid Phrases**	**% Valid**	**% Invalid**
1	691	0	Adapt #1	61	630	8.8	91.8
2	1,850	20	0.35	1,220	630	65.9	34.1
3	3,070	40	0.35	2,440	630	79.5	20.5
4	3,680	50	0.35	3,050	630	82.9	17.1
5	691	0	Adapt #2	61	630	8.8	91.8

Table 3.8 : Training strategies defined for the MLP network to learn the recognition task.

3.3.4.1.1. Pattern Replication

As mentioned previously, the level of pattern replication is dependant upon the problem domain, and an optimum level has to be empirically established. The considerations taken into account in the experiments performed with arithmetic expressions will also be considered for the natural language domain.

For all experiments, the learning rate and momentum term is set to 0.35 and 0.9 respectively. Since there are only 61 possible valid input patterns, in contrast to 1,270 invalid patterns, only the valid phrases pattern class is replicated. All training set configurations consisted of 49.6% of invalid phrases (630 patterns) to ascertain as to whether the network is able to generalise to the remaining 50.4% invalid phrases.

The network must be able to recognise valid phrases and reject all invalid phrases. It is only expected to generalise to reject invalid phrases.

A number of training experiments were performed to ascertain the optimum balance of valid and invalid training examples. Table 3.9 shows some typical results obtained from a number of network configurations trained with the training sets defined in table 3.8. It is evident that using 15 hidden units provided almost complete learning of the training set (99.52%) and an acceptable level of generalisation (92.25%).

After numerous experiments, **R** set to 20 for the valid phrases, proved to be the optimum level for this domain. Replication levels below this did not improve the training set imbalance. This is illustrated in figures 3.12 and 3.13.

Training Strategy	**Hidden Units**	**RMS After 10,000 Epochs**	**% Training Data Learnt**	**% Generalisation**
2	12	0.12	99.04	90.67
3	12	0.17	98.25	92.25
4	12	0.13	98.57	90.04
2	13	0.26	97.93	93.04
2	14	0.29	98.57	92.57
3	14	0.12	98.41	91.94
4	14	0.12	99.36	90.67
2	15	0.03	99.52	92.25

Table 3.9 : The tabled results correspond to the learning parameters used for the MLP networks together with performance results.

Although using 12 hidden units provided the most compact representation of the training data, the generalisation performance was lower than configurations containing a higher number of hidden units. The optimum number of hidden units was considered to be 15 since it is the smallest number that gives virtually complete coverage of the training data *and* maintains a generalisation rate above 92%.

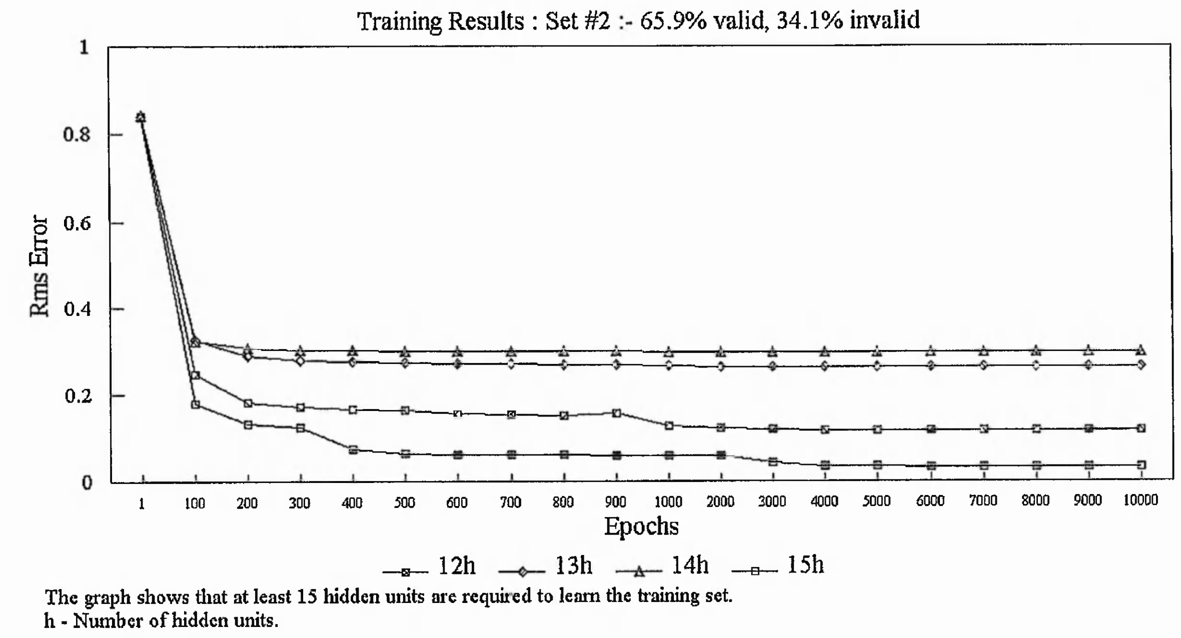
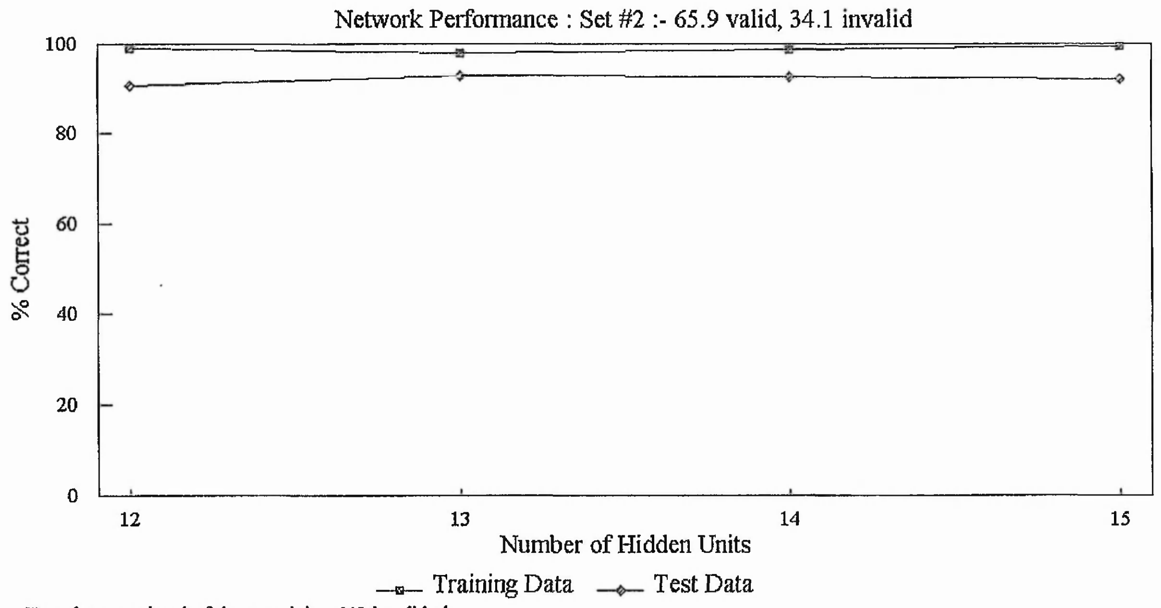


Figure 3.12 : Training performance of four MLP network configurations for the recognition process.

Using 17-20 hidden units provided no significant increase in performance and exhibited similar behaviour to the other network configurations (15 or less).



Test data consisted of the remaining 640 invalid phrases.

Figure 3.13 : Performance of the MLP network configurations trained using training strategy #2.

3.3.4.1.2. Pattern-error Sensitive Learning Rates

Table 3.10 shows the results of the experiments undertaken with the pattern-error sensitive learning rate. The beginning of section 3.3.4.1 presented two forms of the pattern-error sensitive learning rate; *Adapt#1* which is proportional to the absolute output error, and *Adapt#2*, which is proportional to either the absolute output error *or* the absolute error on the hidden units depending upon the current connection links being adapted.

Training Set	Hidden Nodes	Learning Rate	RMS after 10000 Epochs	% Training Data Learnt	% Generalisation
1	12	Adapt#1	0.25	94.6	96.68
1	15	Adapt#1	0.22	96.82	97.47
5	15	Adapt#2	0.25	95.55	96.68

Table 3.10 : Results with pattern-error sensitive learning rate.

The momentum term was fixed to 0.9 through all training sessions. It can be seen that although a relatively high RMS was attained for all experiments, using a learning rate dependant upon the absolute error (Adapt#1) with 15 hidden units provided a higher level of generalisation than other experiments. However, this is at the expense of the training set as not all training patterns were learnt. Using Adapt#1 consistently enabled the MLP networks to generalise better than when using Adapt#2 or even replication.

3.3.4.1.3. Summary of Results

Replication of training pairs provided more complete learning of the training set in comparison to using pattern-error sensitive learning rates. This is in contrast to the results achieved during the training experiments for the arithmetic expression parser, where pattern-error sensitive learning rates provided complete learning of the training set (with the consequence of lower generalisation rates than pattern replication).

Pattern-error sensitive learning rates provided, on average, 5% higher levels of generalisation compared with that of pattern replication. It was also evident that the higher the level of replication the lower the overall generalisation performance. As proved with the arithmetic expression parser, the more biased the network is to invalid phrases the greater the generalisation performance. This also accounts for the generalisation performance of the networks which have pattern-error sensitive learning rates; invalid patterns are more predominant than valid cases.

Experiments with 15 hidden units (203 floating point weight values), both with pattern replication and pattern-error sensitive learning rates, provided almost complete learning of the training set and better generalisation results.

It is possible that a combination of pattern replication and adaptable learning rates is required together with the appropriate architecture to find an optimum connectionist solution to the problem.

3.3.5. The MLP Network for Lexical Disambiguation

The task of this MLP network is to provide lexical alternatives for lexically ambiguous or invalid input. It acts as a transformation network, whereby ambiguous or invalid input will be transformed into its most likely valid alternative. It accepts a binary representation of the temporal input window as input; integrated with three additional units, called clamp units (see figure 3.11). Therefore, there are eight possible unique input patterns per input phrase, one for each clamp unit combination. There are 12 output units, to provide an alternative input pattern per time step.

3.3.5.1. Training And Test Experiments

The temporal input window consists of three input symbols which may be a combination of syntactic tags and constituent tags (e.g., *determiner adjective NP2*). The combination of input symbols may be grammatically correct or incorrect according to the grammar in figure 3.9. If the combination is syntactically incorrect it may be due to an invalid syntactic category assigned by the lexicon. Each training pair consists of an input pattern consisting of a bit-representation of the temporal input window together with a clamp unit configuration and a corresponding output pattern that represents the desired modification of syntactic tags in the temporal input window. For example, the phrase, *adjective noun noun*, may be the input pattern, for which *adjective adjective noun* is the desired modification; hence a noun in position two has been converted into an adjective. This requires clamp unit 2 to be set to 0 to allow position two to change its syntactic category.

Consequently, three types of phrases have been identified that can be presented to the MLP network with an appropriate clamp unit configuration for it to learn the lexical disambiguation task. The three phrase types are :

- *Valid phrases.* These provide the network with phrases to which a grammar rule can be successfully applied. For the disambiguation network, there are 62 valid phrases to which a rule can be applied.

- *Modifiable phrases.* These are phrases to which a grammar rule cannot be applied without some form of modification. Modifications can only be applied to terminal symbols within the input phrase, as non-terminal symbols represent reductions from previous time-steps. There are a total of 47 modifiable patterns, which are based on 'distorted' valid phrases. A possible 7 training patterns are associated with each modifiable phrase, as the input patterns with a clamp unit combination of '1 1 1' force no change on the output units. A modifiable invalid phrase will yield different valid alternatives depending upon which inputs are clamped.

- *Distracter phrases.* For the disambiguation module, a distracter phrase is one that consists of a random sequence of input symbols for which no grammar rule can be applied even with slight modification (i.e., one symbol being changed). There are 1,212 possible distracter patterns each with 8 possible training patterns. A total of 606 distracter phrases (50%) were used for training data, and the remaining 50% of phrases were used for testing. The disambiguation module is expected to repeat distracter phrases on the output layer irrespective of which inputs are clamped. This will force the connectionist recogniser to signify no reduction and thus a failed parse.

The training set configurations used can be seen in table 3.11.

Training Strategy	**Valid Phrases**	**Modifiable Phrases**	**R**	**Total Valid Phrases**	**Distracter Phrases**	**% Valid**	**Number Of Test Phrases**
1	496	165	10	6,610	4,848	57.7	5,072
2	248	165	12	4,956	4,848	50.5	4,864
3	496	165	4	2,644	4,848	35	5,072
4	496	329	6	4,950	4,848	50.5	4,848
5	496	329	3	2,475	4,848	33	4,848

Table 3.11 : Training sets defined to find the optimum level of replication, **R**, of valid and modifiable patterns.

All training sets include 50% of the distracter patterns. Training sets 1 and 2 contain the highest replication factor of valid phrases. Training set 1 contains all valid phrases, 50% of modifiable phrases and 50% of distracter phrases; it is therefore biased toward valid phrases by 7.7%. The test phrases consist of the remaining 50% of modifiable phrases and the 50% distracter phrases. Training set 2 reduces the bias in set 1, by containing 50% of all phrase types. The corresponding test data for set 2 will therefore contain 50% of the remaining valid phrases, 50% of the modifiable phrases and 50% of the remaining distracter phrases. The valid phrases contained in training set 3 constitute just 35% of the entire training set. It contains all 496 valid phrases and 50% of the 329 modifiable phrases together with 50% of the distracter patterns. Replication of the valid phrases is reduced to a factor of 4. The test data will therefore contain the remaining 50% of the modifiable and distracter phrases. Training sets 4 and 5 contain all of the valid and modifiable phrases with a replication factor of 6 and 3 respectively and 50% of distracter phrases. This is to assess the ability of the network to learn all valid phrases, 50% invalid phrases, and to generalise to the remaining distracter patterns. The results of the training and test experiments performed with two MLP network configurations is summarised in table 3.12.

Training Strategy	**Hidden Units**	**RMS After 1000 Epochs**	**% Training Data Learnt**	**% Pure Generalisation**
1	50	0.04	74.6	71.29
2	50	0.03	93.2	86.31
3	50	0.04	82.7	75.47
4	50	0.07	89.5	78.71
5	50	0.19	89.3	83.81

Table 3.12 : Results of the training and test experiments performed with MLP network configurations of 50 hidden units.

All training and test experiments for the disambiguation module have been performed on the CNAPS parallel machine due to its high processing speed. The standard Back-propagation algorithm provided with the CNAPS system was used, therefore all training experiments have been performed with pattern replication to balance the training set as no facility was available to adjust the learning rate formula. Due to time and access constraints, each MLP network configuration was trained for a maximum of 1,000 epochs. The maximum number of hidden units was limited to 50 to make full use of the physical nodes on the CNAPS rather than using software simulation of nodes.

The following points can be made from the results displayed in table 3.12 :

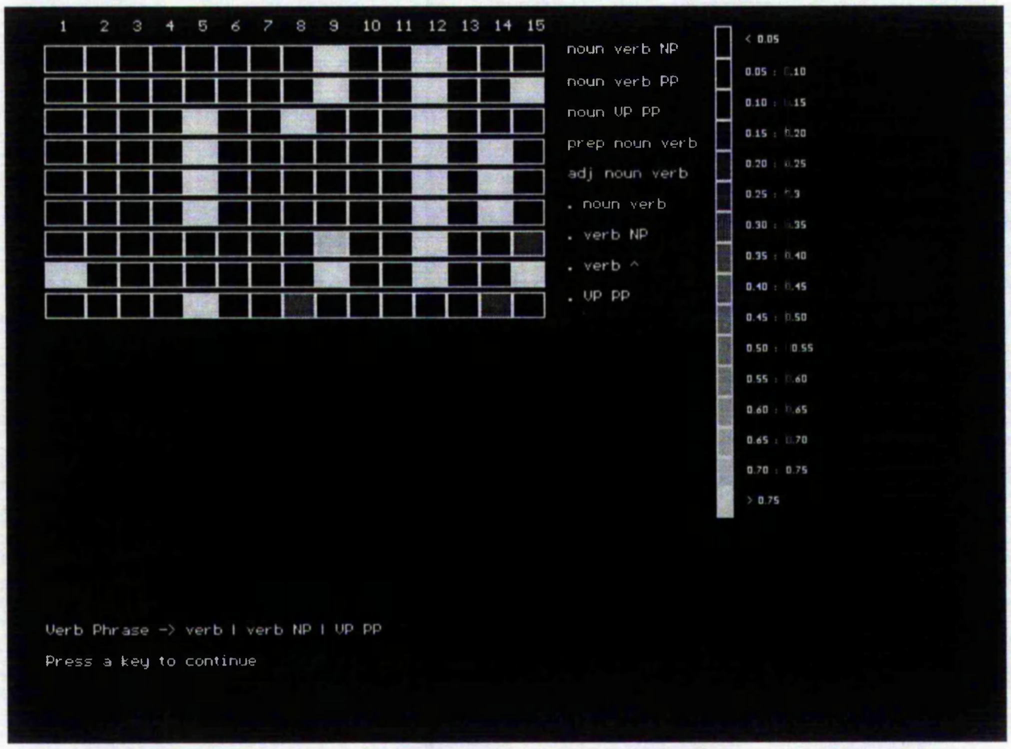
- Within the time and access constraints set, the MLP network is unable to fully learn all of the training data. As expected, the more training data learnt the greater the level of generalisation. It also demonstrates that this is a harder problem to learn than the recognition and segmentation problem explained in Section 3.3.4.

- As with the arithmetic expression parser, it can be seen that the higher the level of replication of valid phrases the lower the generalisation performance. This is because the majority of test cases are distracter patterns, so the more biased the network is to distracter patterns within the training set the greater the generalisation performance. For example, training set 5 contained a lower replication factor than training set 4 and the MLP network trained with training set 5 was able to generalise to 5% more of the test data than that trained with training set 4.

Although the results are disappointing it is envisaged that by allowing longer training times would enable the MLP networks to learn all of the training data and produce higher levels of generalisation.

3.3.6. Representational Analysis

The weight vector associated with each hidden unit encodes some feature of the input data. The product summation of the input vector and the weight vector determines whether a hidden unit will activate, and to what degree. This section briefly analyses the activation pattern across the hidden layer of the recogniser network (see Section 3.3.4) to detect which hidden units respond to which input phrase. It is expected that the MLP network will develop similar distributed representations for input phrases belonging to the same rule of the grammar.



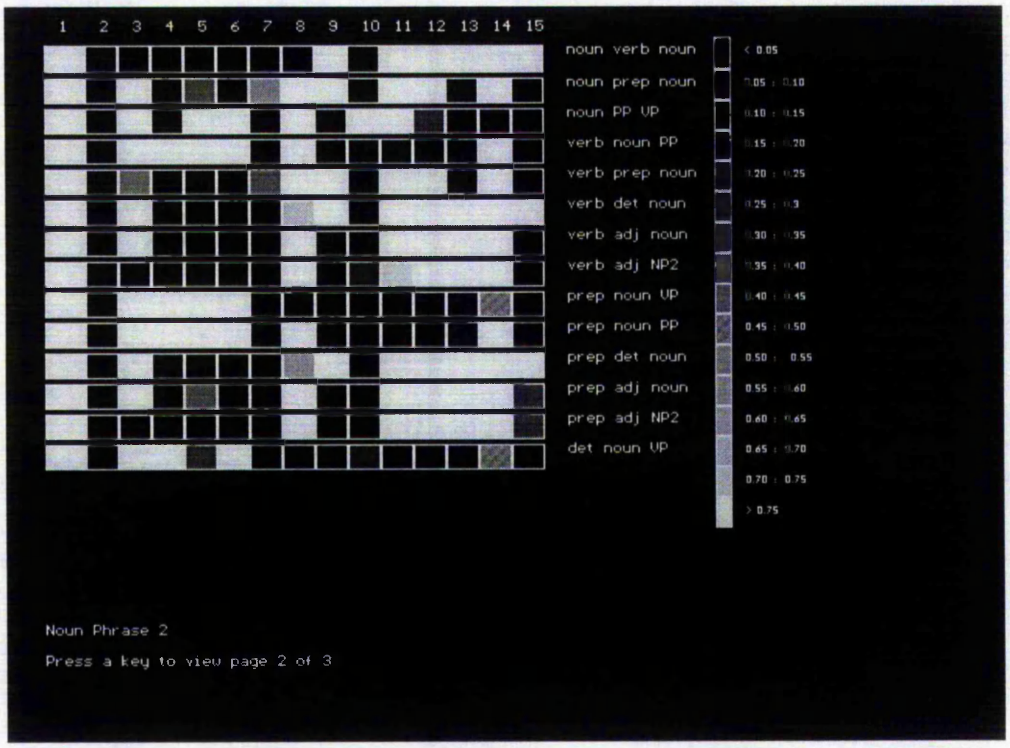
Verb Phrase -> verb | verb NP | VP PP

Press a key to continue

Figure 3.14 : Hidden unit activations formed by the recogniser when presented with trigrams requiring the application of the verb phrase rule (VP) of the grammar (see figure 3.9).

All hidden unit analyses were performed on the connectionist recogniser trained with training set 2 using 15 hidden units (see table 3.9). Figure 3.14 shows the hidden unit activations of the recogniser in response to a valid input phrase. Each column in the grid represents a hidden unit, and is numbered. Each row represents the pattern of activation across the hidden units in response to the input pattern displayed to the right-hand side of the row. It can be seen from figure 3.14 that hidden unit 12 strongly detects the features associated with verb phrases, as do units 5,9, and 14. It appears that the connectionist network is encoding positional information, where the position of the verb determines the pattern of activation that will represent the grammar rule. For example, hidden units 9 and 12 strongly respond to those trigrams where the verb is preceded by a noun and followed by a noun phrase or prepositional phrase. Hidden units 5,12 and 14 activate when the verb is positioned at the end of the temporal input window.

Further similarities of pattern activation can be found in trigrams belonging to the NP2 rule of the context-free grammar. Figure 3.15 shows that hidden units 1,3,11,12,13,14, and 15 strongly activate in response to most patterns belonging to the NP2 rule. It is evident that activation from units 1,3,8,9,11,12,13,14 and 15 combine to represent a noun supported by a determiner.



Noun Phrase 2

Press a key to view page 2 of 3

Figure 3.15 : Hidden unit activations formed by the recogniser when presented with trigrams requiring the application of the noun phrase2 rule (NP2) of the grammar (see figure 3.9).

A noun which is complemented by a preposition is represented by the combined variable activation of units 1,3,7,8,9,11,12 and 14. It can be seen from these distributed representations, that the representation for all grammar rules overlap; and the same hidden units can represent different rules in response to different inputs. As expected, the hidden units appear to encode positional information about the input, dedicating particular units to certain syntactic types and their position. The distributed representation of grammar rules is in contrast to classical symbolic methods of rule representation. The representations overlap and the same units or entities are able to represent different feature information or *rules* at different time steps.

3.3.7. Parse Tree Derivation from The Hybrid Parser

During parsing, input constituents are constantly grouped and 'reduced' in a manner dictated by the recogniser. Symbolic structures are used to store the reductions performed at previous parse states. Figure 3.16 illustrates how the parse tree is constructed from the symbolic reduction table for a sentence with the following syntactic components : *determiner noun verb preposition determiner adjective noun* (e.g., The girl cycled to the paper shop). It can be seen that the sentence has been divided up into subject(*NP_2*) and predicate(*VP_2*), with the prepositional phrase (*PP_1*) belonging to the transitive verb phrase (*VP_2*). The contents of the reduction table show that the parse tree was formed from right to left; the first node created being *NP2_1* to act as parent to the right-most *noun*. The node *NP2_2* was then created to parent the sister nodes, *adjective* and *NP2_1*, and so on.

ID	REDUCTION
NP2_1	*noun*
NP2_2	*adj NP2_1*
NP_1	*det NP2_2*
PP_1	*prep NP_1*
VP_1	*verb*
VP_2	*VP_1 PP_1*
NP2_3	*noun*
NP_2	*det NP2_3*
S	*NP_2 VP_2*

Reduction Table

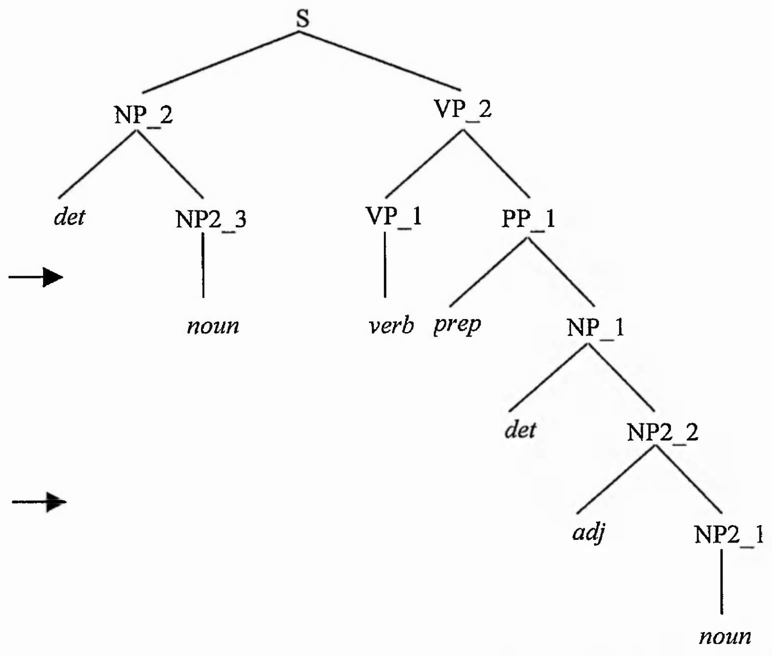


Figure 3.16 : A reduction table and corresponding parse tree formed by the hybrid parser for an input sentence such as, *The girl cycles to the paper shop.*

The valence for all tree formations is limited to two, where each parent node in the tree dominates at most two sister nodes. As with the arithmetic expression parser, the output of the hybrid architecture lends itself to formation of RAAM representations of the input sentence. This would incrementally encode the structure within a set of connection links. Each reduction activated by the connectionist recogniser would relate to a RAAM encoding of the corresponding input constituents. For example, parent node NP2_1 would correspond to a recursive encoding of the right-most noun, to obtain a hidden layer representation, *HNP2_1*. Similarly, NP2_2 would correspond to a recursive encoding of *adjective* and *NP2_1* to form the hidden layer representation *HNP2_2*; and hidden layer representation *HNP_1* would encode *determiner* and *NP2_2* etc. This hidden layer representation ,*HS*, would encode *HNP_2* and

HVP_2 which represents the entire structure of the sentence.

3.3.8. Extending The Context-free Grammar

Further experiments were undertaken to construct a realistic context-free grammar for the English language to assess the ability of the recogniser network (defined in Section 3.3.4) to learn a realistic subset of the English syntax.

There is no complete grammar for the English language [64]. However, the ATN transition diagram defined by Noble [65] represents a large natural language grammar for the English language and is suitable for extracting a set of context-free grammar rules for use as training data. The grammar is a subset of the authoritative English grammar defined by Quirk, Greenbaum, Leech and Svartvik [64]. However, the ATN diagram was incomplete and therefore had to be extended which proved slow, complicated and cumbersome. A total of 41 phrase types (constituent tags) were defined from the ATN diagram and a total of 233 rules were extracted from the ATN (see Appendix C). A temporal input window of 8 input symbols was used and each input symbol was represented by 25-bits and the output layer contained an output unit for each grammar rule. Initial experiments proved disappointing, the MLP network was trained with a small sample of 233 phrases as training data. After numerous training sessions, the MLP network managed to eventually learn 90% of the training data using 30 hidden units.

However, due to the problems encountered with the integrity of the grammar and the slow training times that occurred with larger training sets further experiments were halted. For example, adding further grammar rules required further output units to be added and the training data to be regenerated. This points to an inherent limitation when using a linguistic grammar or rule base for training connectionist networks. Grammar rules are always open to amendment and such alterations may unintentionally change the function of other rules in the grammar. If a connectionist architecture is based directly on the grammar then the architecture must be amended accordingly. This implies a serious limitation with connectionist architectures that use localist representations for specific grammars such as the localist output scheme used for the connectionist recogniser defined in Section 3.3.4 and for purely localist parsers such as Henderson's NNEP parser [14] and Stevenson's CAPERS parser [111].

3.4. Discussion

This chapter has presented some preliminary investigations into using simple context-free grammars to train feed-forward MLPs to act as grammar recognisers within a hybrid shift-reduce parsing framework. The architectures integrate symbolic and connectionist structures to act as push-down automata to parse arbitrary length arithmetic expressions and simple natural language sentences.

Feed-forward MLP networks trained with Back-propagation proved adequate for the identification of valid arithmetic expressions. The arithmetic expressions considered constituted a small context-free language domain and one which made it easy to construct training data based on the use of a sliding input window of four symbols. There was no ambiguity that could not be resolved by processing four input symbols. Providing that all reducible expressions were provided to the MLP network, it was able to learn to reject invalid expressions whilst accepting the 17 valid expressions, with only six hidden units.

Using a similar architecture, an MLP network was trained to identify valid phrases for a simple natural language grammar. The natural language grammar proved more complex than that for arithmetic expressions due to the requirement for position variant phrase recognition. The task of the MLP network was therefore to identify a valid phrase, the phrase type *and* its position within the temporal input window. The network required 15 hidden units to learn 99.5% of the training data and provided impressive generalisation rates of 92%. It successfully rejected the large numbers of ungrammatical phrases, however this was at the expensive of not fully learning the training data. Although a reasonable generalisation rate of 92% and 86% were obtained for the recogniser and disambiguation module respectively, further experiments to find a significantly better fit to the input data were unsuccessful.

Significant limitations were encountered when attempting to use a single feed-forward MLP network to identify and validate phrases generated from a large context-free grammar. Such architectures are tightly coupled with the grammar and amendments to the grammar results in corresponding changes to the architecture and the training and test data. Non-recurrent MLP architectures are limited to using fixed length temporal input windows and therefore a maximum phrase length must be predetermined. Also, a large number of examples of fixed length input, from each pattern class, are required for an MLP network to *possibly* learn the grammar (i.e., either by inclusion of all the relevant patterns in the training set or by generalisation).

The deficiencies of strictly feed-forward MLPs for language processing have recently been reported by other researchers. Lawrence[61] found feed-forward MLPs to be impractical for fully acquiring the small context-free grammar used by Elman[74]. He stated that although a large temporal input window allowed successful training, the MLP network was not forced to model a grammar. However, he proved that simple recurrent architectures with Back-propagation learning (i.e., the SRN network[19]) trained with the same grammar are able to acquire its linguistic rules.

It could also be questioned as to whether presenting a connectionist architecture directly with artificial grammar rules or associating an architecture with a specific grammar is beneficial or holds any advantage over classical symbolic parsers. A more natural use of connectionist networks would be to apply a connectionist architecture to the task of implicitly deducing linguistic constraints directly from realistic input sentences (i.e., a corpus of naturally occurring text). The emergence of pre-tagged and pre-parsed natural language corpora enables the development of more realistic approaches to the automatic syntactic analysis of unconstrained text and the automatic acquisition of grammatical structure from annotated text corpora via learning.

Subsequent chapters present solutions that overcome the limitations of the parsing architectures presented in this chapter. This is accomplished by decomposing the parsing problem into simpler tasks and using alternative connectionist architectures.

Chapter 4

The Corpus-Based Parsing Model

4.1. Introduction

This chapter provides a high-level description of the hybrid syntactic parser that integrates modular connectionist architectures with symbolic structures to automatically learn syntactic structure from a large text corpus. The modular architecture developed is motivated by the results of the preliminary investigations conducted in Chapter 3 with context-free grammars and feed-forward MLP architectures. Section 4.2 presents the Lancaster Parsed Corpus (LPC) as the fundamental basis of the grammatical framework for the parser and explains its composition, adequacy and limitations. Section 4.3 presents an overview of the hybrid parsing model developed to parse sentences sampled from the LPC, explaining the modular architecture, input representations of syntactic tags and phrase labels, the parsing algorithm and a parsing example. Section 4.4 presents the composition of the training and test samples and the constraints placed upon them. Finally, Section 4.5 concludes with a brief summary of the chapter.

4.2. The Lancaster Parsed Corpus

Rather than using a set of strict grammar rules as the fundamental basis for training data, a pre-parsed corpus of English text will be used. This will enable the connectionist networks used to learn less constraining grammars implicitly from sentence examples. The grammatical structures learnt would also reflect those that naturally occur rather than hypothetical hand-crafted structures.

The most accessible corpus which has been successfully annotated for syntactic structure, and the most suitable for forming the underlying grammatical framework for a system attempting to learn syntactic structure by example, is the Lancaster Parsed Corpus (LPC)[147] and it will therefore be used to form training and test data. The LPC is a corpus of British English sentences selected from printed publications of the year 1961 and is a subset of the word-tagged Lancaster-Oslo/Bergen (LOB) Corpus [146]. The LPC was formed as a result of Garside and Leech's original attempt to parse the entire LOB corpus with a HMM-based probabilistic parser [142]. Although this proved too ambitious, a sample of 145 text extracts was successfully processed from a possible 500 extracts from the LOB corpus. Each word of the LPC has been tagged by the Lancaster research team's CLAWS [148] word tagger. Garside and Leech's parser processed each words syntactic tag and not the words themselves during parsing. The syntactic word tags used by the modular parser proposed in this chapter will therefore be those produced by CLAWS, and the constituent tags used will be those defined by Garside et al [142].

The following sections discuss the corpus size, composition and its limitations.

4.2.1. Corpus Size and Composition

The sentences contained in the LPC are grouped in terms of their text category (see table 4.1). The categories are from A to R and are taken from the original LOB Corpus, which in turn were based on those of the Brown Corpus of American written English, for which the LOB Corpus is a British matching equivalent. Each category refers to the source from which the sentence was extracted, text category A refers to those sentences extracted from Press Reportage articles. Table 4.1 shows the number of sentences and words in each category in the LPC. The word-tagged LOB Corpus, which has provided the input to the LPC, consists of 1,013,737 tagged running words, divided into 500 samples of approximately 2,000 words each. The LPC is therefore a sub-corpus of it, consisting of 13.29% of the tagged LOB Corpus.

Text Category	**Description**	**No. of Sentences**	**No. of Words**
A	Press: reportage	728	8,188
B	Press: editorial	781	8,832
C	Press: reviews	594	7,145
D	Reviews	721	8,803
E	Skills, trades and hobbies	563	8,347
F	Popular lore	584	7,526
G	Belles lettres, biography, essays	406	5,987
H	Miscellaneous (government documents, etc)	441	5,798
J	Learned & scientific writings	450	7,537
K	General fiction	1,119	11,279
L	Mystery & detective fiction	1,188	13,010
M	Science fiction	769	7,525
N	Adventure & western fiction	1,344	12,919
P	Romance & love story	1,369	13,826
R	Humour	720	8,018
	Total	11,827	134,740

Table 4.1 : Contents of the LPC.

The LPC contains sentences from the first 10 text samples of each text category of the LOB Corpus, except for Categories M and R, which contained only 6 and 9 text samples respectively in the original corpus. However, since the LPC samples are restricted to the first ten (or less) samples in each category, they are more limited in genre/domain than the LOB corpus.

However, the Lancaster Parsed Corpus is a tree-bank broadly representative of the syntax of written English across a variety of styles and text types and will be used for training and testing the hybrid parser presented in this chapter. It is the intention that the parser will learn and be able to generalise from the grammatical rules and structures present in the LPC to form a wide coverage of the English language.

4.2.2. Syntactic Notation

4.2.2.1. Word Tags

Each word in the LPC is followed by an underscore character and a sequence of symbols (normally capital letters) which represents a word tag, i.e., a label giving the grammatical class of a word. For example, in the example sentence (S4.1), the word tag VBZ means 'third person singular verb', CC means 'coordinating conjunction such as AND, OR, BUT etc'. Appendix D provides a full list of word tags.

(S4.1) Bob_**NP** knows_**VBZ** but_**CC** he_**PP3A** would_**MD** nt_**XNOT** talk_**VB** either_**RB** **._.**

In the LPC punctuation marks and brackets (e.g., `.' `?' `!' `:' `;' `,' `-' `(' `)') are treated as words for the purposes of word tagging. Although punctuation marks are treated as parts of a sentence in their own right, they will be omitted from the parser's training and test data in order to reduce the complexity. However, the full stop will be used to denote the beginning and end of a sentence.

The LPC also uses idiom tags to refer to words that behave syntactically in an idiomatic way e.g., 'per cent','far from' and 'out of'. Such words are tagged using the syntactic tag appropriate for the sequence as a whole together with two digits, the first of which indicates the total number of elements in the idiom, the second standing for the number of the position of that particular element within the sequence. For example, consider the following :

out_**IN21** of_**IN22**

where IN refers to a preposition. This would be grouped as one unit using the terminal symbol IN appended with the symbol =. However, for simplicity the parser presented in this chapter will treat IN= as a non-terminal symbol and will ignore the numerical code on the word tag. Table 4.3 shows the total number of syntactic categories available for each syntactic grouping i.e., nouns, verbs, prepositions, conjunctions and punctuation (including the null character, ^). It can be seen that there are 83 word tags for words belonging to a noun phrase group; 33 word tags associated with words belonging to a verb phrase group etc.

4.2.2.2. Constituent Tags

The constituent tags are the non-terminal symbols that label groups of word tags and other constituent tags in order to generalise about grammatical structure. Constituent tags are denoted by capital letters that indicate the major class of constituent that the tag labels with additional lower-case letters to indicate a subclassification.

There are five main classes of constituent tags used in the LPC : sentence tags, finite clause tags, non-finite and verbless clause tags, major phrase tags and minor phrase tags. The full set are shown in appendix E.

Word Tag's Syntactic Group	Number of Word Tags
Noun Phrase	83
Verb Phrase	33
Preposition Phrase	4
Conjunction (e.g., the tag CC for the word AND)	3
Punctuation	20
Total	143

Table 4.3 : The types and numbers of syntactic word tags used in the LPC.

There are three types of sentence tags : S, Sq and Si tags. The S tag simply refers to an entire sentence and is commonly considered to be the root of the parse tree. Sq refers to a piece of direct quotation and is normally an independent piece of language that occurs in fictional dialogue enclosed in quotation marks. However, due to punctuation being omitted from the training and test data for the parser presented in this chapter, Sq will always be recognised as S or a coordinated S depending upon the context. Si means an interpolated sentence in that it is a grammatically independent piece of language that is inserted (normally in enclosed brackets) in another sentence, but it is not grammatically part of it. Finite clause tags represent a group of words and constituents that contain a finite verb. A finite verb normally has a subject and a tense. Non-finite clause tags represent groups of words and constituents that contain no verb or a non-finite verb. A non-finite verb usually has no tense, no subject or a subject the verb does not agree with. Major phrase tags represent groups of words and constituents that belong to major phrase types such as noun phrases, verb phrases, prepositional phrases, adjective phrases and adverbial phrases. Minor phrase tags represent some aspect of a phrase like a determiner functioning as the head of some phrase within a noun phrase or groups of words that act as a genitive in a noun phrase like '*my fiancee's*' in '*my fiancee's favourite dress*'.

4.2.2.3. Constituent Coordination

The symbols &,+,- and / can be appended to constituent tags to denote constituent coordination. The process of coordination joins two short clauses with a conjunction like *and*, *but*, *or*, *nor*, or *neither*. These are termed coordinating conjunctions and appear before the clause they will coordinate. A constituent tag appended with & denotes a compound phrase i.e., a phrase that contains another phrase of the same type embedded within it. For example, N& denotes a compound noun phrase, in which two or more noun phrases are joined by a coordinating conjunction. The + symbol represents a coordinated constituent that appears after a conjunction e.g., N+ represents a coordinated noun phrase. As figure 4.1 illustrates, the second or subsequent coordinated noun phrase is treated as subordinates of the first in the LPC.

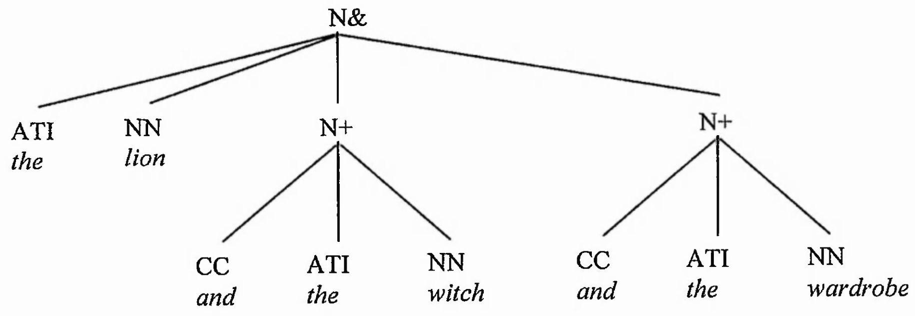


Figure 4.1 : Subordinate constituent structures represented in the LPC.

When a clause is not preceded by a coordinating conjunction, i.e., it may have been preceded by a comma, then the - symbol is used to denote a coordinated constituent without a coordinating conjunction.

When the two or more coordinated elements do not belong to the same class, Garside et al [142] have used similar constructs to GPSG in that '*slash tags*' are used i.e., a composite tag in which the classes of the items concerned are separated by a slash (/). The label to the left of the slash represents the main constituent group to which the input symbols belong. The symbol to the right of the slash represents the constituent tag missing from the construction.

For example, figure 4.2 illustrates that the adjective phrase is missing a *Po&* for the coordinated preposition phrase, *Po+*.

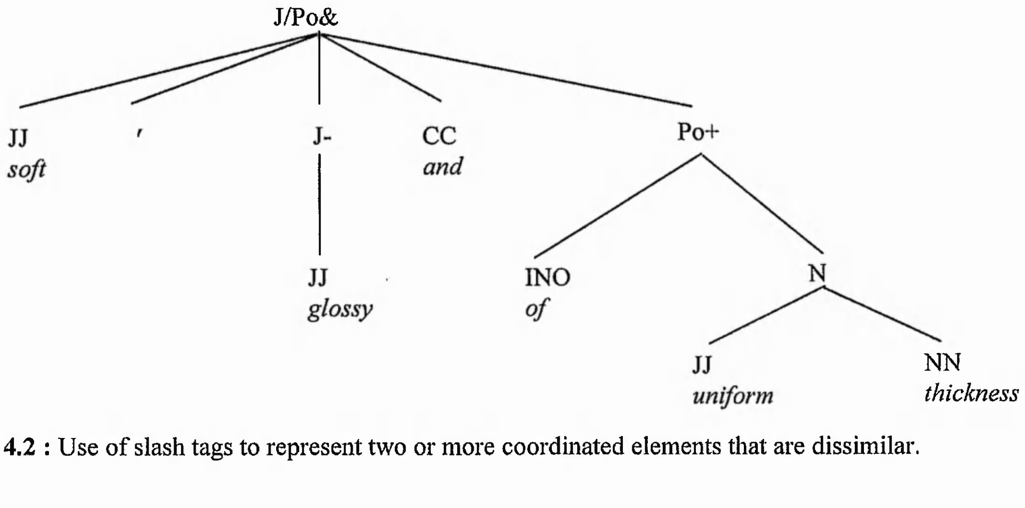


Figure 4.2 : Use of slash tags to represent two or more coordinated elements that are dissimilar.

For simplicity, each slash tag in the LPC will be treated as a single non-terminal symbol and although this will limit the ability of the parser to coordinate constituent structure it is intended that the parser will be able to coordinate similar structures as those found in its training sample.

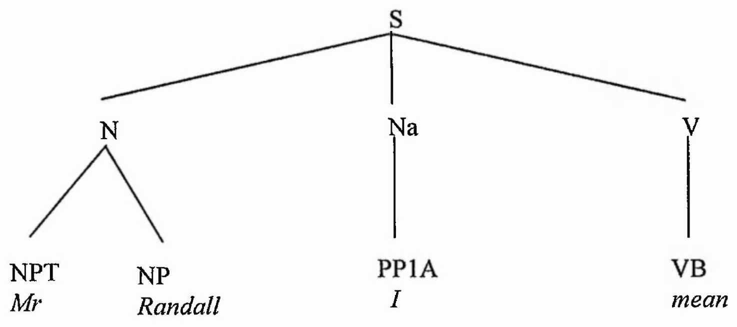
If coordination takes place on the word level only, (e.g., where two or more words inside the same constituent are linked by the conjunction *and*), the coordination is indicated in the same way as constituent coordination. Garside et al [147] state the reason for this is that the set of coordinated words behave, with regard to the rest of the sentence, like a single word. Word level coordinations of words that belong to different classes are treated by slash tags in the same way as described previously.

4.2.2.4. Representation of Constituent Structure

Labelled bracketing encodes the same information as the standard parse tree but represents it in a linear fashion. In the LPC, *[* represents the beginning of a constituent, and *]* represents the end or completion of a constituent. The symbols alongside these brackets are labels for constituents. For example, *[V* denotes the beginning of a verb phrase and *V]* denotes the end of a verb phrase.

Although the parser's output represents constituent structure with labelled bracketing, constituent structure will largely be represented in tree form throughout the remainder of this thesis for clarity and simplicity.

a) Parse tree



b) Labelled Bracketing

[S [N Mr_NPT Randall_NP N] [Na I_PP1A Na] [V mean_VB V] S]

Figure 4.3 : (a) Parse tree for the sentence, *Mr Randall I mean.* (b) The corresponding labelled bracketing format of the sentence structure.

4.2.3. Corpus Limitations

Although the LPC is a relatively large parsed corpus with a variety of text types it has limitations. For example, the sentences included in the LPC are considerably shorter, on average, than the average sentence length for the LOB Corpus as a whole (the average for LPC is 11.39 words per sentence, as compared with an average of 19 words per sentence for the LOB corpus). The reason for this is that the probabilistic parser used by Garside et al [147] to automatically parse 145 text extracts from the LOB corpus failed to parse most sentences over 20-25 words in length. These unparsed sentences were therefore omitted from the LPC. For the remaining sentences, which had been parsed, Garside et al took the parse that the parser had assigned the highest probability as the correct parse. The resulting parses were then subject to manual post-editing where human linguistic experts manually rectified parse errors.

Another limitation is that during the post-editing phase, some ambiguities were encountered where there was uncertainty about the most likely interpretation of a sentence in its context. Although Garside et al state that such problems only arose infrequently, little information is provided about the nature of the ambiguities. It is therefore possible that conflicting training and test patterns, extracted from the corpus, may be due to these ambiguities.

4.3. A Connectionist Corpus-Based Natural Language Parser

4.3.1. Description of the Model

The results presented in Chapter 3 illustrated that the use of a single feed-forward MLP with a temporal input window is adequate for phrase structure recognition in restricted domains (Tepper et al [153,154]). However, if the grammar used is large and complex, the size of the connectionist network and the resulting training sets required will become large and intractable. Such problems have led to connectionists adopting a modular approach to parsing [62,126,127,139,162,163,164,166,167,168,169,170,171], using a mixture of feed-forward and recurrent connectionist networks, each with its own dedicated task. Although high levels of generalisation and processing capabilities were reported over singular connectionist networks, small context-free grammars were still used and the combinations of connectionist architectures proved limited. There is also no reliable or generally accepted method of combining connectionist architectures to perform high-level tasks such as parsing.

The modular parsing approach adopted in this thesis is aimed at processing a realistic subset of natural language and to learn general grammatical constraints that are prevalent in naturally occurring language used by native speakers. It is also the intention that the parser will be able to parse sentences of arbitrary length although a degradation in performance will be expected as very long phrases are processed. Although it is not the purpose of the parsing model to strictly model human syntactic processing, if the behaviour of the system makes similar structural preferences and exhibits similar performance errors to humans then this is both beneficial and encouraging.

In order to identify the modules of the system it is necessary to decompose the parsing problem into two or more stages. The fundamental processes involved in syntactic parsing are identifying phrase boundaries and recognising phrase types. The two basic modules are therefore phrase segmentation and phrase recognition. This would not be a natural distinction in symbolic processing as it is unclear as to whether it would be feasible to specify general symbolic rules for phrase delimitation without complex construction-specific heuristics. It is, however, a natural distinction in pattern recognition and connectionist network terms. Finding phrase boundaries is analogous to finding edges or contours of objects in computer vision, and recognising phrases is analogous to object recognition.

The phrase segmentation process can be further decomposed into two sub-processes : a right-to-left phrase delimiting process and a left-to-right phrase delimiting process. The task of the right-to-left delimiter (RLD) is to recognise the beginning of a syntactic phrase, and the left-to-right delimiter (LRD) to recognise the end of a syntactic phrase. The number of input symbols processed by either delimiter before the beginning or end of a phrase is encountered is variable and not known *a prior*, therefore it is essential that the connectionist network assigned to either task is able to sequentially process linguistic input. Locally recurrent MLP architectures have been developed to implement sequential processing within MLP networks. These architectures are locally recurrent in that the output activations of one layer of units are fed back as input into a previous layer (and succeeding layer) of units rather than being fed to every unit (including itself) within the network to form a fully recurrent architecture. Locally recurrent MLP architectures that have recurrent connections feeding from the output units back to the input units (such as Jordan's Sequential Machine [28] and Back-propagation-in-time [156, 160]) allow previous outputs to constrain the processing of current inputs. When processing linguistic information, however, contextual information is essential and thus the ability to process the current input with respect to the previous input (rather than output) could be considered to be more useful. Recurrent MLP networks that have recurrent connections feeding from the hidden units back to the input units, such as Elman's Simple Recurrent Network (SRN) [19], have proved very successful at temporal processing of linguistic input and enabling MLP networks to establish temporal relationships over sequential input symbols. The recurrence of hidden unit activations is also more powerful than recurrence of output unit activations for cognitive tasks [132], since they represent an internal reduced description of the input representation. The network is therefore able to build its own representation of time. However, SRNs possess severe memory limitations due to the use of the untrained 'don't-care' cycles. As there is no learning on the recurrent connections and error correction is not performed until a target output is provided, errors in processing cannot be corrected until the target output is encountered. This is problematic when the target output vector is not encountered until several time steps of unconstrained processing (i.e., don't-care cycles) and consequently some earlier input information is lost.

To overcome this problem by ensuring targets are available at every processing stage and that previous inputs are not forgotten, Ghahramani and Allen [132] have developed the Temporal Auto-associative SRN (TASRN). This consists of auto-associative learning of the current input and hidden state during each processing stage. The TASRN is a suitable architecture for both the RLD and LRD delimiters providing sequential processing of linguistic input and an increased memory limit. The RLD, LRD and TASRN networks are fully detailed in Chapter 5.

A feed-forward MLP trained with Back-propagation is adequate for performing phrase recognition, however as illustrated in Chapter 3, complexities are encountered when expecting a single MLP to recognise the position of the phrase and to reject invalid phrases. This has motivated the decision to employ a strictly feed-forward MLP network trained with Back-propagation to act as a simple phrase structure recogniser whose output activations are subject to a 'nearest match' computation to find the closest valid phrase for an input phrase. This removes the problem of phrase validation. The position of the phrase within the MLP network's input window will also be fixed to eliminate the complexities associated with phrase variant recognition. Context from the left-hand side *and* right-hand side of the extracted phrase will be presented to the MLP to aid recognition.

Three core symbolic structures are used. A symbolic linked list is used to store tag information, a symbolic stack to store parse state information (i.e., the *Parse-stack*) and the resulting parse tree, and finally a Last-In-First-Out stack structure is used as an *Input-stack* to store the current input state. These symbolic components of the parser reflect the hybrid nature of the parser with subsymbolic processing within the connectionist networks and communication at a symbolic level between them.

The *Scheduler* controls the interaction between the symbolic and connectionist components and the flow of information. The Scheduler is the supervisory code that implements the deterministic shift-reduce parsing strategy. The parser implements a shift-reduce algorithm similar to that defined by Shieber [105,106]. The parser will process each constituent sequentially and 'shift' across the input i.e., process the next word, until a group of words correspond to a valid syntactic phrase. A 'reduction' can then take place, i.e., the appropriate words in the Input-stack are replaced by one symbol denoting the syntactic phrase to which the group of words belong. Each time the state of the Input-stack changes due to a reduction, the parser will reset its read position back to its original starting position.

The parser is strictly deterministic in that input symbols are continually read until the correct syntactic phrase can be selected for a particular group of words. Once the parser makes a commitment to a particular reduction i.e., makes a structural preference, its decision is final and cannot be modified at a later processing stage. The parser's delimiting modules will have learnt to identify phrase boundaries, and once the delimiters have been able to delimit a phrase the parser will be allowed to look-back and look-ahead of the phrase before making a final decision as to the syntactic phrase. It is the use of look-back and look-ahead symbols that enables the parser to analyse context to the left *and* right-hand side of the phrase.

In contrast to the traditional left-to-right processing approach, the parser will process each sentence from right-to-left. The advantage of parsing in a right-associative manner is that structural ambiguity requiring no semantic information, will be resolved immediately and a common structural preference is made when processing structural ambiguity that requires semantic processing. The common structural preferences made by the parser will be those reflected in the training samples extracted from the LPC. Since the training sentences are naturally occurring English text extracts from varying contexts it is hoped that the structural preferences made by the parser are similar to those made by people. An advantage here is that no construction specific heuristics is built into the parser to guide its decisions, therefore it will be interesting to contrast the structural preferences made by the parser with those made according to Frazier's Minimal Attachment and Late Closure principles [114,115,116].

4.3.2. Word and Constituent Tag Representations

Previous distributed parsing systems have placed little emphasis on encoding input data in a way that will aid learning i.e., an input representation that makes clear distinctions in input space between the different types of allowable input. Representations that use one input unit per symbol are similar to localist representations and thus inhibit information sharing amongst units. Each addition of a new input symbol therefore increases the dimensionality of the input layer by one unit. Another approach has been to use fixed-width bit patterns to represent all the input symbols by associating a binary value to each symbol. In this case, the number of bits used is the minimum required to encode all the symbols. For example, if there are fifteen possible input symbols then four bits are required to encode all of them. This is more efficient than associating each input unit with an input symbol. However, symbols that are presented to the connectionist network as a random or linear bit pattern of fixed length can impede learning as dissimilar input patterns may be very close in input space but belong to a totally separate output category. This increases complexity and further hidden units would be required to learn the problem. An increasingly popular method adopted by some connectionists [23,126,138,139] has been to use auto-association to form hidden unit representations or 'reduced descriptions' of the input data. The resulting hidden unit vectors represent contextual features of the input and are used as input to their parsing network. The limitations of this technique are that the reduced description may not be an efficient reduction in terms of units and the representation of all input symbols will change if new input symbols are required. This results in further training processes as the connectionist networks dependent upon the input representations will also have to be retrained.

The input representation developed in this section will aim to aid learning via simply segmenting the input space into separate regions that correspond to different word and constituent tag types. The Euclidean distance function is used to measure the distance between input bit vectors. This takes a middle path between the extremes of 1 input unit per symbol and minimal encoding.

Each sub-section has an associated signalling bit that is only active when an input symbol belongs to that sub-section. The first bit of the sub-section has been designated for this purpose and the remaining bits of the sub-section are used to encode the input symbol. For example, as there are 33 verb phrase word tags, 6 bits are needed to represent them, plus 1 bit for signalling that it is a VP tag giving a total of 7 bits.

Figure 4.4 shows the input representation for a symbol as consisting of 61 bits.

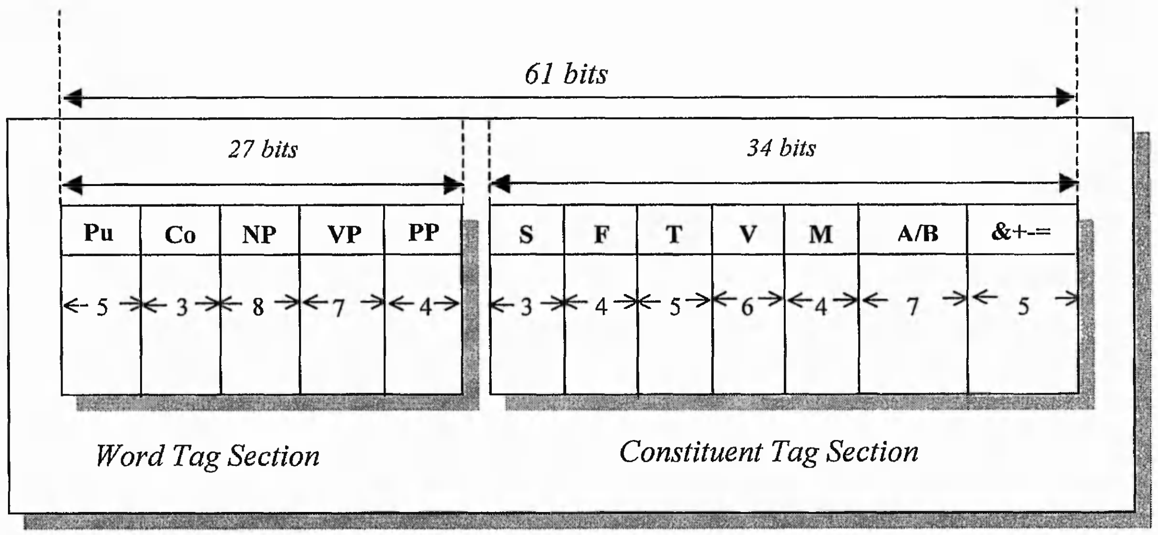


Figure 4.4 : Word tag and constituent tag input representation.

The representation of any two symbols in different syntactic groups will be orthogonal to each other. Two symbols belonging to the same group will not be orthogonal. If the connectionist network needs to make a 'decision' based upon group membership then there is one input unit (the appropriate signalling bit) on which that decision can be based. This means that it is easier for the network to learn rules of that type.

To represent coordinating or coordinated constituent tags, the constituent tag is represented in the constituent tag segment and the appropriate coordination symbol is also active. For example, the compound preposition phrase, *P&*, is represented as follows :

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0

It can be seen that *P* is represented as *1 0 1 1 0 0* in the major phrase sub-section, *V*, of the constituent tag segment of the bit pattern. The compound phrase indicator, *&*, is represented by *1 1 0 0* in the coordination sub-section, *&+-=*. For *P* as opposed to *P&* all units in the coordination sub-section would be inactive.

The slash tag sub-section of the constituent tag segment of the representation is used to encode all slash tags present in the LPC. This constrains the types of coordinated phrases that can be represented. However, this is justified as attempting to construct a representation that will allow arbitrary coordination of dissimilar phrases would result in combinatorial explosion and impractical training sets. Also, the slash tags that do occur in the LPC are those that are prevalent in naturally occurring English texts and it is therefore envisaged that a realistic coverage of such coordinated phrases is obtained from the training or test data.

The representation defined clusters similar inputs together in input space. For example, the Euclidean distance between the determiner word tag, *ATI* and the singular noun word tag, *NN* is 1.41421. If both input vectors were equal then the Euclidean distance would be 0 therefore this indicates that *ATI* and *NN* are close in input space. This is an accurate measure as both word tags belong to the same NP sub-section. However, if a word tag belongs to a different sub-section then the Euclidean distance will increase i.e., the Euclidean distance between the word tag for a common noun, *NN*, and the word tag for the verb *have, HV*, is 2.64575. The distance between word tags and constituent tags would be greater e.g., the Euclidean distance between the word tag for a singular noun, *NN*, and the constituent tag for a noun phrase, *N* is 2.82843.

4.3.3. The Parsing Architecture and Algorithm

As described in Section 4.3.1 the two fundamental processing phases of the parser are *phrase segmentation* and *phrase recognition.* A schematic of the architecture developed to perform these tasks is shown in figure 4.5. The system receives the tagged input sentence as a whole on its Input-stack and will process each word tag sequentially starting with the right-most symbol and then 'shift-left' across the input.

The parsing architecture consists of three connectionist components, three core symbolic components and a scheduler process to control the interaction between the connectionist and symbolic modules. The connectionist components form the phrase segmenting and recognition modules i.e., the RLD, LRD and Recogniser networks. The RLD and LRD networks are both TASRN recurrent networks, and the Recogniser is a feed-forward MLP as mentioned previously. The core symbolic components of the system are the tag database, Parse-stack and the Input-stack. The tag database is a linked list of symbolic structures that store each word tag and constituent tag symbol with its associated bit-representation. The Parse-stack stores the intermediate and final parse states. The Input-stack is a Last-In-First-Out (LIFO) stack structure that stores the input symbols which may consist of word tags or constituent tags.

The Scheduler is the supervisory process for parsing. The Scheduler will first remove any punctuation and then delimit the input sentence with full stops to act as begin and end markers. All of the input symbols are then pushed onto the Input-stack from left-to-right.

The maximum number of allowable look-back and look-ahead symbols must be pre-determined before parsing. The number of look-back symbols (x) used by the RLD network is fixed at 4 symbols, and the number of look-ahead symbols (y) used by the LRD network is fixed at 2 symbols. Also, the LRD network will only use 2 of the available look-back symbols. The justification for this look-back and look-ahead configuration is given in Chapter 5. The maximum number of symbols within a phrase (reduction) must also be predetermined. The maximum phrase length, z, is established simply by extracting every possible phrase from the LPC and setting z to the length of the longest phrase. The value of z for the entire LPC is 10.

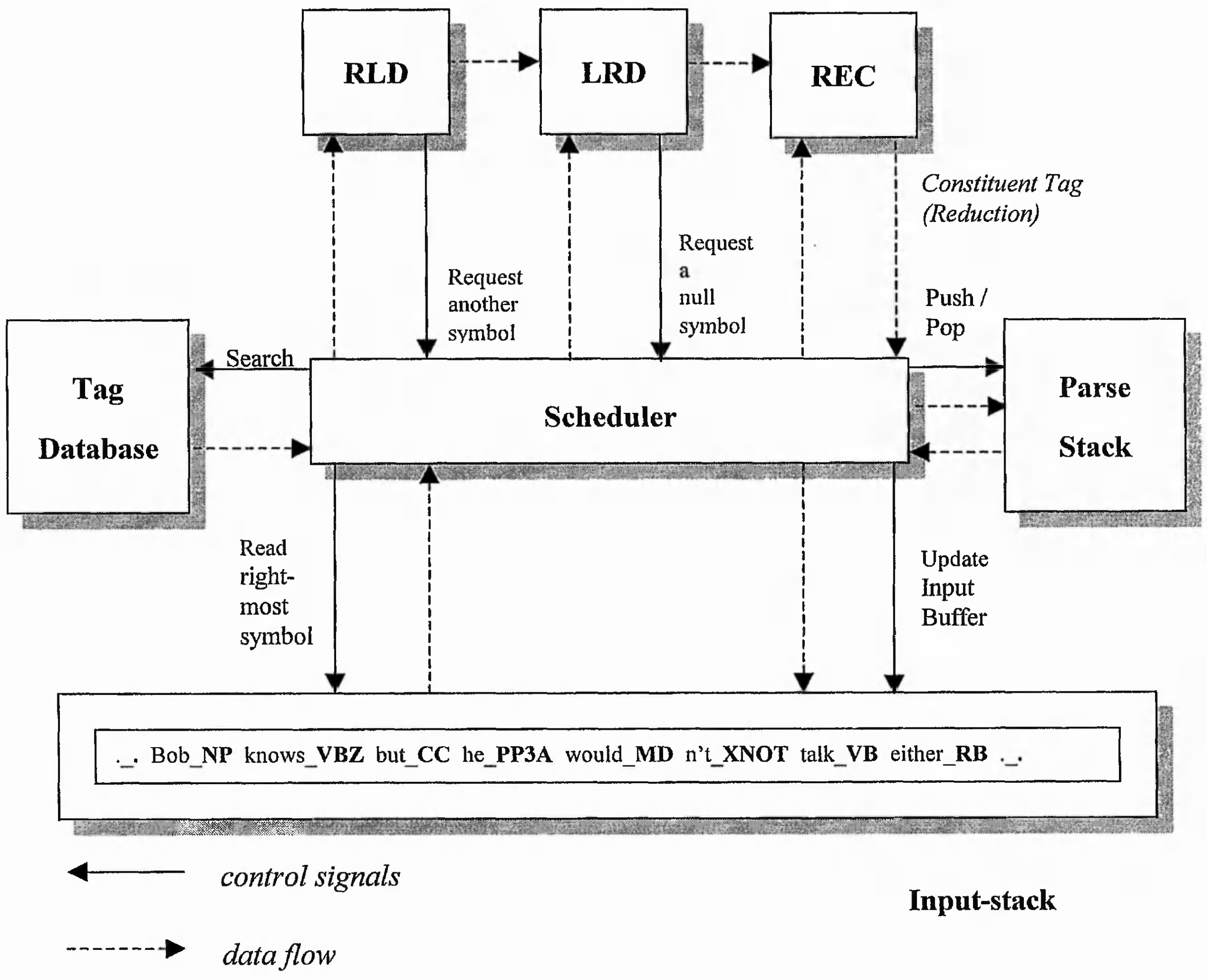


Figure 4.5 : A schematic of the hybrid parsing architecture. Temporary stacks used to hold data passing between the RLD and LRD modules, and between the LRD and REC modules have been omitted for clarity.

If there are not enough available input symbols to present the maximum number of look-back, look-ahead or phrase symbols to the appropriate connectionist network, then null symbols are used to 'pad' out the look-back, look-ahead or phrase to the predetermined length. A null symbol, ^, is simply a bit pattern of 61 zeros.

Before parsing commences, the context units of the RLD and LRD networks are initialised to 0.5.

The symbol at the top of the Input-stack is popped off and the tag database is searched for the bit representation of the symbol. If an entry for the symbol is not found in the tag database then the parse is aborted on the basis that an invalid input symbol has been encountered. If an entry has been found then the bit representation of the input symbol is passed to the input layer of the RLD network[17]. The RLD network then performs a forward-pass computation. The output response of the RLD network is a single floating point value between 0.0 and 1.0 representing the probability of the current input symbol being the beginning of a phrase. If the RLD output response exceeds a pre-determined threshold, ***t***, then this indicates that the beginning of a syntactic phrase has been encountered together with four look-back symbols. A typical threshold value for ***t*** is 0.7. If the RLD output response is below the threshold ***t*** the current input symbol is pushed onto a temporary Last-in-First-Out (LIFO) stack called the *LR-stack.* Another input symbol is then popped off the Input-stack and processed in the same way as mentioned. This is repeated until the RLD output response exceeds the threshold. If after all of the input symbols have been popped off the Input-stack and processed by the RLD network, the output response fails to exceed the threshold then the RLD network will be passed up to x-1 null symbols for the unavailable look-back (as the last input symbol will always be the 'beginning of sentence' marker it will act as a look-back symbol). If after all the null symbols have been processed by the RLD network and it's output response is still below ***t*** then the input sentence is rejected on the basis that the beginning of a valid syntactic phrase cannot be found. However, if at any stage the output response of the RLD network exceeds ***t*** then this signals that the beginning of a syntactic phrase has been found together with x look-back symbols. The context units of the RLD network are then reinitialised to 0.5.

After the beginning of a phrase has been found, the LR-stack contains all the input symbols processed by the RLD network including the look-back symbols.

The number of look-backs required for the LRD network is 2 so the two unwanted look-back symbols are popped off the LR-stack and pushed onto another temporary stack called the REC-stack. As well as the two unwanted look-back symbols, the REC-stack will eventually contain input symbols processed by the LRD network.

The symbols contained on the LR-stack can now be processed sequentially by the LRD network to find the end of the phrase. The last symbol pushed onto the LR-stack is popped off. This is passed to the input layer of the LRD network. The LRD network then performs a forward-pass computation. The output response of the LRD network indicates the end of a phrase in the same way as the RLD output indicates the beginning of a phrase. When the output is below the threshold ***t*** then the current input symbol is pushed onto the REC-stack and another symbol is popped off the LR-stack. Again, null symbols are used to pad the input when there is not enough input symbols for y look-ahead symbols. If the output response of the LRD network exceeds ***t*** then this signals that the end of a syntactic phrase has been found together with 2 look-back symbols and 2 look-ahead symbols. The context units of the LRD network are then reinitialised to 0.5. However, if the LRD network's output response fails to exceed the threshold after processing all symbols on the LR-stack and any null symbols used, then the input sentence is rejected on the basis that the end of a valid syntactic phrase cannot be found. The input symbols that remain unprocessed on the LR-stack will eventually be pushed back onto the Input-stack.

[17] From hereon, rather than explicitly stating that the word tag or constituent tag is converted to its bit representation before being presented to a connectionist network it will be assumed that Scheduler has performed this task.

After the end of a phrase has been found, the REC-stack contains all four look-back symbols, the phrase symbols and only one of the two look-ahead symbols (see Chapter 6). The remaining look-ahead symbol is pushed back onto the LR-stack. If the number of phrase symbols is less than z then the phrase on the REC-stack is padded out with null symbols. The look-ahead symbol must therefore be popped off the REC-stack so that the relevant number of null symbols can be added before the look-ahead symbol is pushed back onto the REC-stack. The contents of the REC-stack will be presented to the Recogniser network.

The Recogniser network is a strictly feed-forward MLP architecture and has a fixed input length of $x + z + 1$ input symbols (at 61 bits per symbol). At this stage in processing the REC-stack consists of the look-back symbols with null padding, the phrase symbols with null padding and the single look-ahead symbol. As the size of the look-back, phrase and look-ahead symbols are of fixed length and always presented to the Recogniser in the same order, the task of recognising the phrase is made simpler in that the phrase will always be in the same position within the input. The Recogniser network and the constraints placed upon it is discussed in further detail in Chapter 6.

To recognise the extracted phrase, the contents of the REC-stack must be converted into its bit representation and presented to the input layer of the Recogniser network. Each symbol, including the null symbols, is popped off the REC-stack and the tag database is searched for its associated bit representation. After all the symbols have been popped off the REC-stack and presented to the Recogniser network's input layer, the Recogniser network will perform a forward-pass computation. The output of the Recogniser network is a vector of 61 floating point activation levels. Each activation is then converted into either a 1 or 0 depending on whether it exceeds 0.5. If the activation level is above 0.5 then it is set to 1, else if it is equal to or below 0.5 it is set to 0. The resulting bit vector is the representation of the constituent tag that the Recogniser network reduces the current input phrase to. However, rather than searching the tag database for a perfect match of the bit vector, a nearest match computation is performed to find the nearest matching constituent tag. The entire tag database is sequentially searched and the Euclidean distance is calculated between the output vector and the database tag vector. The constituent tag vector closest to the output vector is selected. The Recogniser's 'decision' is final and cannot be modified. The task of the Recogniser network is now complete in that it has produced a tag for a given group of input symbols. However, if the tag is a terminal symbol (i.e., a word tag) then the sentence is rejected on the basis that a valid constituent tag could not be recognised for the sequence of input symbols extracted by the RLD and LRD networks. If the tag is a constituent tag then it assumed to be valid and processing continues.

The Parse-stack and Input-stack need to be updated before further phrases can be extracted. For the Parse-stack to be updated relationships need to be established between previous parse states (i.e., constituent tags already stored on the Parse-stack) and the input symbols bound to the current constituent tag (current parse state). Each input symbol assigned to the constituent tag by the Recogniser network is checked to examine whether it is a word tag or constituent tag. If it is word tag then the next input symbol is checked. If it is a constituent tag then the Parse-stack is searched for the first occurrence of that particular constituent tag. If an occurrence is found then the status of that occurrence is examined. If the status is *active* then this indicates that it is already dominated by another constituent tag on the Parse-stack. A search is then performed for the next occurrence that is *inactive.* If a matching constituent tag is found on the Parse-stack and its status is *inactive* then it is attached to the current constituent tag i.e., an attachment operation is performed. If there is more than one matching constituent tag then the first one (the nearest) is selected. For example, if the Recogniser network has recognised an input sequence such as '*to_TO N*' as a preposition phrase, *P*, it is clear that the input symbol *N* is a constituent tag representing previously processed input symbols. Assume that *N* consists of the terminal symbols *the_ATI boat_NN* and it is stored on the Parse-stack as :

[N the_**ATI** boat_**NN N]** : *inactive*

The new entry, *P*, on the Parse-stack would be :

[P to_**TO [N** the_**ATI** boat_**NN N]** : *active* **P]** : *inactive*

where *N* is now attached to *P* and thus becomes *active* and as *P* is the latest entry on the Parse-stack, representing the current parse state, it is set to the default state of *inactive.*

The Input-stack must now be updated using the constituent tag produced by the recogniser and the contents of the LR-stack and the REC-stack. First, all the look-back symbols used by the Recogniser are pushed back onto the Input-stack (excluding null symbols). The constituent tag produced by the Recogniser network is then pushed onto the Input-stack. The look-ahead symbol used by the Recogniser network are then pushed onto the Input-stack. Finally, the unprocessed symbols remaining on the LR-stack are pushed back onto the Input-stack. This process constitutes the reduction of input symbols. The contents of the Input-stack is now ready to be processed again in the same order i.e., starting with the last symbol. The above procedure is repeated until all input symbols have been processed and cannot be processed further.

If all input symbols have been processed and the final constituent tag produced by the Recogniser network is *S* or *S&* (depending upon the context) which dominates the remaining parse states stored on the Parse-stack then a valid parse is reported. However, if all input symbols have been processed and the parser has failed to produce *S* or *S&* or if *S&* is produced and there are unprocessed input symbols remaining on the Input-stack then a failed parse is reported.

4.3.4. A Parsing Example

The previous section provided an overview of the parser's architecture and algorithm, and the symbolic and numeric processing that takes place in the resulting corpus-based parsing system. This section briefly provides a high-level description of how the model parses sentence S4.2 using the procedure described above.

(S4.2) ._**.** Bob_**NP** knows_**VBZ** but_**CC** he_**PP3A** would_**MD** nt_**XNOT** talk_**VB** either_**RB** ._**.**

The Scheduler will first remove any punctuation and then delimit the input sentence with 'full stops' to act as begin and end markers before pushing all the input symbols onto the Input-stack from left-to-right. The input to the parser is a list of word tags corresponding to the words of the sentence, so the actual input to the parser would be **. NP VBZ CC PP3A MD XNOT VB RB .** rather than '*bob knows but he would not talk either*'. However, the actual word rather than its word tag will be referred to in this section for clarity. The following input symbols are therefore currently on the Input-stack :

. Bob knows but he would nt talk either .

Table 4.4 shows the processing stages of the parse and details the contents of the Parse-stack at each stage.

	Input-stack	**Extracted Phrase**	**Reduction**	**Parse-stack Entries**
1	. Bob knows but he would n't talk either .	Either	R	[R either R]
2	. Bob knows but he would n't talk R .	would n't talk	V	[V would n't talk V] [R either R]
3	. Bob knows but he V R .	He	Na	[Na he Na] [V would n't talk V] [R either R]
4	. Bob knows but Na V R .	but Na V R	S+	[S+ but [Na he Na] [V would n't talk V] [R either R] S+]
5	. Bob knows S+ .	Knows	V	[V knows V] [S+ but [Na he Na] [V would n't talk V] [R either R] S+]
6	. Bob V S+ .	Bob	N	[N Bob N] [V knows V] [S+ but [Na he Na] [V would n't talk V] [R either R] S+]
7	. N V S+ .	N V S+	S&	[S& [N Bob N] [V knows V] [S+ but [Na he Na] [V would n't talk V] [R either R] S+] S&]

Table 4.4 : Summary of the processing stages for the example parse.

However, the contents of the temporary stacks (LR-stack and REC-stack) are omitted for clarity.

It can be seen from processing stage one that the RLD and LRD networks have extracted *either* as the first valid syntactic phrase. The phrase consists of only one symbol and the Recogniser network requires that the phrase is fixed at 10 symbols. An additional nine null symbols are therefore added to the phrase to pad it out to the 10 symbol limit. The input to the Recogniser is therefore :

he would nt talk ***<u>either ^ ^ ^ ^ ^ ^ ^ ^ ^</u>*** *.*

The actual phrase to be recognised is emphasised in bold and is underlined to illustrate the fact that to the left of the phrase are the look-back symbols (as extracted by the RLD network), and to the right of the phrase the end-of-sentence (EOS) symbol, **.**, is the look-ahead symbol. The Recogniser network performs a forward-pass computation and the corresponding Euclidean distance between the resulting output vector and all the constituent tags in the tag database selects *R* as the nearest match. *R* is attached to *either* and the following is pushed onto the Parse-stack :

[R either_**RB R]** : *inactive*

The Input-stack is then updated to reflect this 'reduction' by pushing back on the four look-back symbols, the constituent tag *R*, and the look-ahead symbol. The Input-stack now consists of the following symbols :

. *Bob knows but he would nt talk R* **.**

It can be seen from processing stage two in table 4.4 that the RLD and LRD networks have extracted *would nt talk* as the next valid syntactic phrase. An additional seven null symbols are therefore used to pad out the phrase. The REC-stack contains the following :

Bob knows but he ***<u>would nt talk ^ ^ ^ ^ ^ ^ ^</u>*** *R*

The Recogniser network performs a forward-pass computation and *V* is selected as the nearest matching constituent tag. The following is therefore pushed onto the Parse-stack :

[V would_**MD** nt_**XNOT** talk_**VB V]** : *inactive*

The Input-stack is updated by pushing back on the four look-back symbols, the constituent tag *V*, the look-ahead symbol, *R*, and the remaining unprocessed symbols on the LR-stack (in this case the EOS symbol). The Input-stack now consists of the following symbols :

. *Bob knows but he V R* **.**

The third phrase extracted by the RLD and LRD networks is *he*. This signals that symbols '*. Bob knows but*' are look-back symbols and '*V R*' are the look-ahead symbols. The Recogniser only requires one look-ahead symbol and the REC-stack therefore contains the following :

. Bob knows but ***<u>he ^ ^ ^ ^ ^ ^ ^ ^ ^</u>*** *V*

The contents of the REC-stack is presented to the Recogniser network. The Recogniser network performs a forward-pass computation and *Na* is selected as the nearest matching constituent tag. The following is therefore pushed onto the Parse-stack :

[Na he_**PP3A Na]** : *inactive*

The Input-stack is updated by pushing back on the four look-back symbols, the constituent tag *Na*, the look-ahead symbols, *V* and *R*, and the remaining unprocessed symbols on the LR-stack (in this case the EOS symbol). The reduction has now been performed and the Input-stack now consists of the following symbols :

. *bob knows but Na V R* **.**

The fourth phrase to be extracted from sentence S4.2 is *but Na V R*. As there were not enough available input symbols to the left of the phrase to form 4 look-backs, the RLD network would have used one null symbol to pad the number of look-back symbols to the desired length. This is reflected in the contents of the REC-stack :

. *Bob knows* ^ ***but Na V R*** ^ ^ ^ ^ ^ ^ .

The Scheduler organises the null padding to the right of the look-back symbols within the REC-stack. This is so that the look-back symbols are positioned further away from the phrase. The Recogniser network then performs a forward-pass computation in response to the contents of the REC-stack and *S+* is selected as the nearest matching constituent tag. As there are constituent tags within the phrase itself, the Scheduler matches each constituent tag against matching *inactive* constituent tags already on the Parse-stack. The following attachments are therefore made using this approach :

[S+ but_**CC [Na** he_**PP3A Na]** : *active* **[V** would_**MD** nt_**XNOT** talk_**VB V]** : *active* **[V** would_**MD** nt_**XNOT** talk_**VB V]** : *active* **[R** either_**RB R]** : *active* **S+]** : *inactive*

Note that the previously inactive parse states are now active as they have 'parent' constituent tags which dominate them within the structure. *S+* is inactive as it has yet to be assigned a 'parent' constituent tag.

The Input-stack is then updated by pushing back on the four look-back symbols, the constituent tag *S+*, and the EOS symbol. The Input-stack now consists of the following :

. *bob knows S+* **.**

As shown in table 4.4., the fifth phrase extracted by the RLD and LRD networks is *knows*. The nine null symbols are added to pad the phrase out and the left-most look-ahead symbol, '*S+*', is pushed back onto the REC-stack. The

REC-stack therefore contains :

*. Bob ^^ <u>**knows** ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^</u> S+*

The Recogniser network performs a forward-pass computation and *V* is selected as the nearest matching constituent tag. The following is pushed onto the Parse-stack :

[V knows_**VBZ V]** : *inactive*

The Input-stack is updated by pushing back on the look-back symbols, the constituent tag *V*, the look-ahead symbol, 'S+' and the unprocessed symbols remaining on the LR-stack. The Input-stack now consists of the following symbols :

. *Bob V S+* **.**

The sixth phrase extracted by the RLD and LRD networks is *Bob* and the REC-stack contains :

. ^^^ <u>***Bob*** ^ ^ ^ ^ ^ ^ ^ ^ ^ ^</u> *V*

The Recogniser network performs a forward-pass computation and *N* is selected as the nearest matching constituent tag. The following is pushed onto the Parse-stack :

[N Bob_**NP N]** : *inactive*

The Input-stack is updated by pushing back on the look-back symbols, the constituent tag *N*, the look-ahead symbol, '*V*' and the unprocessed symbols remaining on the LR-stack. The Input-stack now consists of the following symbols :

. *N V S+* **.**

Table 4.4 shows that the last phrase to be extracted by the RLD and LRD network is *N V S+* and the REC-stack contains :

. ^^^<u>*N V S+* ^ ^ ^ ^ ^ ^ ^ ^</u> .

The Recogniser network performs a forward-pass computation and the compound sentence tag, *S&*, is selected as the nearest matching constituent tag indicating that a successful parse has been achieved. As there are constituent tags within the phrase itself, the Scheduler first matches each constituent tag against matching *inactive* constituent tags already on the Parse-stack to obtain the *final* parse state.

It can be seen that the following sentence structure is formed by the parser for sentence S4.2 :

[S& [N Bob_**NP N]** [V knows_**VBZ V]** [S+ but_**CC** [Na he_**PP3A Na]** [V would_**MD** nt_**XNOT** talk_**VB V]** [V would_**MD** nt_**XNOT** talk_**VB V]** [R either_**RB R]** S+] **S&]**

Note that the compound sentence constituent tag, *S&*, is active by default as it is the most dominant constituent tag and represents the entire sentence structure. This is the final parse state and corresponds to the traditional tree representation illustrated in figure 4.6.

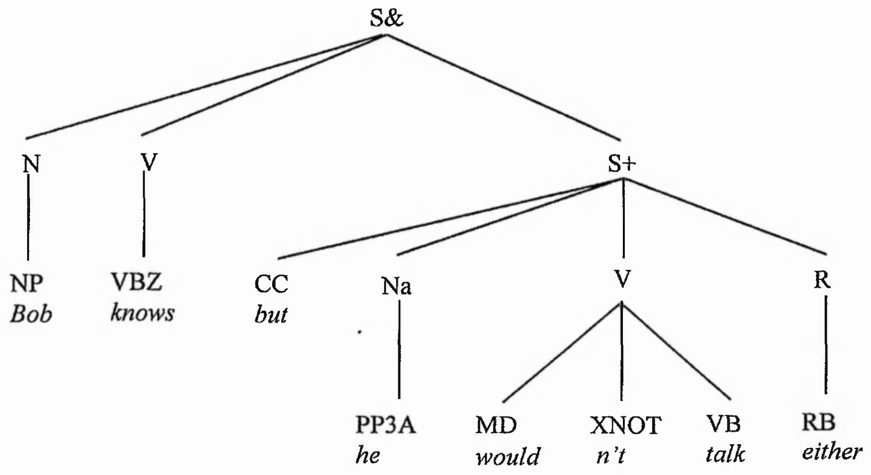


Figure 4.6 : Parse tree for *Bob knows but he would n't talk either.*

4.4. Composition of the Training and Test Samples

This section provides an explanation of the training and test samples extracted from the LPC. Section 4.4.1 first details how sentences from the LPC are converted into training sequences for the RLD, LRD and Recogniser networks. Section 4.4.2 discusses the problem of natural replication of structure and how it will be processed. The problem of structural conflicts that occur in the corpus and how they will be removed from the training and test data is also discussed in this section. Section 4.4.3 provides a description of the actual training sample and its general linguistic properties. Section 4.4.4 provides a description of the test sample and its general linguistic properties. Finally, Section 4.4.5 defines the test strategy that will be used to test the performance of the parser.

4.4.1. Sentence Decomposition

Each sentence in the LPC can be broken down into training and test sequences for the RLD and LRD networks and into training and test patterns for the Recogniser. For the purpose of this discussion, as in Section 4.3.4, the maximum number of look-back symbols allowed for the RLD network will be assumed to be 4; the number of look-back symbols used by the LRD network will be assumed to be 2; the maximum number of look-ahead symbols allowed for the LRD network, 2; and it will be assumed that only one of the look-ahead symbols processed by the LRD network can be used by the Recogniser network to aid its decision. The maximum phrase length is fixed at 10 symbols as before. Chapters 5 and 6 will detail the optimisation of these parameters for the RLD, LRD and Recogniser networks respectively.

The RLD and LRD networks will be presented with training and test *sequences* and the Recogniser networks will be presented with training and test *patterns*. It is important to distinguish the difference between sequences and patterns. The sequences defined for the RLD and LRD networks refer to a sequence of symbols that will be presented to the connectionist networks sequentially i.e., one symbol at a time. The length of each sequence is variable and dependant upon the sentence structure. The network must possess a memory to inherently establish temporal relationships between symbols as each is encountered on the input. If the sequence is a training sequence then a target output vector will be provided with each input symbol in the sequence. The form of the target output vectors is considered further in Chapter 5. If the sequence is a test sequence then no target vector is supplied and the actual output of the network is interpreted to decide whether to process further input. By contrast, the single training or test pattern presented to the Recogniser network should contain all the information it needs to classify the phrase type. Temporal relationships do not therefore need to be established as time is mapped onto space : all the input symbols are presented to the network in one step via the input layer. If the pattern is a training pattern then the desired output response i.e., a specific constituent tag, will be provided to the network so that the associated output error can be calculated. However, if the pattern is a test pattern then no desired output response is given and the output of the Recogniser network is used to indicate the constituent tag classification.

To illustrate how an LPC sentence is decomposed into input sequences and patterns, consider sentence S4.3 and its annotated form :

(S4.3) Mr Macleod was not at the week-end meeting .

[S[N Mr_**NPT** Macleod_**NP N][V** was_**BEDZ** not_**XNOT V] [P** at_**IN [N** the_**ATI** week-end_**NN** meeting_**NN N]P] S]**

Table 4.5 shows the training/test sequences and patterns extracted directly from the annotated version of S4.3 by processing from right-to-left and applying the look-back and look-ahead limits previously described.

	RLD Input Sequence	LRD Input Sequence	Recogniser Input Pattern	Recogniser Output Pattern
1	. **NN NN ATI** IN XNOT BEDZ NP	XNOT IN **ATI NN NN** . ^	NP BEDZ XNOT IN **ATI NN NN** ^ ^^^^^^ .	N
2	. **N IN** XNOT BEDZ NP NPT	BEDZ XNOT **IN N** . ^	NPT NP BEDZ XNOT **IN N** ^^^^ ^^^^ .	P
3	. P **XNOT BEDZ** NP NPT . ^	NPT NP **BEDZ XNOT** P .	. NPT NP ^ **BEDZ XNOT** ^^^^^ ^^^ P	V
4	. P V **NP NPT** . ^^^	^ . **NPT NP** V P	. ^^^ **NPT NP** ^^^^^^^^^ V	N
5	. **P V N** . ^^^	^ . **N V P** . ^	. ^^^ **N V P** ^^^^^^^^ .	S
6	. S . ^^^	^ . S . ^		

Table 4.5 : Training sequences and patterns extracted from an LPC sentence. The symbols in bold type are those in the phrase itself, the remaining symbols form look-ahead/look-back context.

As the phrase boundaries have already been defined in the LPC it is a simple task to extract phrases with look-back and look-ahead symbols in the desired right-associative manner. All the LPC sentences used in training and test experiments have been decomposed in this manner.

4.4.2. Removing Natural Replication and Conflicts

The raw training and test sentences extracted from the LPC will contain naturally occurring structural replication and conflicts. Structural replication occurs when different sentences share the same phrase structure. For example, most sentences will generate . *S* . ^ ^ ^ for the RLD network and ^ . *S* . ^ for the LRD network since it is the network's last sequence of the parse for the given sentence.

Also, since word tags are used rather than the words themselves many words share the same input representation and thus the level of replication across the entire set of training and test sentences is increased. The consequence of this natural replication for training is that the training sets generated will be imbalanced : some sequences will occur more often in the training set than others. These prominent sequences may be learnt by the network at the expense of sequences that occur less frequently. Duplicate training sequences for the RLD and LRD networks and duplicate training patterns for the Recogniser network are therefore removed.

Conflicts arise when an input pattern for the Recogniser network or input sequence for the RLD or LRD network occurs more than once within the respective training set and one or more occurrences have different target outputs associated with it. For example, consider the two following raw LRD sequences :

a) . ***AP21 AP22 NNS***

b) . *AP21 AP22 NNS* ***HV RB BEN VBN*** *P*

To indicate that sequences a) and b) encode different phrases the embedded phrase has been highlighted in bold. Assume that the LRD network has initialised its context units to some initial value, e.g., 0.5, before each input sequence and assume that the target output value for each input symbol is 0.0 except for the last input symbol of the sequence, which is 1.0 to signify the end of the sequence. During LRD training it is evident that sequences a) and b) will conflict when the LRD network processes the *NNS* symbol. In sequence a) the desired output value for *NNS* is 1.0 as it indicates the end of the sequence. The LRD network will then use the Back-propagation learning algorithm to correct its weight values in order to respond to *NNS* with 1.0. However, when the LRD encounters sequence b) it is attempts to adjust its weights so that its output response to *NNS* is 0.0 rather than 1.0 as it is not the last input symbol for this particular training sequence. This conflict will result in the network being unable to correctly respond to one of the sequences.

As with the resolution of lexical and phrasal ambiguity, the solution is to add further contextual information to resolve the sequence ambiguity. This will reduce the number of conflicts at the expense of increasing the length of the input sequences and thus increasing training times. The success of conflict resolution for the RLD network will be dependent upon the number of look-backs used, and for the LRD network on the number of look-back *and* look-ahead symbols. Chapter 5 details the optimum number of look-back and look-ahead symbols for the RLD and LRD networks. Chapter 6 details the optimum number of look-back and look-ahead symbols for the Recogniser network.

4.4.3. The Training Sample

To gain an sample of suitable size, every 8th sentence was extracted from the LPC for training. Table 4.6 gives the sentence length statistics for this sample. This sample provides a total of 1,478 sentences available for training data i.e., 12.5% of the entire LPC. However, this proved too ambitious using the computational resources available. For example, the training set (before balancing using sequence replication) for the RLD network consisted of 12,465 training sequences containing a total of 106,250 training patterns. The preliminary training experiments with this training set were performed on a SunSPARC 10 server and resulted in an average time of 5 hours per epoch. After 20 epochs, a TASRN network with 60 hidden units had managed to learn 100% of length 5 sequences and none of the other sequences.

Full Training Sample	
No of Sentences	1,478
% of LPC	12.5
Minimum Sentence Length	2
Average Sentence Length	13
Maximum Sentence Length	41

Table 4.6 : The initial training sample extracted from an LPC sentence.

These training times were deemed impractical and thus training was terminated and a method was defined to reduce the training set size. Since the aim of the research is to investigate and assess the approach to connectionist parsing taken rather than the computational power and language coverage alone, a complexity constraint was placed upon the LPC sentences in the full training sample. The complexity constraint is defined in terms of the number of symbols the RLD network needs to process before it recognises the beginning of a valid syntactic phrase i.e., if the RLD network has to process more than a predetermined (threshold) number of symbols before the beginning of a phrase is found then the sentence containing that phrase is removed from the training sample. The threshold imposed is 9 input symbols including look-back symbols, thus for a sentence to be included in the constrained training set the RLD network must encounter the actual beginning of each phrase in the sentence by the *fifth* input symbol (the remaining four symbols being look-back symbols).

Constrained Training Sample	
No of Sentences	654
% of LPC	5.5
Minimum Sentence Length	2
Average Sentence Length	7
Maximum Sentence Length	27

Table 4.7 : The complexity reduced training sample.

This reduces the size of the training set and the complexity of the constituent sentences with the expected consequence of limiting the parser's ability to process arbitrarily complex sentences. The composition of the resulting training sample is a subset of that shown in table 4.6 and can be seen in table 4.7. The actual training sample consists of 654 sentences and is 5.5% of the entire LPC corpus rather than 12.5% as before. The average sentence length of the training corpus is only 7 words compared to the 13 word average of the initial training corpus.

However, 5.5% is considered to be a reasonable coverage of the LPC which allowed computationally feasible training sets containing simple and complex sentence structures. For example, in terms of training set size, the RLD network's raw training set (before balancing using sequence replication) was reduced by 22.5% to 2,809 training sequences containing a total of 21,487 training patterns. Preliminary training experiments using a TASRN network with 110 hidden units with this training set were performed on a SunSPARC 10 server and resulted in an average time of 20 minutes per epoch. After 582 epochs, the TASRN had learnt only 11% of the total training sequences (76% of the training patterns) using the pattern-error sensitive learning rate. The sequences learnt consisted of all 277 of the shortest length sequences (6 symbols and which is also the minority sequence class) and 31 (3%) of the length 7 sequences. Although it was unable to learn input sequences above seven symbols, the training sets are suitable for balancing using sequence replication and can be adequately increased without significantly constraining the architecture size or resulting in impractical training times to potentially learn the remaining 2,501 unique training sequences.

Table 4.8 shows the actual linguistic properties of the training data in terms of the prominent types of phrase structure that naturally occur in the training sample. It can be seen that noun phrases occur more frequently than other phrase types in the training sample.

Predominant Phrase Types In The Constrained Training Sample			
Phrase Type	**Description**	**Occurrences**	**% of Training Data**
N	Noun phrase.	825	32.88
V	Finite verb phrase.	444	17.16
P	Prepositional phrase.	216	8.35
S	Sentence.	174	6.72
R	Adverbial phrase.	111	4.29
J	Adjective phrase.	90	3.48
Po	Prepositional phrase beginning with 'of'.	81	3.13
Vi	Non-finite verb phrase.	63	2.43
Na	Noun phrase marked as the subject of the verb.	63	2.43
Ti	To-infinitive clause.	59	2.28
Nq	Wh-noun phrase.	32	1.24
Fn	Finite nominal clause.	32	1.24
Fr	Relative clause.	28	1.08
Vg	Non-finite verb phrases (for –ing clause).	26	1.00
Tg	-ing clause.	25	0.97
S&	Compound sentence.	22	0.85
Vn	Non-finite verb phrases (for past-participle clause).	18	0.70
N+	Subsequent conjoin of a compound noun phrase.	18	0.70
N&	Compound noun phrase.	17	0.66
Fa	Finite adverbial clause e.g., finite subordinate clause of time and/or condition.	15	0.58

Table 4.8 : The predominant phrase types present in the constrained training sample.

This is expected as every training sentence will contain at least one type of noun phrase. Generally, the major phrase groups such as noun phrases, finite verb phrases, prepositional phrases, adverbial phrases and adjective phrases *naturally* occur more frequently than other phrases in the LPC. It is these phrase types that form the fundamental structure of natural language sentences.

4.4.4. The Test Sample

As with the training sample, the test sentences must be sampled across all text categories in the LPC. However, when sampling the LPC for test sentences it is essential that none of the test sentences are present in the training sample. For this reason, the LPC was sampled at every 8th + 1 sentence for test purposes. This results in a total of 1,478 unique sentences available for test data i.e., 12.5% of the entire LPC.

However, this represents an unconstrained test sample that will contain sentence structures which exceed the complexity of those structures in the training sample and thus represent an unfair test domain. The complexity constraint applied to the training sample (i.e., the number of symbols the RLD network needs to process is restricted to 9) is also applied to the test sample to form a constrained test sample. This will enable tests to be performed with sentences that are comparable in terms of complexity to the training set sentences.

Table 4.9 shows the resulting constrained test sample that will be used to test the parser.

Constrained Test Sample	
No of Sentences	687
% of LPC	5.8
Minimum Sentence Length	2
Average Sentence Length	7
Maximum Sentence Length	27

Table 4.9 : The constrained test sample that will be used to test the parser.

As with the constrained training sample, the constrained test sample is also nearly 6% of the entire LPC at 5.8% and the average sentence length is also 7 words.

Table 4.10 shows the actual linguistic properties of the constrained test sample in terms of the prominent types of phrase structure that naturally occur in the sample. As with the composition of constrained training sample, it can be seen that the major phrase groups such as noun phrases, finite verb phrases, prepositional phrases, adverbial phrases and adjective phrases *naturally* occur more frequently than other phrases in the LPC.

Predominant Phrase Types In The Constrained Test Sample			
Phrase Type	**Description**	**Occurrences**	**% of Test Data**
N	Noun phrase.	872	31.54
V	Finite verb phrase.	467	16.89
P	Prepositional phrase.	246	8.90
S	Sentence.	200	7.23
R	Adverbial phrase.	126	4.56
J	Adjective phrase.	81	2.93
Na	Noun phrase marked as the subject of the verb.	76	2.75
Po	Prepositional phrase beginning with 'of'.	75	2.71
Vi	Non-finite verb phrase.	74	2.68
Ti	To-infinitive clause.	68	2.46
Fn	Finite nominal clause.	47	1.70
Fr	Relative clause.	35	1.27
Nq	Wh-noun phrase.	34	1.23
Vg	Non-finite verb phrases (for –ing clause).	32	1.16
Tg	-ing clause.	31	1.12
S+	Subsequent conjoin of a compound sentence.	27	0.98
N&	Compound noun phrase.	19	0.69
S&	Compound sentence.	17	0.61
Fa	Finite adverbial clause e.g., finite subordinate clause of time and/or condition.	16	0.58

Table 4.10 : The predominant phrase types present in the constrained test sample.

4.4.5. Test Methodology : Natural and Pure Generalisation

Although the sentences contained in the training sample will not appear in the test sample, a number of matching syntactic structures will be found in the test sample because in some cases, sentences will coincidentally share the same sentence or clause structure as those in the training sample. A certain level of natural overlap of syntactic structure is therefore expected and for this reason, testing of the connectionist networks with the constrained test sample will be performed in two phases : tests for *natural* generalisation and tests for *pure* generalisation.

Tests for natural generalisation will include all sequences (for the RLD and LRD networks) and patterns (for the Recogniser network) generated from the test samples whether they occur in the respective training sets or not. Tests for pure generalisation will exclude all sequences and patterns in the test samples that also occurred in the respective training sets. The pure generalisation figure will provide a measure of how well the connectionist networks have learnt to generalise from the training data to different sequences/patterns within the test data.

The natural generalisation rate, G_n, is calculated as :

(E4.1)

$$G_n = \frac{N_c}{N_t}$$

where N_t is the total number of test patterns or sequences (depending upon the connectionist module being tested) and N_c is the number of test patterns/sequences classified correctly calculated as :

(E4.2)

$$N_c = N_x + N_y$$

where N_x is the number of overlapping patterns/sequences correctly classified and N_y is the number of unique test patterns/sequences classified correctly.

Similarly, the pure generalisation rate, G_p, is calculated as :

(E4.3)

$$G_p = \frac{N_y}{N_p}$$

where N_p is the total number of unique test patterns/sequences.

The greater the pure generalisation rate, G_p , the more the connectionist network is deemed to have learnt the essential features of the training data rather than memorised them and thus the better the performance.

4.5. Summary

This chapter has provided a high-level description of the hybrid syntactic parser that integrates modular connectionist architectures with symbolic structures to automatically learn syntactic structure from a large text corpus. The modular architecture developed is motivated by the results of the preliminary investigations presented in Chapter 3. The Lancaster Parsed Corpus (LPC) was presented as the fundamental basis of the grammatical framework for the parser.

The parsing architecture and algorithm developed was discussed and demonstrated. Also, the composition and linguistic properties of the training and test samples were also discussed. Chapter 5 presents the RLD and LRD connectionist architectures in detail explaining training performance and generalisation results and their computational adequacy and limitations.

Chapter 5

Phrase Segmentation

5.1. Introduction

This chapter provides a detailed description of the phrase segmentation aspects of the corpus-based parsing model. Segmenting natural language sentences into clauses and clauses into phrases using two recurrent connectionist networks simplifies the overall task of parsing. The phrase segmentation technique developed here is a method of extracting phrases from sentences without the need for predefined grammar or parse rules. A number of systematic experiments using the training and test samples demonstrate the effectiveness of this technique for extracting the right-most reducible phrase from natural language sentences.

Section 5.2 begins with a brief overview of the phrase segmentation technique in the parsing model. Section 5.3 describes the output target function for the phrase segmentation networks. Section 5.4 describes the recurrent connectionist architectures that constituent the Right-to-Left Delimiting (RLD) network and the Left-to-Right Delimiting (LRD) network that perform the segmentation task. Section 5.5 presents the RLD network and its training and test performances. Section 5.6 presents the LRD network and its training and test performances. Finally, Section 5.7 concludes with a brief summary of the chapter.

5.2. Overview of Phrase Segmentation

Phrase segmentation is the process of discovering the beginning and end boundaries of valid syntactic phrases within a sentence. It plays a fundamental role in the parsing model and is implemented as a two stage process. Since the parser's processing strategy is strictly right-associative the first stage is to discover the beginning of a phrase via scanning the sentence starting from the right-most symbol and then shifting-left towards the left-most symbol of the sentence until the beginning of a phrase is found or there are no input symbols remaining. This first task is assigned to the RLD connectionist network. The RLD network can only process one symbol at a time and must establish its own temporal relationships between the input symbols in order to identify the symbol that indicates the beginning of a phrase. The second stage is to discover the end of the phrase via taking as input the symbols processed by the RLD network starting from the last left-most symbol and then shifting-right towards the right-most symbol of the sentence until the end of the phrase is found or there are no input symbols remaining. This second task is assigned to the LRD connectionist network. As with the RLD network, the LRD network can only process one symbol at a time and must establish its own temporal relationships between the input symbols.

After the RLD network has found the beginning of the phrase and the LRD network has found the end of the phrase both connectionist networks must reinitialise their 'internal' context units so that they can 'forget' unwanted context.

The extracted input symbols, which may consist of word tags and/or constituent tags, are then passed onto the Recogniser network which will attempt to recognise the phrase that has been extracted.

It must be noted that only one phrase can be extracted from the sentence at any one time. Another phrase can only be segmented from the sentence when the Recogniser has recognised the previously extracted phrase and the input stack and parse stack have been updated accordingly i.e., the constituent tag for the recognised phrase is pushed onto the Input-stack.

5.3. *Semi-Linear Output Targets (SLOTs)*

The length of input sequences to the RLD and LRD networks will be of variable length and the objective of the networks is to produce a maximum activation of 1.0 when the beginning or end of a phrase is encountered (depending on the network's task). The simplest method for training purposes would be to associate target values with the last symbol of each sequence i.e., the rest of the input symbols have a don't care output and hence no learning cycle occurs until the target value is processed. However, the problem here is that the parser will be monitoring the network output activation after every pattern presentation and an output activation of 1.0 may appear before the end of the sequence. Another method would be to assign 0.0 as the target output activation for all but the last input symbol of each sequence. However, this provides no indication as to the position of each symbol within a sequence and a pattern imbalance would occur in the training set as there will always be more patterns belonging to the 0.0 pattern class than the 1.0 pattern class. This could be overcome by replication of class '1.0' patterns in the training data, however, a better method perhaps is to inform the network of how far towards a phrase boundary it is.

The target output activation for each input symbol in the RLD and LRD training sequences will be a value proportional to its position within the input sequence. The target output activation for the first symbol will always be a 'don't care' value as more than three symbols must be processed before the connectionist network can produce the maximum activation. The second symbol of a sequence will have a 0.0 target output value as this constitutes the beginning of the learning cycle. The second to last input symbol will have a 0.4 target output value to represent its closeness to the last input symbol. The last input symbol will have a target output value of 1. The intervening input symbols will have a linear target output value within the range of 0.0 and 0.4 and which is representative of each symbols position within the input sequence. This enables the RLD and LRD networks to associate positional information with each input symbol whilst retaining a significant margin between the last symbol and all previous ones.

This *semi-linear output target (SLOT)* function is illustrated in figure 5.1. The linear value within the range of 0.0 and 0.4 for intervening input symbols is defined as :

(E5.1)

$$target\_output\_for\_symbol\_x = \frac{0.4*(x-2)}{seqlen-3}$$

where x is the position of the current symbol and *seqlen* is the sequence length.

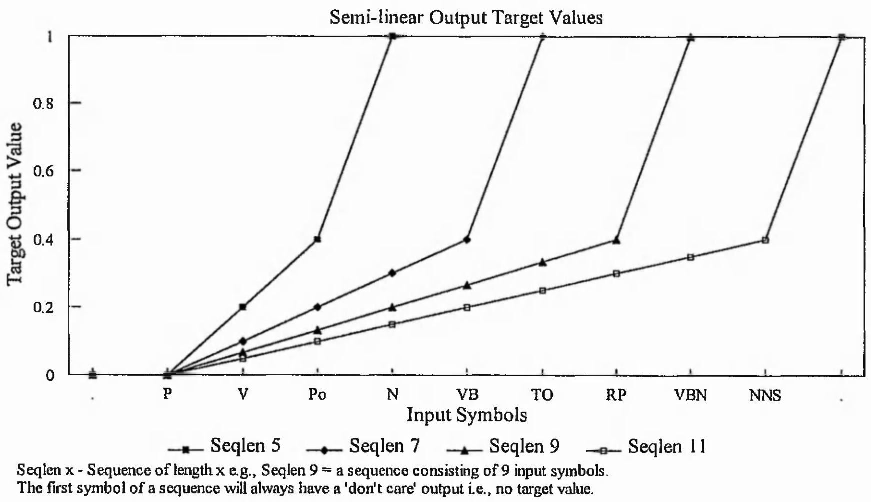


Figure 5.1 : Graph illustrating the use of semi-linear output targets for training sequences.

A limitation of assigning SLOT values to input symbols is that conflicts will arise when similar input sequences are encountered during training. The only actual training sequence in Figure 5.1 is the sequence with all 11 input symbols, the others are simply subsets of this sequence to illustrate the SLOT function. However, sequences that are subsets of other sequences will occur elsewhere in the training set. This results in the same input pattern having different output target values in different sequences. The conflict is only resolved when the last symbol of a sequence is encountered which is obviously different from the other sequences. For example, consider the two conflicting input sequences :

a) DTI NN INO NP
b) DTI NN INO N VBN

The target output values for the tags DTI and NN, will be the same for both sequences i.e., * and 0.0 respectively, where * means 'don't care'. However, since the sequences are of different length, the symbol, INO, will have a different target output value. Although this minor conflict may hinder training it is not detrimental to the learning task as a number of training experiments were performed with a small sub-set of the training data using binary output and SLOT output values. In all cases, the TASRN networks were able to learn all the training data with SLOT output values in fewer epochs than the training data with binary output values.

A threshold value of 0.7 is used to test the output. This compensates for the output variations caused by these minor conflicts since only symbols that force an output activation above 0.7 signal the end of a valid sequence, and this represents a situation of high 'confidence' in having reached a sequence end. This also illustrates why the margin between the penultimate symbols (0.4) and the last one (1.0) is helpful.

The following sequences are typical examples of conflicts that *are* detrimental to training, irrespective of the SLOT function :

c) DTI NN INO NN
d) DTI NN INO NN VBN

These types of conflicts are removed from the training set in favour of the longest sequence. In the above example, sequence d) would remain in the training set and sequence c) would be removed. The motivation behind this is that the parser will learn to expect more information before making its decision rather than making it prematurely. Optimising the amount of look-back for the RLD network and the amount of look-back and look-ahead for the LRD network will reduce the number of detrimental conflicts within the training set (see Sections 5.5.1 and 5.6.1).

5.4. Temporal Auto-Associative Simple Recurrent Networks

The connectionist architecture used for either the RLD network or the LRD network must possess the following attributes :

- The architecture must be able to process each input symbol of the training or test sequence sequentially as the sequence will be of arbitrary length.

- The architecture must possess a memory in order to learn temporal relationships between each input symbol in the sequence.

Strictly feed-forward MLP architectures trained with Back-propagation are unsuitable for processing temporal input sequences as the input layer is of fixed size and context must be encoded spatially onto the input layer. The connectionist architecture developed in Chapter 3 for phrase delimiting and recognition attempted to use a temporal input window to enable strictly feed-forward MLPs to process unconstrained input sequences. However, there were numerous disadvantages with this approach : the resulting training and test performance was considerably poor given the size of the grammar the MLP network was expected to learn; the temporal context is limited to the size of the input window, thus if the contents of the input window cannot be disambiguated then the parse will fail on the current input. These limitations have also occurred in other connectionist parsers that have implemented feed-forward MLPs with temporal input windows.

Elman [19] developed the Simple Recurrent Network (SRN) to extend the feed-forward MLP architecture to include recurrent connections from the hidden layer to an extended input layer (see figure 5.2). This forms a memory system in that the recurrent connections put cycles in the spread of activation through the network to allow the network to store state information. Each symbol from an input sequence can then be input into the network sequentially, and the network's previous hidden unit state is used to constrain the processing of the current symbol. This represents time by the effect it has on processing rather than by additional dimensions of the input. The SRN has predominantly been used for sequentially processing linguistic input [19, 26, 62, 63, 74, 127, 137, 138, 163] and is therefore an obvious candidate for the implementation of the RLD and LRD networks.

The input units that receive the corresponding hidden unit values during a processing cycle are called *context units*. The context units are hidden in that they do not explicitly interact with the outside world. Context units are commonly initialised to 0.5 before a new input sequence is presented to the SRN so that any previous hidden unit information is removed.

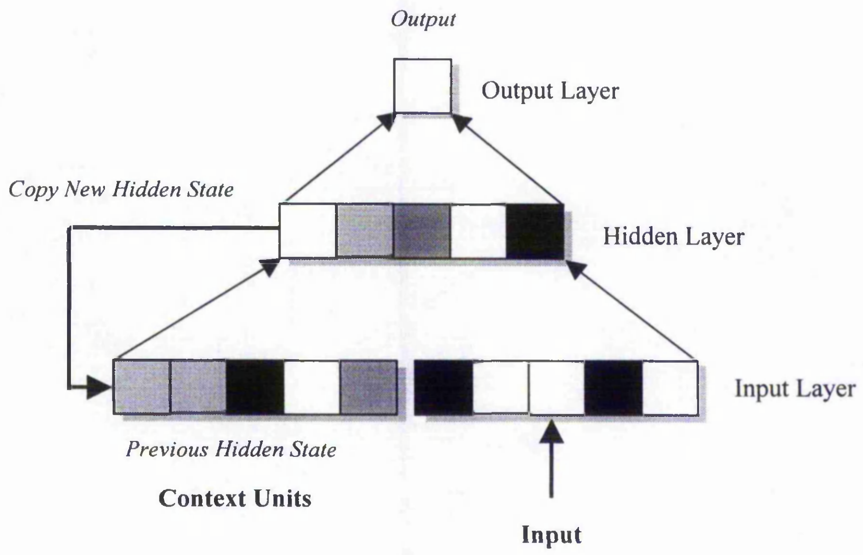


Figure 5.2 : The SRN architecture.

When a symbol from the input sequence is presented to the input layer, the input units and the context units activate the hidden units; and the hidden units then activate the output units. The hidden unit activations are then fed back to activate the context units ready for the next input. The reason for this is that recurrence of hidden unit activations is more powerful than recurrence of output unit activations for cognitive tasks [132], since they represent an internal reduced description of the input representation. The network is therefore able to build its own representation of time.

Depending upon the task there may or may not be a learning phase in a time cycle. When there is a learning phase the output activations are compared with the target output response and the Back-propagation algorithm is used to adjust the connection strengths. Recurrent connections are fixed at 1 with no adjustment. At time *t+1* the next input in the training sequence is presented on the input units but the context units now contain values which are exactly the hidden unit values at time *t*. Context units serve to remember previous internal states and thus provide the memory. Hidden units develop representations which are useful encodings of temporal properties of the sequential input and are sensitive to temporal context. The sensitivity of the representation to temporal context forces the SRN to produce an output response based upon the current input but with reference to the previous inputs processed.

SRNs possess the necessary attributes to represent the tasks associated with phrase segmentation. However, the vast amount of research carried out with SRNs have revealed their limitations. The recursive representations developed within an SRN begin to degrade after about three levels of embedding [19, 74, 132]. This indicates that the memory is bounded with an upper limit, after which the network will lose or 'forget' previous inputs in the sequence. This would be detrimental to the performance of the parser if input sentences generate long sequences for the phrase segmenting networks. This is particularly important with the RLD network as the RLD network may process up to nine input symbols (it would be greater than 9 but for the complexity constraint applied) before recognising the beginning of a phrase.

Ghahramani and Allen [132] identify the use of 'don't care' cycles and the lack of error-correction performed on the recurrent connections as the main reasons why SRNs forget previous input information. A don't care cycle is where the SRN's output response to the current input pattern (i.e., a symbol from an input sequence) is ignored and no corresponding error-correction is performed to adjust the response. These don't care inputs therefore have no target output vectors and although the hidden unit response given for a don't care input may be incorrect the hidden unit state is still copied back to the context units without correction. The problem here is in making associations between the input symbols that have target output vectors. Such associations can only be made after intervening input symbols, which may not have target output vectors, have been processed. The hidden state therefore becomes distorted and associations between the input symbols with target output vectors become 'diluted' and the network 'forgets' previous input information.

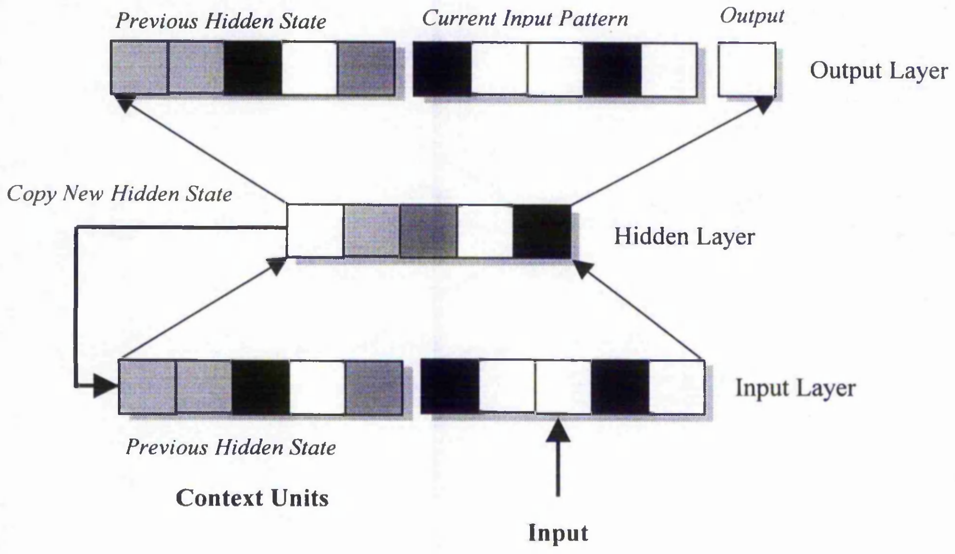


Figure 5.3 : The TASRN architecture.

As the number of don't care cycles associated with the training sequences for the RLD and LRD network is 1, the noise added to the hidden state due to unconstrained processing is not considered a major concern. However, the tendency of the SRN to forget previous input information with respect to sequence length is of concern. Ghahramani and Allen addressed both problems by extending the SRN architecture to perform Temporal Autoassociation. This extension to the SRN will be termed Temporal Autoassociative SRN (TASRN) and the architecture is illustrated in figure 5.3. The basic idea of the TASRN is to use autoassociation to ensure that the hidden unit representation retains information about previous inputs for as long as possible regardless of whether there is a current target or not. This is achieved by adding additional output units that encode the context information and the current input pattern. These 'internal' output units always have target vectors associated with them, even when the 'external' output forms a don't care cycle, therefore an error-correction is always performed.

To compare the performance of TASRN networks with that of SRN networks for learning the RLD and LRD training sequences, a small subset of the RLD and LRD training data was extracted from the constrained training set. The composition of these training sets is shown in table 5.1.

Subset of Constrained Training Sample for SRN/TASRN Comparison			
RLD Sequences	**RLD Patterns**	**LRD Sequences**	**LRD Patterns**
168	1,428	115	805

Table 5.1 : Small subset of the constrained training data for the RLD and LRD networks.

The RLD and LRD training sets shown in table 5.1 are balanced in that deliberate replication is added so that sequences of different lengths are represented equally throughout the training set. This is achieved by first removing the natural replication of *sequences* (not patterns) and then deliberately replicating the sequences based on their sequence length. The level of replication is based on the most frequently occurring sequence length. For example, after removing natural replication from the LRD training set, there were a total of 54 sequences consisting of 323 patterns. The most frequently occurring sequence length was 6, occurring 23 times in total and thus setting the level of replication. The training sequences of lengths 5, 7, 8 and 9 were then replicated so that all sequence lengths occurred 23 times in the training set. As shown in table 5.1 this results in 115 sequences consisting of 805 patterns. The motivation behind this is that previous experiments with RLD and LRD training sets showed that, without balancing the training sets, the most frequently occurring sequence lengths were learnt at the expense of other training sequences. Experiments showed this to be detrimental to the network's performance in all cases even when using the pattern-error sensitive learning rate defined in Chapter 3. Unbalanced training sets will therefore not be considered further for the RLD and LRD networks.

Back-propagation with a momentum term is the learning algorithm used to train the SRN and TASRN networks in on-line mode. A momentum value of 0.9 is used for all experiments. Hidden unit and output unit activation values are calculated using the standard sigmoid function. However, the learning rate schedules used will be different. The first set of training experiments performed is to establish whether the SRN and TASRN could learn the RLD and LRD tasks with a fixed learning rate of 0.35. The objective of the second set of training experiments is to assess whether the pattern-error sensitive adaptive learning rate defined in Chapter 3 can aid the learning of the RLD and LRD tasks. All context units are initialised to 0.5 before each new input sequence. Each network will be presented with one input symbol per time step together with a semi-linear output target that is dependant upon the sequence length. The number of hidden units used is based upon the minimum required for the TASRN network to learn the RLD and LRD tasks. The SRN network is then trained using the same number of hidden units to obtain a comparable result. All training experiments are limited to convergence or 1,000 epochs whichever takes the least time.

Training Experiments Using a Fixed Learning Rate

The minimum number of hidden units required for the TASRN network to learn the RLD and LRD tasks was found to be 50 hidden units for the RLD network and 45 hidden units for the LRD network. The TASRN networks were both able to learn all training sequences within 1,000 epochs. In comparison, after 1,000 epochs the SRN network was unable to fully learn the RLD and LRD tasks with 50 and 45 hidden units respectively. This is illustrated in table 5.2.

It can be seen from table 5.2 that the actual RMS error values are misleading for the LRD task in that both the SRN and TASRN networks obtained an RMS error value of 0.05, however only the TASRN network has learnt all of the training sequences. Each test was repeated five times (with different starting weights) and the results were consistent.

Although the SRN network consistently obtained an RMS error value below 0.06 and thus appeared to have converged it still failed to learn all of the LRD training data with 45 hidden units. The SRN network was consistently unable to learn the RLD training data with 50 hidden units after 1,000 epochs.

Training Results for RLD Networks				
	Epochs	**RMS Error**	**%Patterns Learnt**	**% Sequences Learnt**
SRN	1,000	0.10	98.4	88.1
TASRN	830	0.05	100	100
Training Results for LRD Networks				
	Epochs	**RMS Error**	**%Patterns Learnt**	**% Sequences Learnt**
SRN	1,000	0.05	99.3	95.7
TASRN	537	0.05	100	100

Table 5.2 : Training results for the SRN and TASRN networks using a fixed learning rate of 0.35.

Training Experiments Using a Pattern-error Sensitive Learning Rate

The Pattern-error Sensitive learning rate defined in Chapter 3 was previously used to avoid the need for pattern replication. However, the RLD and LRD tasks present a temporal domain consisting of variable length input sequences that must be processed sequentially, so sequence replication (and thus pattern replication) is essential. The following tests attempt to assess whether using the pattern-error sensitive learning rate will provide more complete learning of the training data than the fixed learning rate. The TASRN networks were both able to learn all training sequences within 1,000 epochs. In comparison, after 1,000 epochs the SRN network was again unable to fully learn the RLD and LRD tasks with 50 and 45 hidden units respectively. This can be seen in table 5.3.

Training Results for RLD Networks				
	Epochs	**RMS Error**	**%Patterns Learnt**	**% Sequences Learnt**
SRN	1,000	0.10	97.6	84.5
TASRN	1,000	0.06	100	100
Training Results for LRD Networks				
	Epochs	**RMS Error**	**%Patterns Learnt**	**% Sequences Learnt**
SRN	1,000	0.07	98.8	93
TASRN	649	0.05	100	100

Table 5.3 : Training results for the SRN and TASRN networks using the pattern-error sensitive learning rate.

It can be seen from table 5.3 that the pattern-error sensitive learning rate has *not* improved the overall learning performance of the SRN networks in this instance. However, the RLD task is the harder problem for both the SRN and TASRN to learn. In all experiments, the SRN was unable to learn all the RLD and LRD training sequences within 1,000 epochs. Although the TASRN took longer to learn the RLD and LRD tasks using the adaptive learning rate rather than the fixed learning rate, the TASRN networks fully learnt all training sequences within 1,000 epochs. The adaptive learning rate schedule will continue to be used in order to allow further assessments.

The obvious trade-off between the TASRN and SRN architecture is the requirement for additional output units and a consequential increase in the number of connections. However, the TASRN consistently outperformed the SRN during the numerous experiments performed with the training data sub-sets. Future experiments will therefore use TASRNs with the pattern-error sensitive learning rate.

5.5. Right-to-Left Segmentation

As mentioned previously, the RLD's task is to find the beginning of the right-most reducible syntactic phrase of an input sentence. It must start with the right-most input symbol (i.e., word or constituent tag) of the input sentence and process each symbol to its left[18] until it has encountered the input symbol denoting the beginning of a phrase and has also processed a fixed number of look-back symbols for contextual analysis. The RLD's task is the most difficult in the phrase segmenting stage as it always has to process all of the input symbols to the right of a possible phrase. The complexity of the task is obviously dependant upon the sentence structure. If the right-most syntactic phrase is embedded deep within the sentence then the number of intervening input symbols may be large thus increasing demands on the RLD's memory system. Additional look-back symbols also have to be processed. However, a complexity constraint has been enforced on the training sample so that the complexity of the training sentences is such that the RLD will only have to process a maximum of nine input symbols (including look-back symbols) until the beginning of the right-most phrase is discovered. This Section details the minimum number of look-backs required for the RLD to remove conflicting sequences that are detrimental to the parsing process. The training and test performance of the TASRN network for this task is then fully detailed.

5.5.1. Optimising the Look-Back

The look-back symbols of the RLD sequences are those symbols immediately to the left of the phrase. These could be viewed as overhead symbols as they are not part of the phrase, however they are essential in the segmentation process. Look-back symbols aid the correct recognition of the phrase by providing the parser with a 'view' of the context to the left of the phrase. This will aid the resolution of sequence conflicts and ambiguity in the training data.

There must be a fixed number of look-back symbols available as using a variable number of look-back symbols will complicate the recognition task as it will be difficult to distinguish the boundary between the phrase and the look-back

[18] In terms of the whole operation of the parser, the RLD is shifting-left across the input sentence.

symbols[19]. Look-ahead symbols are not applicable to the RLD task as all the input symbols to the right-hand side of the phrase will always be processed by the RLD network. If there are an insufficient number of input symbols available for look-back then the null symbol, ^, will be used to pad out the number of look-back symbols to the required length.

All conflicts in the RLD training sequences were resolved with only *4* look-back symbols (including additional null symbols)[20]. This resulted in 2,809 training sequences consisting of 21,487 training patterns in total without any conflicting sequences having to be removed.

5.5.2. Training Experiments Performed

The RLD training data is balanced as explained in Section 5.4. The composition in terms of sequence length of the original (raw) and balanced sets for the training data is shown in table 5.4. A similar pattern is seen for the LRD training data.

RLD Training Sets						
	Length 6 Sequences	**Length 7 Sequences**	**Length 8 Sequences**	**Length 9 Sequences**	**Total Sequences**	**Total Pattern s**
Raw Training Sequences	277	1,015	933	584	2,809	21,487
Balanced Training Sequences	1,015	1,015	1,015	1,015	4,060	30,450

Table 5.4 : The raw and balanced RLD Training sets.

The problem of balancing the training set in this way is that the sequence length used as the basis for replication will actually contain no replicated sequences. For example, although sequences of length 7 will appear in the training set with the same frequency as all other training sequences, each length 7 sequence is unique where as other sequence lengths will contain replicated sequences. It is therefore envisaged that the RLD network will find it more difficult to learn those sequence lengths that have little or no replication i.e., lengths 7 and 8, than those that do have high levels of replication.

Back-propagation with a momentum term of 0.9 is the learning algorithm used to train the TASRN networks in on-line mode for the RLD task. Each network has 61 input units and 1 output unit excluding context units and 'internal' output units (see figure 5.3) . A bias unit of 1.0 is included in the input and hidden layer. Hidden unit and output unit

[19] Both the number of look-back and look-ahead symbols used by the parser will be of fixed length as the Recogniser's task will be to perform position invariant phrase recognition rather than position variant.

[20] This refers only to conflicting *sequences* found in the RLD training set after natural replication has been removed. Resolving structural conflicts from the training *sentences* also involves finding the optimum number of look-back and look-ahead symbols for the LRD sequences.

activation values are calculated using the standard sigmoid function. All training experiments use the balanced training set detailed in table 5.4 and are performed on SunSPARC 10/20 Servers or a P100MHZ Pentium Personal Computer (PC).

Training Results for the RLD Network					
Hidden Units	**Connections**	**Epochs**	**RMS Error**	**% Patterns Learnt**	**% Sequences Learnt**
100	32,562	914	0.0866469	98.05	88.13
115	40,887	2,207	0.0918511	98.12	87.98
150	63,812	424	0.0785796	98.68	91.87
165	75,137	312	0.0717479	98.80	92.22

Table 5.5 : Results of the RLD training experiments performed.

The pattern-error sensitive learning rate is used for all experiments. As the average time taken per epoch is 2 hours no attempt has been made to optimise learning with respect to generalisation. The objective of the training experiments is therefore to learn as much of the training data as possible rather than to find the optimum number of hidden units. The resulting generalisation capability of the network will be relied upon to gain a larger coverage of the overall corpus and of the English language in general, and this will be evaluated.

Table 5.5 summarises the results of the training experiments performed using TASRN networks with a number of hidden unit configurations. The minimum number of hidden units required to gain acceptable training results (above 98% of patterns learnt) for the RLD task was found to be 100. A limit of 165 hidden units was imposed as further hidden units resulted in unreasonable training times. The results of the training experiments performed illustrates an interesting relationship between the RMS error value and the percentage of sequences and patterns learnt. As the number of hidden units increased to 115 it can be seen that although the RMS error is higher than that of 100 hidden units at 0.09, the percentage of patterns learnt is higher at 98.12% and the percentage of sequences learnt is lower by 0.15% at 87.98%. The RMS error value is calculated on a pattern per pattern basis rather than at a sequence level and can therefore be misleading. A training sequence consists of *n* patterns and if the network correctly learns all except for one pattern within a sequence, then the network has failed to learn the entire training sequence. This accounts for the TASRN with 115 hidden units[21] learning more training patterns and less training sequences than the TASRN with 100 hidden units. The breakdown of training results shown in table 5.6 illustrate that increasing the number of hidden units results in the longer training sequences being learnt at the expense of the shorter training sequences.

[21] This network configuration was trained for the longest period. The TASRN was allowed to run uninterrupted on a Sun SPARC 20 for 3 months.

Breakdown of Training Sequences Learnt					
Hidden Units	**% Length 6 Sequences**	**% Length 7 Sequences**	**% Length 8 Sequences**	**% Length 9 Sequences**	**% Total Sequences**
100	96.45	90.05	82.66	83.35	88.13
115	91.53	82.96	84.43	93.00	87.98
150	98.92	90.25	85.22	93.10	91.87
165	95.57	88.77	87.78	96.75	92.22

Table 5.6 : A breakdown of the training results for the RLD network.

The TASRN with 100 hidden units has managed to learn above 90% of length 6 and 7 sequences and below 84% of the length 8 and 9 sequences. The shorter sequences therefore account for its high percentage of sequences learnt. The TASRN configuration with 115 hidden units has managed to learn above 90% of the length 6 and length 9 sequences. However, a reduction of at least 5% occurs for sequences of length 7 and 8. Increasing the hidden units to 150 resulted in a substantial improvement in sequences learnt. This TASRN configuration managed to learn above 90% of sequence lengths 6, 7 and 9. In particular the network was able to correctly learn 98.9% of all length 6 sequences. The majority of length 7 and 8 sequences are unique and this is reflected in the amount of these sequences learnt. Consequently, the TASRN configuration with 150 hidden units was unable to learn above 86% of length 8 sequences within the number of training epochs allowed.

Increasing the number of hidden units further to 165 units provided similar results to that of the configuration with 115 hidden units. The sequences that were frequently replicated during the balancing process were learnt at the expense of the unique sequences. The TASRN with 165 hidden units was able to learn above 95% of length 6 and 9 sequences within the number epochs allowed. Although nearly 89% of the length 7 and 8 sequences were learnt, using 165 hidden units enabled the majority of all training sequences to be learnt.

The resulting generalisation performance of each of the TASRN configurations will determine the amount of language coverage achieved using the complexity constraints defined. The TASRN configuration with the highest level of *sequence generalisation* for the constrained test sample will be selected as the RLD network for the parser.

5.5.3. Test Results

The configurations of 100, 115, 150 and 165 hidden units are tested with the constrained test sample which represents 5.8% of the entire LPC corpus. This will produce generalisation results that are directly comparable with the training results as the test data will be constrained in the same way as the training data. All natural replication of sequences within the test set have been removed, in order to measure performance across the full range of sequence types in an unbiased way.

The testing phase will be performed in two stages. The first stage is to test each configuration's natural generalisation capability, whereby test sequences that may coincidentally occur in the training set remain in the test set. The second stage is to assess each configuration's pure generalisation capability.

5.5.3.1. Constrained Test Sample

The composition of the constrained test sample is shown in table 5.7. It can be seen that the test sample consists of nearly 89% unique test sequences (i.e., sequences that do not appear in the training sample) for the RLD networks. All sequence replication is removed. Table 5.7 shows the composition of the natural test data and the resulting pure test data (pure in that all test sequences that coincidentally occur in the training data have been removed from the natural test data).

Composition of the RLD Constrained Test Sample						
	Length 6 Sequences	**Length 7 Sequences**	**Length 8 Sequences**	**Length 9 Sequences**	**Total Sequences**	**Total Pattern s**
Natural	322	1,049	971	662	3,004	23,001
Pure	285	935	867	576	2,663	20,375
% Unique	88.51	89.13	89.29	87.01	88.65	88.58

Table 5.7 : Composition of the RLD Constrained Test Sample.

Natural Generalisation

Table 5.8 summarises the natural generalisation results obtained from the four TASRN network configurations. As expected, the TASRN with 100 hidden units correctly classified more length 6 and 7 sequences than any other TASRN configuration in the test experiments. However, its performance greatly reduced when the sequence length increased to 8 and 9 symbols, with the TASRN generalising to 78% and 73% respectively. This is expected as the majority of training sequences this configuration learnt were of length 6 and 7.

The TASRN with 115 hidden units produced the lowest overall levels of sequence generalisation at 79.26%. This configuration produced its best generalisation results for lengths 6 and 9. The TASRN with 150 hidden units generalised to approximately 86% of sequences of length 6, 7 and 9. However, the generalisation figure dropped by 4% for length 8 sequences. The TASRN with 165 hidden units produced the highest levels of sequence generalisation overall. This configuration was able to achieve generalisations above 80% for all sequence lengths and correctly processed 92% of length 9 sequences i.e., 6% higher than any other TASRN configuration.

Natural Generalisation Results for the Constrained Test Sample						
Hidden Units	% Length 6 Sequences	% Length 7 Sequences	% Length 8 Sequences	% Length 9 Sequences	% Patterns Correct	% Sequences Correct
100	88.51	87.80	78.06	72.96	96.90	81.46
115	84.78	75.79	78.06	83.84	96.71	79.26
150	86.02	86.94	82.29	85.80	97.65	85.09
165	86.65	86.46	83.93	91.84	97.92	86.85

Table 5.8 : Natural generalisation results for the RLD constrained test sample.

Pure Generalisation

Table 5.9 summarises the pure generalisation results obtained from the four TASRN network configurations. As expected, the generalisation figures are similar to those obtained during the tests for natural generalisation. This is due to 90% of the constrained test sample consisting of pure RLD test sequences i.e., 90% of the sentences within the test sample contain subtle differences in their sentence structure. Given the number of standard syntactic reductions, the differences are largely due to the variations in context.

Pure Generalisation Results for the Constrained Test Sample						
Hidden Units	% Length 6 Sequences	% Length 7 Sequences	% Length 8 Sequences	% Length 9 Sequences	% Patterns Correct	% Sequences Correct
100	87.02	86.42	76.01	69.79	96.57	79.50
115	82.81	74.01	76.24	81.77	96.40	77.36
150	84.21	85.56	80.51	83.85	97.39	83.40
165	84.91	84.92	82.35	90.80	97.68	85.35

Table 5.9 : Pure generalisation results for the constrained RLD test sample.

Although the TASRN network with 100 hidden units is able to generalise to 87% and 86% of length 6 and 7 sequences respectively, its performance reduces by 10% as the sequence length increases to 8 symbols and a further 6% as the sequence length increases to 9 input symbols.

The TASRN configuration with 115 hidden units performed poorly for the shorter sequences. It produced the lowest generalisation figure of the tests for length 6 sequences, generalising to only 83%. Its performance degraded further to only 74% of length 7 sequences. However, its performance gradually increased to nearly 82% as the sequence length increased to 9 symbols.

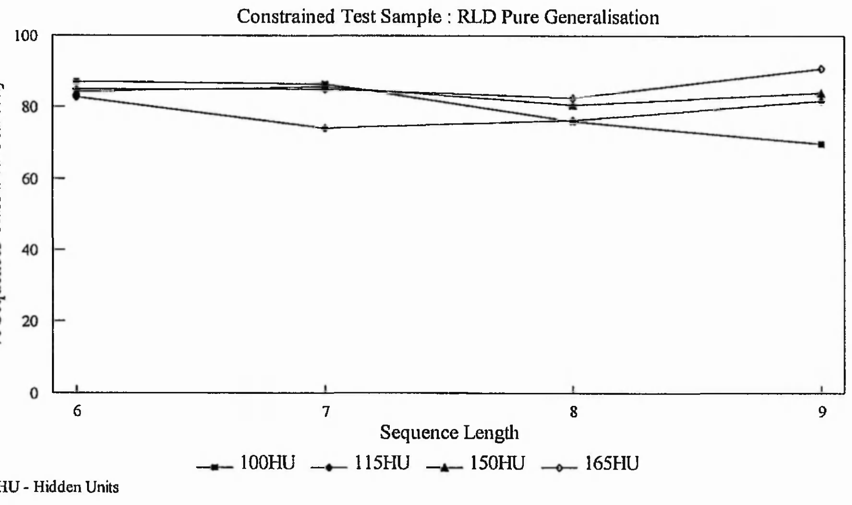


Figure 5.4 : Graph illustrating the pure generalisation performance of the TASRN networks for the constrained RLD test sample.

The TASRN with 165 hidden units again produced the highest levels of sequence generalisation overall. This configuration was able to achieve generalisations above 80% for all sequence lengths and correctly processed 91% of length 9 sequences i.e., 7% higher than any other TASRN configuration. The pure generalisation results for all TASRN configurations is illustrated in figure 5.4.

5.6. Left-to-Right Segmentation

The RLD's task is to find the beginning of the right-most syntactic phrase within a sentence, the LRD's task is therefore to find the corresponding end of the right-most syntactic phrase. The LRD sequences consist of the symbols processed by the RLD network (including a fixed number of the RLD's look-back symbols). The LRD network processes these sequences in reverse order i.e., it must start with a look-back symbol and process each input symbol to its right-hand side until it has encountered the input symbol denoting the end of the phrase and has also processed a fixed number of look-ahead symbols for contextual analysis. The LRD's task is the less difficult in the phrase segmenting stage as the majority of the symbols it will process will be phrasal symbols and contextual symbols without any of the irrelevant symbols that may be to the right of a phrase and that are presented to the RLD network. The complexity constraint applied to the training sample has further simplified the task as it reduces the overall sequence lengths to be processed. The process involved in training and testing the LRD network are similar to those for the RLD network, however there are certain distinct differences. This section investigates the optimum number of look-back and look-ahead symbols required for the LRD to resolve conflicting sequences that are detrimental to the parsing process. The training and test performance of the TASRN network configurations trained to perform the task is then fully described.

5.6.1. Optimising the Look-Back and Look-Ahead

Determining the optimum number of contextual symbols for the LRD network is a more complicated process than that for the RLD network as there are two variables rather than one. The RLD network only required look-back symbols, whereas the LRD network requires look-back and look-ahead symbols. The look-back symbols of the LRD sequences are those symbols immediately to the left of the phrase, and the symbols to the right of the phrase are look-ahead symbols. Although these overhead symbols are not part of the phrase, they are essential in the segmentation process as it provides the parser with 'a view' to the left and right of the phrase and enables the parser to consequently resolve phrasal ambiguity based upon its context. The number of look-back and look-ahead symbols used therefore greatly affects the performance of the parser. The selection criteria for deciding the optimum number of look-back and look-ahead symbols for the LRD module are : the number of detrimental conflicts affecting the training corpus and the number of training patterns and sequences after balancing.

Table 5.10 shows the number of detrimental LRD sequence conflicts in the training corpus using various look-back and look-ahead configurations. The maximum number of available look-back symbols is fixed at four input symbols as the LRD module always receives its input from the RLD module. The maximum number of look-ahead symbols is also fixed at four input symbols as using five look-ahead symbols provided no further improvement over four look-ahead symbols. If there are an insufficient number of input symbols available for look-ahead then the null symbol, ^, will be used to pad out the number of look-ahead symbols to the required length. It can be seen from table 5.10 that using look-back symbols is more effective than using look-ahead symbols to resolve phrasal ambiguity. For example, 23 detrimental conflicts occur when using the minimum of one look-back and one look-ahead symbol. This reduces by only 34.8% to 15 conflicts when increasing the number of look-ahead symbols to four. However, a 52.2% reduction is achieved via only increasing the number of look-backs to four input symbols.

In terms of conflict resolution alone, the optimum number of look-back and look-ahead symbols is 3 symbols for both look-back and look-ahead as increasing either look-back or look-ahead to four symbols results in no further conflict resolution. However, this configuration results in 38,318 training patterns and 4,508 training sequences after balancing. It was decided that the resulting training times would be impractical considering the resources available for training the LRD network.

Training Sentences Affected After Removing LRD Sequence Conflicts				
Look-Back Symbols	**Look-Ahead Symbols**	**Sequence Conflicts**	**Training Sentences Affected**	**% Training Corpus Affected**
4	1	11	12	1.8
4	2	10	10	1.5
4	3	5	5	0.8
4	4	5	5	0.8
3	1	11	12	1.8
3	2	10	10	1.5
3	3	5	5	0.8
3	4	5	5	0.8
2	1	14	15	2.3
2	2	13	13	2.0
2	3	8	8	1.2
2	4	8	8	1.2
1	1	23	26	4.0
1	2	20	22	3.4
1	3	15	17	2.6
1	4	15	17	2.6

Table 5.10 : Training Sentences affected after removing LRD sequence conflicts.

A good compromise was found to be 2 look-back symbols and 2 look-ahead symbols. Although 2% of the training corpus is affected this was deemed acceptable. This configuration results in a 31% reduction in training patterns and only a 9.8% reduction in training sequences (after balancing) than using a configuration of 3 look-back and 3 look-ahead symbols.

5.6.2. Training Experiments Performed

The LRD training data must be balanced before experiments can be performed. Table 5.11 shows the composition of the LRD training set before and after balancing. Sequences of length 5 will appear in the training set with the same frequency as all other training sequences, however, each length 5 sequence is unique and although coverage of these sequences is consequently wider than any other sequence length it envisaged that the LRD network will find it more difficult to learn length 5 sequences due to the lack of replication.

LRD Training Sets						
	Length 5 Sequences	**Length 6 Sequences**	**Length 7 Sequences**	**Length 8 Sequences**	**Total Sequences**	**Total Pattern s**
Raw Training Sequences	1,017	938	417	202	2,574	15,248
Balanced Training Sequences	1,017	1,017	1,017	1,017	4,068	26,442

Table 5.11 : The raw and balanced LRD Training sets.

As with the RLD network, Back-propagation with a momentum term of 0.9 is the learning algorithm used to train the TASRN networks in on-line mode for the LRD task. The number of input units and 'external' output units is the same as for the RLD network. Again, a bias unit of 1.0 is included in the input and hidden layer and hidden unit and output unit activation values are calculated using the standard sigmoid function. All training experiments use the balanced training set detailed in table 5.11 and are performed on SunSPARC 10/20 Servers or a P100MHZ Pentium Personal Computer (PC). The pattern-error sensitive learning rate is again used for all experiments. The time taken per epoch is again long at 1.2 hours. Experiments are therefore carried out to find the number of hidden units that will learn an acceptable number of sequences rather than to optimise the number of hidden units.

The resulting generalisation capability of the network will be relied upon to gain a larger coverage of the overall corpus.

Training Results for the LRD Network					
Hidden Units	**Connections**	**Epochs**	**RMS Error**	**% Patterns Learnt**	**% Sequences Learnt**
65	16,637	449	0.0753102	98.52	91.89
75	20,687	1,171	0.073062	98.96	94.35
110	38,012	731	0.057598	99.11	95.11
115	40,887	973	0.0646898	99.28	96.09

Table 5.12 : Results of the LRD training experiments performed.

Table 5.12 summarises the results of the training experiments performed using TASRN networks with a number of hidden unit configurations. The minimum number of hidden units required to gain acceptable training results for the LRD task was found to be 65 hidden units, 35 hidden units less than the minimum required for the RLD task thus contrasting the complexity between the two segmentation modules. A limit of 115 hidden units was imposed due to time and resource constraints. All configurations were able to learn above 91% of the training sequences. Surprisingly, each configuration learnt at least 97% of the fully unique length 5 sequences. However, all of the

TASRN configurations learnt less of the length 7 sequences than any other sequence length indicating that memory decay is a more dominant feature than the replication factor. The breakdown of training results shown in table 5.13 illustrate that increasing the number of hidden units results in the longer training sequences being learnt at the expense of the shorter training sequences. This is expected as the greater the number of hidden units the better the representation of the auto-associative output so the less the decay. If the auto-association were perfect then this would result in a perfect memory.

Breakdown of Training Sequences Learnt					
Hidden Units	**% Length 5 Sequences**	**% Length 6 Sequences**	**% Length 7 Sequences**	**% Length 8 Sequences**	**% Total Sequences**
65	97.54	82.01	92.92	95.08	91.89
75	96.95	87.51	93.90	99.02	94.35
110	97.74	89.18	94.00	99.51	95.11
115	96.76	91.84	95.77	100	96.09

Table 5.13 : A breakdown of the training results for the LRD network.

It can be seen from table 5.13 that all the network configurations have learnt less of the length 6 and length 7 sequences compared to the number of length 5 and 8 sequences learnt. This is a surprising contrast as all of length 5 sequences are unique and length 8 sequences contain the highest level of replication. However, it is also an indication that the uneven replication (and thus representation in the training data) of length 6 and 7 sequences reduced the ability of the TASRN to learn these sequences. After 381 epochs, the TASRN with 65 hidden units began to oscillate and training was terminated after 449 epochs. This configuration learnt nearly 98% of length 5 sequences at the expense of the longer sequences. The TASRN configuration with 75 hidden units managed to learn above 93% of the length 5, 7 and 8 sequences and 87% of length 6 sequences[22]. Increasing the hidden units to 110 resulted in a substantial improvement in sequences learnt. This TASRN configuration managed to learn at least 94% of length 5, 7 and 8 sequences. The TASRN configuration with 110 hidden units was unable to learn above 89.2% of length 6 sequences within the number of training epochs allowed.

Increasing the number of hidden units further to 115 units provided further improvements in sequences learnt. After 973 epochs this configuration had learnt more than 90% of *all* training sequences and therefore learnt more of the training sequences than any of the other TASRN configurations. It learnt all of the length 8 sequences and nearly 97% of the unique length 5 training sequences and managed to learn nearly 92% of the length 6 sequences. This configuration enabled the majority of all training sequences to be learnt.

[22] This network was trained for the longest period on a Pentium P100 MHZ PC for approximately 2 months.

The resulting generalisation performance of each of the TASRN configurations will determine the amount of language coverage achieved using the complexity constraints defined.

5.6.3. Test Results

As with the test experiments for the RLD network, the LRD TASRN configurations will be tested on the constrained test sample. Each testing phase will test the TASRN configuration's natural generalisation ability and pure generalisation ability.

5.6.3.1. Constrained Test Sample

The composition of the LRD constrained test sample is shown in table 5.14. It can be seen that the test sample consists of nearly 84% unique test sequences (i.e., sequences that do not appear in the training sample) for the LRD networks. All replicated test sequences are removed. As with the training sample, the constrained test sample contains more length 5 and 6 sequences than any other sequence length. Similarly in the training data, length 7 and 8 sequences occur less frequently than any other sequence length.

As all TASRN configurations learnt less of the length 6 training sequences than any other sequence length, significant generalisations to length 6 sequences are therefore not expected. However, high levels of generalisation are expected for length 5 sequences as all the training configurations were able to learn above 96%.

Composition of the LRD Constrained Test Sample						
	Length 5 Sequences	**Length 6 Sequences**	**Length 7 Sequences**	**Length 8 Sequences**	**Total Sequences**	**Total Pattern s**
Natural	1,087	979	443	233	2,742	16,274
Pure	881	832	387	192	2,292	13,642
% Unique	81.05	84.98	87.36	82.40	83.59	83.83

Table 5.14 : Composition of the LRD Constrained Test Sample.

Natural Generalisation

Table 5.15 summarises the natural generalisation results obtained from the four TASRN network configurations. As expected, all the TASRN configurations generalised to above 90% of the length 5 and 8 sequences and the worst performance for all configurations was on length 6 sequences. None of the TASRN configurations were able to generalise to above 88% of the length 6 sequences. TASRN configurations of 75 and 110 hidden units were the only configurations able to generalise to above 90% of length 7 sequences. It is surprising that the configuration with 115 hidden units generalised to only 87% of the length 7 sequences, as it had learnt above 95% of length 7 sequences (more than any of the other TASRN configurations) during the training experiments. Even though the configuration of 115 hidden units learnt more of the training data than any other configuration, it was consistently outperformed by the TASRN configurations with 75 and 110 hidden units throughout the natural generalisation tests. The configuration with 110 hidden units provided the highest overall natural generalisation results, generalising to 91.8% of all test sequences.

Natural Generalisation Results for the Constrained Test Sample						
Hidden Units	**% Length 5 Sequences**	**% Length 6 Sequences**	**% Length 7 Sequences**	**% Length 8 Sequences**	**% Patterns Correct**	**% Sequences Correct**
65	93.38	83.35	86.23	93.13	97.62	88.62
75	96.04	84.68	90.29	93.13	98.09	90.81
110	96.96	85.19	90.52	97.42	98.33	91.76
115	95.31	87.95	86.91	91.42	98.14	90.99

Table 5.15 : Natural generalisation results for the LRD constrained test sample.

Pure Generalisation

Table 5.16 summarises the pure generalisation results obtained from the four TASRN network configurations. As expected, the generalisation figures are similar to those obtained during the tests for natural generalisation. This is due to 84% of the constrained test sample consisting of pure LRD test sequences.

All the TASRN configurations were able to generalise to above 92% of the length 5 pure test sequences. The configuration with 110 hidden units outperformed other configurations, generalising to 97% of these sequences. This is expected as it had learnt more length 5 sequences than any other configuration during the training phase and consequently gained a wider coverage of these sequences. The TASRN network with 65 hidden units revealed its limitations, generalising to 92% of length 5 sequences, the lowest of all the TASRN configurations, even though it learnt nearly 98% of the length 5 training sequences during the training phase. It also produced the lowest generalisation figures for length 6 and 7 sequences, generalising to 81% and 85% respectively. The TASRN configuration with 115 hidden units outperformed all configurations for length 6 sequences, generalising to 86%. This is expected as it had learnt more length 6 sequences than any other configuration during the training phase and consequently gained a wider coverage of these sequences. However, its performance degrades rapidly after length 6 sequences with respect to configurations of 75 and 110 hidden units. It manages to generalise to only 85% of the length 7 sequences and unlike the other configurations it was unable to generalise to above 90% of the length 8 sequences.

Pure Generalisation Results for the Constrained Test Sample						
Hidden Units	**% Length 5 Sequences**	**% Length 6 Sequences**	**% Length 7 Sequences**	**% Length 8 Sequences**	**% Patterns Correct**	**% Sequences Correct**
65	92.17	80.89	85.01	91.67	97.25	86.82
75	95.12	82.33	89.15	91.67	97.75	89.18
110	96.59	83.05	89.15	96.88	98.07	90.45
115	94.21	86.18	85.27	89.58	97.82	89.40

Table 5.16 : Pure generalisation results for the constrained LRD test sample.

The TASRN with 75 hidden units matched the configuration with 110 hidden units for length 7 sequences, generalising to 89%, the highest in the pure generalisation tests. It also illustrates the contrast between natural and pure generalisation, as the configuration with 110 hidden units had learnt more length 7 sequences as reflected by the natural generalisation figures. However, the natural generalisation figures do not reflect this for length 8 sequences between configurations with 65 and 75 hidden units. Although the configuration with 75 hidden units had learnt nearly 4% more of the length 8 sequences than the configuration with 65 hidden units, it correctly generalises to exactly the same natural and pure test sequences.

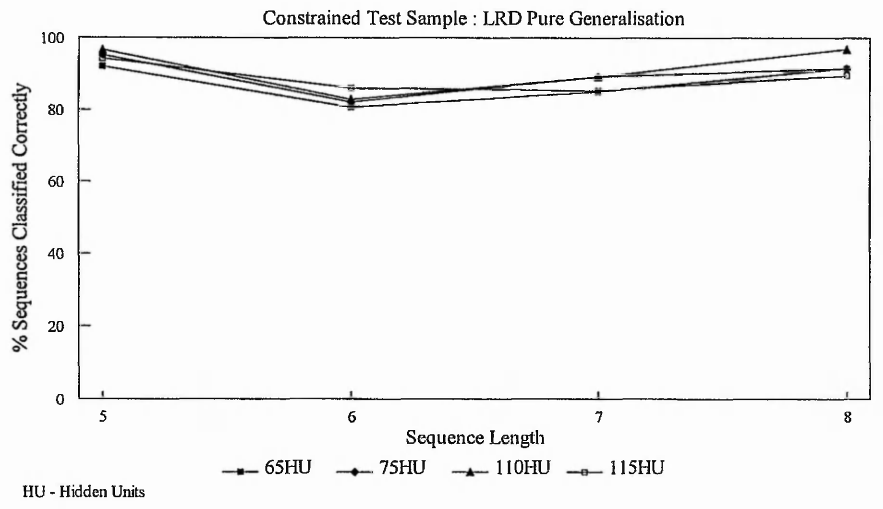


Figure 5.5 : Graph illustrating the pure generalisation performance of the TASRN networks for the constrained LRD test sample.

This is obviously due to the natural test data coincidentally containing sequences that both of the configurations had learnt. The configuration with 110 hidden units marginally outperformed all other configurations in both the natural and pure generalisation tests. The pure generalisation results for all TASRN configurations is illustrated in figure 5.5.

5.7. Summary

This chapter has presented a detailed description of the phrase segmentation modules of the corpus-based parsing model. The TASRN recurrent connectionist network, an extension of Elman's SRN network, is the architecture identified as adequate for the RLD and LRD segmentation modules. A semi-linear output function has also been defined to associate output targets with RLD and LRD sequences.

The optimum number of look-back symbols for RLD module is four input symbols. The TASRN hidden unit configuration that provides optimal performance for the RLD task is 165 hidden units, providing pure test sequence generalisations of 85% for the constrained test sample. The optimum number of look-back and look-ahead symbols for LRD module is two look-back symbols and two look-ahead symbols. The TASRN hidden unit configuration that provides optimal performance for the LRD task is 110 hidden units, providing impressive pure test sequence generalisations of 91% for the constrained test sample.

Chapter 6 presents the phrase recognition module that recognises the phrase extracted from the sentence by the RLD and LRD connectionist networks.

Chapter 6

Phrase Recognition

6.1. *Introduction*

This chapter provides a detailed description of the phrase recognition aspects of the corpus-based parsing model. As illustrated in Chapter 3, significant complexities are encountered when attempting to perform the tasks of phrase delimiting, recognition and validation using a single connectionist network. This has motivated the decision to simplify the recognition task and employ a single connectionist network to act purely as a phrase structure recogniser that has no notion of valid or invalid phrases. The phrase recognition module will therefore always attempt to recognise the phrase extracted by the phrase segmentation modules and thus aids the simplification of the overall parsing task. The phrase recognition technique developed here is a method of recognising syntactic phrases from sentences without the need for predefined grammar rules to help perform the task

Section 6.2 begins with a brief overview of the phrase recognition technique and explains how phrasal ambiguity is resolved in the parser by using position invariant phrase recognition and optimising the number of look-back and look-ahead symbols. Section 6.3 describes the feed-forward MLP architecture that constitutes the Recogniser network. Section 6.4 details the post-processing performed on the output activations of the Recogniser network. Section 6.5 presents the training and test performances of the Recogniser network. Finally, Section 6.6 concludes with a brief summary of the chapter.

6.2. *Overview of Phrase Recognition*

Phrase recognition is the process of associating a single constituent tag with a collection of syntactic and constituent tags (i.e., words and phrases) that constitute a syntactic phrase. This single constituent tag represents the phrase group to which the associated tags belong. In the context of a parse tree, the collection of tags that constitute the syntactic phrase is dominated by the single constituent tag. In the context of the parsing process, this single constituent tag represents high-level linguistic information and is used to reference the phrasal symbols which it dominates. Phrase recognition clearly plays a fundamental role in the parsing model and is essentially performing the phrase reduction step of the shift-reduce parsing strategy.

Phrase recognition is performed within the parser by a single feed-forward MLP architecture called the Recogniser network. The Recogniser network will receive its input symbols from the REC-stack which contains the symbols extracted from the input buffer by the phrase segmentation module. These symbols may be a mixture of word tags and constituent tags. The position of the symbols on the REC-stack will determine whether they are look-back symbols, phrasal symbols or look-ahead symbols. As the REC-stack is a FIFO symbolic stack structure, the symbol at the top of the stack is the look-ahead symbol, the bottom four symbols of the stack are the look-back symbols and the remaining symbols are the phrasal symbols. During the recognition process, these symbols are popped off the REC-stack and mapped onto the input layer of the Recogniser network. The Recogniser network then produces a set of corresponding output activations on its output layer that represent the single constituent tag for the given input phrase. However, further post-processing is performed on the output activations to match it directly to the nearest phrase in input vector space. Section 6.2.1 provides further detail regarding the mapping of the REC-stack symbols onto the Recogniser network's input layer to allow position invariant phrase recognition.

6.2.1. Position Invariant Phrase Recognition and Disambiguation

Deterministic parsers avoid back-tracking and resolve phrasal ambiguity by using look-ahead symbols to reduce the search space via disregarding invalid state space configurations as more symbols are processed as look-ahead. Although Marcus [51] stated that a maximum look-ahead of five words were required to resolve lexical ambiguity, there is no predetermined limit on the number of look-back and look-ahead symbols required to resolve phrasal ambiguity in non-globally ambiguous sentences. However, it is clear that context-sensitive phrase recognition is essential if phrasal ambiguity is to be resolved and back-tracking is to be avoided.

The corpus-based parsing model has no access to semantic or discourse information and relies purely on deterministic processing to resolve any phrasal ambiguity it encounters. It must use the look-back and look-ahead symbols provided by the RLD and LRD segmentation modules. The optimum number of look-back symbols and look-ahead symbols required for the recognition task must be established empirically and the selection is based upon the number of ambiguities resolved within the Recogniser network's training sample. However, before the optimum number of contextual symbols can be determined, a limit must be imposed on the phrase length. This will result in shorter phrases being padded out with the appropriate number of null symbols until it is of the required length. *The maximum phrase length within the entire LPC corpus was found to be 10 symbols. The optimum number of look-back symbols is 4 symbols and the optimum number of look-ahead symbols is 1 symbol. The size of the Recogniser network's input layer is therefore 15 symbols at 61 bits per symbol resulting in a total of 915 input units.* This look-back/look-ahead configuration resolved all phrasal ambiguities in the training sample. As proven in Chapter 5, look-back symbols are more important than look-ahead symbols when recognising phrases in a right-associative manner. Reducing the number of look-back symbols available to the recognition module to 3 symbols resulted in two ambiguous phrases. Increasing the number of look-ahead symbols to 2 resulted in no change.

The recognition task is consequently made simpler in that the phrase will always be in the same position and of the same size within the input layer and the boundaries between the look-back and look-ahead symbols is distinct. This is known as *position invariant phrase recognition.*

If the look-back or phrase is shorter than the fixed limit imposed then it must be appropriately padded out with null symbols to provide position invariance. If the length of the input phrase exceeded the length of the input layer then it could not be processed by the network. Consider the following parsed sentence :

*[S._. [N Cormack_**NP** N][V was_**BEDZ** V][J particularly_**RB** sound_**JJ** J][P at_**IN** [N full-back_**NN** N]P] ._. S]*

Assume that the input layer is fixed at 15 input symbols consisting of 4 look-back symbols, 10 phrasal symbols and 1 look-ahead symbol. Also, presuppose that the parser has successfully reduced *full-back_**NN*** to ***N*** and at_***IN N*** to ***P***. If no fixed limit is imposed on the phrase length and consequently no null padding is used then during training the following training pair would be presented to the Recogniser network to represent the next valid reduction :

Input Symbols : . *NP BEDZ RB JJ P* ^^^^^^^^^

Desired Output : *J*

This complicates the recognition task for the Recogniser network in that the network will find it difficult to learn associations between the output symbol, ***J***, and the input symbols ***RB*** and ***JJ*** as there is no indication within the pattern as to which symbols constitute the phrase, the look-back and the look-ahead. This problem is easily resolved by viewing the input to the Recogniser network as having three fields of fixed length : a look-back field of 4 symbols, a phrase field of 10 symbols and a look-ahead field of 1 symbol. When the length of the look-back, phrase or look-ahead symbols is less than the field length then null symbols are added to increase the number of symbols to the desired length. The resulting training pair is :

Input Symbols : . *NP BEDZ* ^ *RB JJ* ^^^^^^^^ *P*

Desired Output : *J*

This simplifies the learning task as each field within an input pattern is of fixed length and its position is static. It is envisaged that this will aid learning in that the Recogniser network will be able to easily locate the phrase within the input pattern and distinguish it from contextual symbols in order to make its output classification. However, the Recogniser network will be sensitive to position and this may affect generalisation.

6.3. *MLP Architecture and Input Structure*

As the complexity of the recognition task has been greatly reduced from that defined in Chapter 3, a feed-forward MLP architecture with two weight layers is adequate for the architecture of the Recogniser network. The architecture trained with the Back-propagation learning algorithm is adequate for both representing and learning the task. The Recogniser network will simply be expected to learn to associate the input phrase with its corresponding constituent tag during its learning phase, and to perform a 'nearest match' computation to find the most probable constituent tag for the input phrase during testing. The MLP architecture for the Recogniser network is illustrated in figure 6.1.

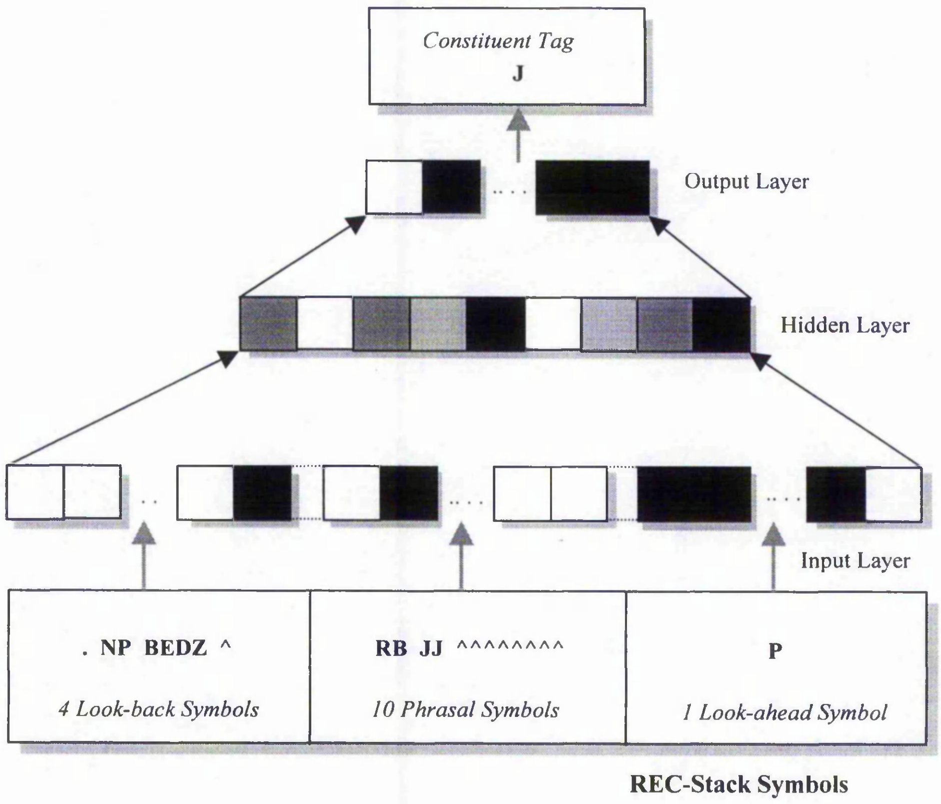


Figure 6.1 : The MLP architecture for the Recogniser network.

It can be seen from figure 6.1 that the contents of the REC-stack are mapped onto the input layer of the Recogniser network. The input layer consists of 915 input units, where each input unit represents a bit. The bit representation of each symbol is extracted from the tag database and then assigned to the appropriate units of the input layer. There are 61 bits per symbol[23] and 15 input symbols hence the total of 915. A weighted sum between the input units and the associated hidden weights will be calculated to activate the hidden units. The hidden layer contains a fixed number of hidden units that is empirically established in Section 6.5. The hidden unit responses will represent reduced descriptions of the linguistic features present across the input units. A weighted sum between the hidden units and the associated output weights will be calculated to activate the output units. The output layer contains 61 output units to represent the bit representation of the constituent tag associated with the input symbols encoded in the input layer. If the Recogniser network is in a learning phase then the output unit activation values are matched against the bit representation of the desired output symbol and Back-propagation is used to perform error-correction throughout the network. However, if the Recogniser network is in a testing phase then the output unit activations are processed further before the appropriate constituent symbol is extracted from the tag database.

[23] For a detailed explanation of the bit representation of a word tag or constituent tag see Chapter 4.

6.4. Output Representation and Nearest Match Classifications

Each symbol is presented to the Recogniser network as a bit vector consisting of 61 bits. The output response vector is the resulting 61 output unit activation values in the output layer computed in response to some input vector on the input layer. The standard sigmoid function is used to calculate each output unit activation value and the result is a continuous value between 0.0 and 1.0. During the training phase, the activation value for a particular output unit may be incorrect and the learning algorithm will perform weight adjustments to correct this and eventually the output response vector will be closer in output vector space to the desired output bit vector. However, during testing there is obviously no desired output bit vector and the Euclidean distance measure is used to find the symbol with the nearest matching bit vector to that produced on the Recogniser network's output layer.

As the output unit activation values are continuous they are first converted to a discrete value before the distance measure is calculated. A threshold value of 0.5 is used as the basis to convert a continuous value to a discrete value. If the continuous value is above 0.5 then it is given the value of 1, else if it is 0.5 or below it is given the value of 0. This is performed for each output unit activation. The result is an output bit vector representation of the output response vector. This is compared with the set of constituent tag representations and the one with the closest match according to its Euclidean distance from the output vector is taken as the network output i.e., the reduction.

6.5. Training Experiments Performed

The training sample for the Recogniser network (see table 4.7) consists of 2,588 training patterns and contains 72 unique phrase groups after all natural replication has been removed. Table 6.1 shows the main composition of the Recogniser network's training set. It can be seen that there is a considerable bias in the training sample towards the noun phrase group. There are 825 unique noun phrases in the training data representing nearly 33% of the entire training set. The verb phrase group represents 17% of the training data with 444 unique types of verb phrases and the preposition phrase group represents nearly half that at 8% with 216 unique preposition phrases. As there is one per sentence, unique higher level sentence structures, represented by the 'sentence' phrase group, occur considerably less in the training sample than the noun, verb and preposition phrase groups with 174 unique types of sentence. The phrases that complement noun and verb phrases also occur frequently in the training set. Adverbial phrases and adjective phrases represent approximately 4% of the training sample. The remaining phrases that naturally occur in the training sample are derivatives of these phrase groups and the complete composition of the Recogniser network's training sample can be seen in Appendix F. The main phrasal groups that naturally occur within the training data are therefore N, V, P, S, R and J.

Back-propagation with a momentum term of 0.9 is the learning algorithm used to train the Recogniser network in on-line mode. A bias unit of 1.0 is included in the input and hidden layer. The CNAPS parallel computer is used to train the Recogniser network. The CNAPS system is dedicated to parallel processing in a way that is suitable for connectionist networks, and provides major speed improvements. However, the constraints imposed by the system reduce the flexibility of the training experiments that can be performed. For example, the pattern-error sensitive learning rate could not be used and there are limitations on the training set size. Therefore neither training set balancing for pattern-error sensitive learning rate were used.

Main Composition of The Recogniser Network's Training Sample			
Phrase Group	**Description**	**Occurrences**	**% of Training Data**
N	Noun phrase.	825	32.88
V	Finite verb phrase.	444	17.16
P	Prepositional phrase.	216	8.35
S	Sentence.	174	6.72
R	Adverbial phrase.	111	4.29
J	Adjective phrase.	90	3.48
Po	Prepositional phrase beginning with 'of'.	81	3.13
Vi	Non-finite verb phrase.	63	2.43
Na	Noun phrase marked as the subject of the verb.	63	2.43
Ti	To-infinitive clause.	59	2.28
Nq	Wh-noun phrase.	32	1.24

Table 6.1 : The predominant phrase groups in the Recogniser network's training sample.

Initial MLP configurations were 40 hidden units and 45 hidden units. After 5 trial runs both configurations were unable to learn the training data after 1,000 epochs. Increasing the number of hidden units to 50 significantly improved learning. This configuration learnt all training patterns after 800 epochs. Table 6.2 details this result. It confirms that by reducing the complexity of the recognition task an MLP architecture is adequate to *fully* learn and represent the task. Although time constraints limited the ability to optimise learning with respect to generalisation it is believed that using 50 hidden units is close to the minimum configuration as the configuration with 45 hidden units was unable to learn the task after five trial runs. The resulting generalisation capability of the network will be relied upon to gain a larger coverage of the overall corpus and of the English language in general.

Training Results for the Recogniser Network				
Hidden Units	**Connections**	**Epochs**	**RMS Error**	**% Patterns Learnt**
50	48,911	800	0.005	100

Table 6.2 : Results of the training experiment with 50 hidden units for the Recogniser network.

6.6. Test Results

As with the phrase segmentation networks, the Recogniser network will be tested on the constrained test sample to assess the natural generalisation ability and pure generalisation ability of the MLP configuration with 50 hidden units.

6.6.1. Constrained Test Sample

Table 6.3 shows the number of natural and pure test patterns present in the constrained test sample after natural occurring pattern replication has been removed. The constrained sample consists of 2,433 pure test patterns which represents nearly 88% of the sample.

Constrained Test Sample		
Test Patterns for Natural Generalisation	**Test Patterns for Pure Generalisation**	**% Unique**
2,765	2,433	87.99

Table 6.3 : The amount of natural and pure test patterns in the constrained test sample (naturally occurring pattern replication has been removed.

The basic phrasal composition of the constrained test sample is shown in table 6.4. It can be seen that above 90% of the noun (*N*), preposition (*P*) and adjective (*J*) phrases in the test sample consist of pure test patterns. Above 80% of the verb (*V*) phrases and adverbial (*R*) phrases are also pure test patterns. However, there is a considerable overlap between the sentence level (*S*) phrases found in the training and test sample, as only 58% of the sentence level phrases constitute pure test data.

For both natural and pure generalisation tests, the RMS error value for each test set is based upon the raw output activations of the Recogniser network. The nearest-match classification defined in Section 6.4 will then be used to increase the accuracy of the network and consequently improve the level of generalisation. A summary of the natural and pure generalisation results for the constrained test sample is shown in table 6.5.

Major Phrases In The Constrained Test Set							
	N	**V**	**P**	**S**	**R**	**J**	**Total Major Phrases**
Natural Test Patterns	872	467	246	200	126	81	1,992
Pure Test Patterns	793	389	234	115	110	76	1,717
% Unique	90.94	83.29	95.12	57.50	87.30	93.83	86.19

Table 6.4 : The basic phrasal composition of the constrained test sample. All replicated patterns have been removed.

It can be seen from table 6.5 that using the nearest match computation significantly improves the levels of generalisation. The generalisation rate for the natural test patterns increased by 9.8% to 89%. The generalisation rate for the pure test patterns increased by 11.2% to 87.6%.

Generalisation Results for The Constrained Test Sample					
	Number of Patterns	**RMS Error**	**Correct Classifications**	**Correct Classifications after 'Nearest Match' Computation**	**% Generalisation**
Natural Test Patterns	2,765	0.0683	2,190	2,461	89.01
Pure Test Patterns	2,433	0.0726	1,861	2,132	87.63

Table 6.5 : The natural and pure generalisation results for the constrained test sample.

The incorrect classifications produced by the Recogniser network, after the nearest match computation has been applied, is an indication of the recognition errors that will be made during parsing.

After analysing the errors made by the Recogniser network, it is clear that in some cases the network over-generalises. For example, as context becomes unfamiliar the Recogniser network prefers to classify specific verb phrase groups as a general verb phrase group i.e., produces *V* rather than *Vi* or *Vg*. Understandably, the Recogniser network is unable to recognise a phrase irrespective of its position within the input sentence. The look-back, phrasal and look-ahead symbols are of fixed position within the training patterns and an inherent limitation of position invariant phrase recognition is that the connectionist network learns to become sensitive to the position of the phrase. This accounts for some of the recognition errors made by the Recogniser network.

Table 6.6 shows a sample of the Recogniser network's training data that teaches it to associate the word *am* (progressive tense), represented by the word tag *BEM*, with the finite verb phrase group, *V*. It can be seen from table 6.6 that the Recogniser network learns to associate *V* with *BEM*, and *V* with *BEM VBG* only when *BEM* is preceded by the word *I*, represented by the word tag *PP1A*. However, during the pure generalisation tests, the Recogniser network incorrectly classifies the following input pattern :

. PP1A BEM JJ PP1AS ^^^^^^^^^ V

The desired association is to associate the word tag for a personal pronoun, *PP1AS*, with the constituent tag for a noun phrase group, *N*. However, the actual association made by the Recogniser network is *V*. It is therefore natural to postulate that the Recogniser network is using context in its computation i.e., it has learnt a complex rule involving context rather than learning a simple rule to associate *V* with *BEM*, and *V* with *BEM VBG* only when *BEM* is preceded by *PP1A*. The Recogniser network makes similar errors throughout the test experiments.

Sample of Training Patterns to Associate *BEM* with a *V* Reduction		
Look-back Symbols	**Phrasal Symbols**	**Look-ahead Symbol**
PP2 VB WP PP1A	BEM ^^^^ ^^^^^	.
. PP1A ^^	BEM ^^^^ ^^^^^	N
. PP1A ^^	BEM VBG ^^^^^^^ ^	Fa
. PP1A ^^	BEM VBG ^^^^^^^ ^	.

Table 6.6 : A sample of training patterns used to enable the Recogniser network to associate the word tag *BEM* with the constituent tag *V*.

Another interesting example of the network learning complex rules involving context, is when the network attempts to associate a present participle of a verb, represented by *VBG*, with the constituent tag for a non-finite verb phrase, represented by *Vg*. It is essential to first assess what the network has learnt with respect to the relationship between *VBG* and *Vg*.

Table 6.7 shows a sample of the training patterns used to teach the Recogniser network to associate *VBG* with *Vg* and *VBG* with *V*. It can be seen from table 6.7 that the Recogniser network learns to associate *VBG* with *V* only when *VBG* is preceded by the word tag *BEDZ*, within a phrase. The word tag, *BEDZ*, represents the word *was*. If the *VBG* is preceded with some other context then the Recogniser network will associate it with *Vg*.

Although the Recogniser network has successfully learnt all of the training data, table 6.8 shows the incorrect associations made by the Recogniser network when processing the pure test sample. Table 6.8 shows that the differing context is causing the Recogniser network to incorrectly associate *VBG* with *V* rather than *Vg*. The lack of coverage of *Vg* phrases in the training data (only 26 instances in the entire set) could also be responsible for the misclassifications.

Sample of Training Patterns to Associate *VBG* with a *Vg* Reduction		
Look-back Symbols	**Phrasal Symbols**	**Look-ahead Symbol**
RB AT JJ NN	VBG ^^^^^^^^^	Ti
. ATI NPT RB	VBG ^^^^^^^^^	P
. ^^^	VBG ^^^^^^^^^	N
INO JNP NN INO	VBG ^^^^^^^^^	N
VBN DTI NN INO	VBG ^^^^^^^^^	N
Sample of Training Patterns to Associate *VBG* with a *V* Reduction		
NP ATI NNS WP	BEDZ VBG ^^^^^^^^	N
. PP3A ^^	BEDZ VBG ^^^^^^^^	Ti
. NP ^^	BEDZ VBG ^^^^^^^^	N
RN JJ CS NP	BEDZ VBG ^^^^^^^^	.
. PP1A ^^	BEM VBG ^^^^^^^^	Fa

Table 6.7 : Training patterns used to enable the Recogniser network to associate the word tag *VBG* with the constituent tag *Vg* and *VBG* with *V*.

The errors shown in table 6.8 can be again attributed to the lack of coverage and the complex rules learnt by the network. Even though the problem is made simpler by fixing the position of the phrase, complications will always arise when unfamiliar context is presented with the phrase. In all except for the first test pattern, the look-back symbols contain symbols that can be associated with the finite verb phrase group.

The Recogniser Network Associating *VBG* with *V* Rather Than *Vg*		
Look-back Symbols	**Phrasal Symbols**	**Look-ahead Symbol**
NN CC NN NNS	VBG ^^^^^^^^^	P
QL RB JJ CS	VBG ^^^^^^^^^	N
BEZ VBN INO NNS	VBG ^^^^^^^^^	N
CC NN CC NS	VBG ^^^^^^^^^	N
TO VB ATI NNS	VBG ^^^^^^^^^	N
. RN PP1AS BER	VBG ^^^^^^^^^	N

Table 6.8 : Incorrect associations made by the Recogniser network between *VBG* and *V*.

Table 6.9 further illustrates the problem as the Recogniser network incorrectly associates the word tag for the past participle of a verb, *VBN*, with *Vg*. The desired association is with the constituent tag for the appropriate derivative of the non-finite verb phrase group, *Vn*. The network also similarly confuses finite nominal clauses (*Fn*) with high-level sentence structures (*S*), noun phrases with adjective phrases and compound phrases with their non-compound equivalent. These minor errors are not considered detrimental to the parsing task as the network is able to associate phrases with the correct major phrase group. The Recogniser network appears to be over generalising within phrasal groups and that the granularity of the recognition is reduced when familiar phrases are presented in different contexts.

The Recogniser Network Associating *VBN* With *Vg* Rather Than *Vn*				
Look-back Symbols	**Phrasal Symbols**	**Look-ahead Symbol**	**Desired Association**	**Actual Association**
PP3A BEDZ QL RB	VBN ^^^^^^^^^	.	Vn	Vg
VB IN JJ NNS	TO BE VBN ^^^^^^^	Fn	Vi	Vg
. HV XNOT PP2	VBN ^^^^^^^^^	.	Vr	Vg

Table 6.9 : A sample of association errors made by the Recogniser network between *VBN* and *Vg*.

It is clear that overall the Recogniser network is computationally adequate for the phrase recognition module producing an impressive generalisation rate of nearly 88% for the pure test data of the constrained test sample.

6.7. Summary

This chapter has presented a detailed description of the phrase recognition module of the corpus-based parsing model. The recognition task has been significantly simplified from that in Chapter 3. Consequently, a feed-forward MLP network trained with the Back-propagation learning algorithm proved computationally adequate for the role of the Recogniser network within the parsing framework.

The optimum number of contextual symbols for Recogniser network is four look-back symbols and one look-ahead symbol. The hidden unit configuration that provides optimal performance for the Recogniser network's task is 50 hidden units, providing impressive pure test pattern generalisation results of 88% for the constrained test sample.

The training and testing of the individual connectionist modules of the parser is now complete and Chapter 7 presents the sentence level test results.

Chapter 7

Sentence Level Test Results

7.1. Introduction

Chapters 5 and 6 presented test results for the individual connectionist modules and thus provided the testing of each unit of the parser. This chapter provides the test results of the parser's performance on sentences belonging to the training and test samples. It identifies the parser's behavioural characteristics and structural preferences by analysing the parser's performance on the constrained test sample. The intention is to demonstrate the range of language the parser is capable of parsing. Section 7.2 presents a brief evaluation of the sentence level results for the training sample. Section 7.3 provides a detailed evaluation of the parser's performance on sentences from the constrained test sample to assess its ability to generalise to sentences of the same structural complexity as the training sample. Artificial sentences are also created to test the parser with sentence structures that contain specific syntactic constructions such as coordination and centre-embedding. Section 7.4 provides a comparison of the parser's performance with other LOB parsers to further assess its adequacy. Finally, Section 7.5 concludes with a summary of the chapter.

7.2. Evaluation on The Training Sample

Table 7.1 shows the parser's performance on the sentences extracted from the training sample. It can be seen that of the 654 training sentences, the parser successfully parsed 96.9% of them and managed to match 62.2% of the 654 sentences with the equivalent LPC parse. However, the parser failed to process 3.1% of the training sentences.

The parser failed on 20 training sentences and the critical errors made by the parser can be attributed to unlearnt RLD and LRD training sequences. After analysing these sentences, it is clear that the phrase segmentation modules were unable to learn the sequences from the longer sentences of the training sample within the fixed training time constraints. The minimum sentence length of the failed parses is 8 words, the maximum is 19 words and the average sentence length is 12 words. Each module was tested in isolation to locate the errors i.e., the sentences were decomposed into RLD and LRD sequences. The RLD network is responsible for the majority of the errors, producing a total of 29 sequence errors across 18 of these training sentences. The LRD produced 15 sequence errors across 12 sentences. The full sentence and output of the parser for each failed parse can be seen in Appendix I.

Sentence Level Test Results for The Training Sample			
Training Sentences	**% Successful Parses**	**% Failed Parses**	**% Matching LPC Parses**
654	96.9	3.1	62.2

Table 7.1 : The sentence level results for the training sample.

In total, the RLD network misclassified 316 of its 4,060 training sequences (7.8%) and the 29 sequence errors made by the RLD network during the parse failures account for 9.2% of the 316 misclassifications. The LRD network misclassified 199 (4.9%) of its 4,068 training sequences and the 15 sequence errors made by the LRD network during the parse failures account for 7.5% of the 199 sequence errors made during the training process. The LRD failed largely on finding the end boundary of the noun phrases and preposition phrases.

The major limitation of the connectionist modules failing is that it has a knock-on effect throughout the parsing process and there is no recovery mechanism. For example, if the RLD network prematurely activates or fails to activate when processing a sentence it will either group the wrong input symbols or force parsing to terminate. This results in invalid symbol combinations being passed onto the LRD network. The LRD network will then have to generalise to find a corresponding end boundary to the phrase. This extracted 'phrase' is then passed to the Recogniser network. Consequently, this changes the whole state of the parse and is detrimental to the parser's performance. However, a 3.1% failure rate is low and demonstrates the ability of the connectionist modules to generalise to compensate for failures made by the other networks.

Although, the parser successfully parsed 96.9% of the 654 training sentences, it only managed to successfully match 407 sentences (62.2%) with its desired (or *target*) parse in the LPC. The remaining 227 (34.7%) parses did not match the target parse due to unlearnt RLD and/or LRD sequences and the removal of conflicting LRD sequences. However, the LRD conflicts represent attachment ambiguity in the LPC corpus itself, and the decision to remove them was essential for the modules to train and for the parser to settle on one parse. It is envisaged that increasing the training times will enable the RLD and LRD networks to learn all of the training data and reduce the magnitude of errors made by the parser.

7.3. Evaluating The Parser's Behaviour Using The Constrained Test Sample

Associated with each sentence in the constrained test sample is its labelled bracketed parse found in the LPC. This associated LPC parse is termed its *target parse*. The output of the parser for each test sentence is thus compared with its target parse. The intention is to assess whether the parser has correctly learnt to parse sentences according to the linguistic constraints the connectionist modules deduced from the LPC sentences during the training phase. As each module was trained independently it will also be interesting to assess how the integration of these modules affect the parser's behaviour.

Table 7.2 shows the parser's performance on the sentences extracted from the constrained test sample. Although the match rate is lower than the training data, the results are comparable and this indicates that the training data was fairly representative.

Sentence Level Test Results for The Constrained Test Sample			
Test Sentences	**% Successful Parses**	**% Failed Parses**	**% Matching LPC Parses**
687	93.4	6.6	49.0

Table 7.2 : The sentence level results for the constrained test sample.

The parser failed to parse 45 sentences and these critical errors can be mainly attributed to the incorrect *pure generalisations* made by the RLD network and Recogniser network as only two unlearnt training sequences were responsible for the parse failures made. The minimum length of these 45 sentences is 6 words, the average is 12 words and the maximum is 24 words and again the parser is failing on complex sentences commonly consisting of above 12 words. The full sentence and output of the parser for a sample of 20 failed parses can be seen in Appendix J. Even though the Recogniser network has learnt all of its training data, it was responsible for a large number of errors during the parse failures. For example, the Recogniser network contributes to 13 of the 20 parse failures shown in Appendix J. Clearly, the Recogniser network is sensitive to position and context, therefore these generalisation failures represent a limitation of the phrase invariant recognition technique. The network's coverage of phrases is relatively low (the training sample consisted of only 2,588 phrase patterns) and the 87.6% pure generalisation rate on the overall constrained test sample reflects this (see Chapter 6). Although the number of parse failures is relatively low, it is envisaged that the magnitude of parse errors can be significantly reduced by allowing the RLD network to fully learn its training data and increasing the coverage of phrases in the Recogniser networks training set. The position invariant recognition technique is also discussed further in Chapter 8.

The parser successfully matched 337 of the 687 test sentences i.e., 49%. A subset of 10 matching parses is used for the purpose of this assessment. However, a further subset of 50 matching parses is provided in Appendix K for the interested reader.

The parser produced alternative linguistic structures for 305 of the 687 test sentences i.e., 44.4%. Although, the majority of the linguistic constraints were in the training data in some form, the learning failed to acquire a general enough representation of the constraints. The mismatching parses thus indicate that a number of constraints were not represented in the training data and some may be contradicted elsewhere in the corpus, particularly if semantic information was used to originally determine the correct parse. Although there is no universal method for measuring the similarity between two parse trees, it is possible to approximate how 'near' a mismatching parse is to the desired or target parse by simply grouping the module errors for a sentence together. Although this would give a broad measure of the parser's accuracy and provide an indication as to the level of correctness of a parse tree approximation, it is accepted that this will not provide an accurate measure. However it is deemed adequate for the purposes of this work and a statistically more accurate measure of parse tree approximations constitutes further work and is beyond the scope of the thesis.

Table of Module Errors To Determine The Accuracy of Mismatching Sentence Parses				
Mismatching Sentences	**Sentences with 1-3 Mod Errors**	**Sentences with 4-5 Mod Errors**	**Sentences with 6-10 ModErrors**	**Sentences with >10 Mod Errors**
305	194 (63.6%)	58 (19.0%)	35 (11.5%)	18 (5.9%)

Table 7.3 : The magnitude of module errors made for the 305 mismatching parses for the test sentences (Mod=Module).

Table 7.3 shows the magnitude of module errors for the 305 mismatching parses. The greater the number of module errors for a sentence the greater the differences between the mismatching parse and the desired parse for that sentence. It can be seen that 63.6% of the mismatching sentences were caused by up to only three module errors with 75 of these mismatching parses having only 1 module error. The output from the parser could thus be considered a close approximation of the LPC target parses. However, module errors of between 4 and 10 (inclusive) account for just over a third of the mismatching sentences and represents highly inaccurate parse tree approximations. Also, 6% of the mismatching parses were grossly inaccurate approximations containing more than 10 module errors. This could indicate that a number of linguistic constraints were not fully represented in the training data or may be contradicted elsewhere in the corpus.

Overall, the results of this analysis clearly demonstrates that the majority of the mismatching parses are close to their LPC parse.

A further subset of 50 mismatching parses is provided in Appendix L for the interested reader.

7.3.1. Overview

Although a thorough large-scale linguistic analysis of the resulting collection of matching and mismatching parse trees (i.e., tree-bank) produced by the parser in response to the constrained test sample is beyond the scope of this thesis, the behavioural characteristics of the parser will be assessed using a limited sample of this tree-bank.

As mentioned, the parser successfully matched 337 parse trees and produced 305 incorrect parse tree approximations (mismatches) for the of the 687 sentences in the constrained test sample. A subset of 10 matching parses and 10 mismatching parses is used for the purpose of this assessment. The intention of the assessment is to determine to what extent the mismatches would undermine the parser's effectiveness in terms of the task that it might be used for. The severity of a mismatch is thus indicated by the consequential shift in semantic interpretation of the resulting structure produced by the parser. The assessment also seeks to investigate the nature and types of the mismatches.

Also, a small number of sentences are artificially created to assess the limitations of the parser with respect to specific syntactic constructions. This may involve adapting existing LPC sentence structures. The expected LPC parse is then hypothesised and compared with the actual output of the parser. It should be noted that as the sentences are not from the LPC they fall outside the range that the parser might reasonably expect to generalise to.

Three types of test results will therefore be analysed to determine the behavioural characteristics of the parser :

i. Matching parses
ii. Mismatching parses
iii. Parses of artificially generated sentences

To aid clarity, the parse trees analysed are grouped in terms of the structural characteristics they possess. These groups are as follows :

- Simple sentence structures
- Multiple clause structures and constituent coordination
- Centre-embedded structures
- Mismatches due to Recogniser error
- Mismatches due to punctuation removal

Each group of test results is discussed in subsequent sections and explained using the appropriate matching and mismatching parse tree structures produced by the parser. A summary is given at the end of each section.

7.3.2. Simple Sentence Structures

Simple sentence structures commonly consist of a single clause that may contain a subject noun phrase and a finite verb phrase that may or may not have an associated object noun phrase structure. For example, the matching parse for the sentence *Mr Randall I mean*, illustrated in Figure 7.1, consists of a simple sentence structure with a subject noun phrase preceded by a noun phrase that consists of a titular noun and a name. This is succeeded by a *V* phrase that dominates the present tense verb *mean*. Figure 7.1 also illustrates that the parser has learnt to correctly associate first person singular personal pronouns with the subject noun phrase, *Na*.

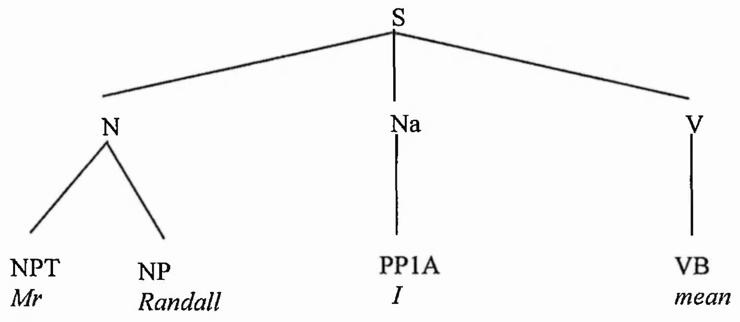


Figure 7.1 : The parser's matching parse for *Mr Randall I mean.*

Figure 7.2 shows the matching parse for the LPC sentence, *maybe a scheme is coming to fruition.* It illustrates a simple adverbial phrase *R* (i.e., dominates the single adverb *maybe*), followed by *N, V, P* structures. The *N* phrase consists of an indefinite article, *a*, and the main noun *scheme.* The *V* phrase contains the auxiliary verb, *is* and the main present tense verb *coming.* Although the preposition phrase consists of the simple preposition, *to*, followed by a noun, it shows that the parser has learnt to distinguish *to-infinitive* phrases from prepositional phrases beginning with *to.*

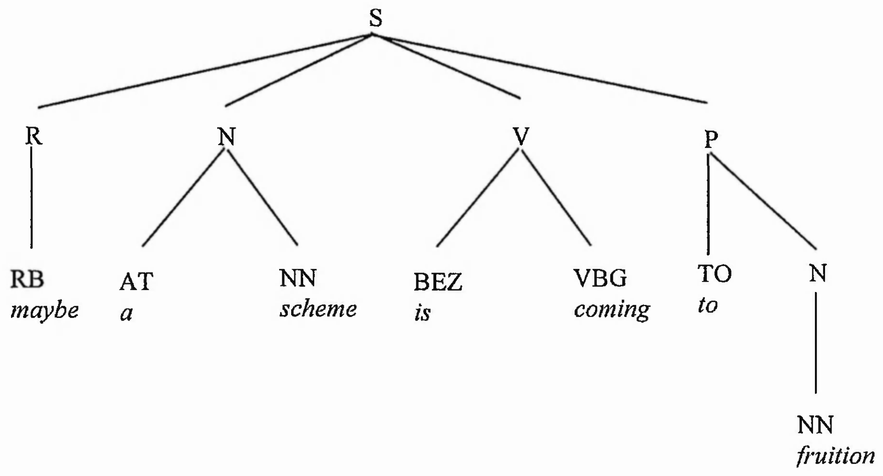


Figure 7.2 : The parser's matching parse for *maybe a scheme is coming to fruition.*

The matching parse for the sentence, *he'd begun to sweat*, illustrated in figure 7.3., demonstrates that the parser has successfully learnt to group the auxiliary verb *had* (represented by *'d*) and the main verb, *begun* (past tense). As an auxiliary verb *had* is used to make the *Perfect tense* of the main verb, *begun.* The parse structure in figure 7.3 shows that the parser has correctly learnt the relationship between non-finite verb phrases (*Vi*) and *to*-infinitive clauses (*Ti*).

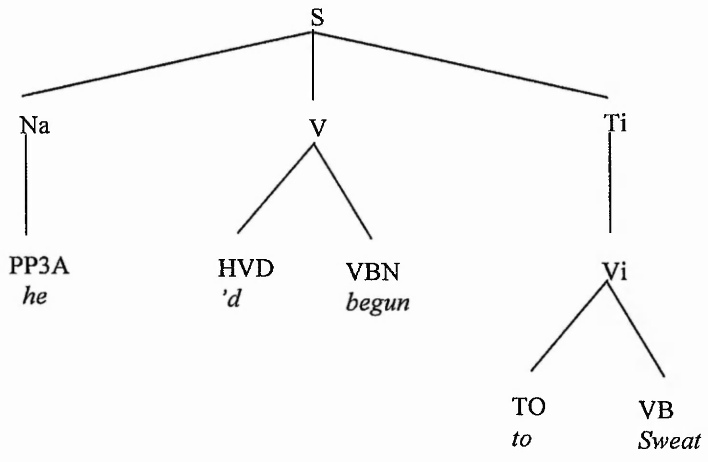


Figure 7.3 : The parser's matching parse for *he'd begun to sweat.*

The matching parse for *I was afraid of that*, shown in figure 7.4., strengthens the hypothesis that the parser has learnt to allow an adjective phrase to complement the subject noun phrase when the main verb phrase consists of an auxiliary verb such as *were*, *was*, *be* etc.

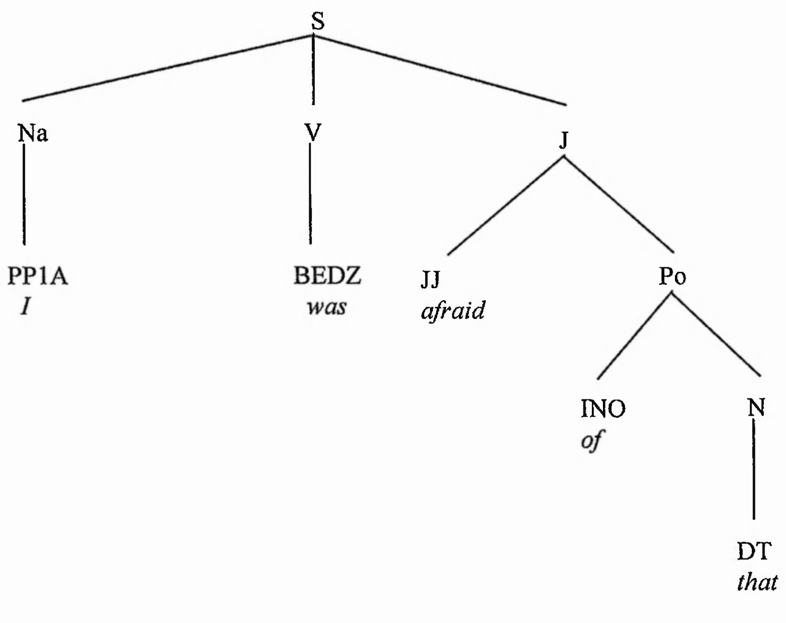


Figure 7.4 : The parser's matching parse for *I was afraid of that.*

The preposition, *of that*, is correctly attached to the adjective phrase to function as a post-modifier of the main adjective, *afraid.*

It is evident from the matching parse in figure 7.5 that the parser is able to recognise *to*-infinitive clauses (*Ti*) in a variety of contexts. Clearly, it has learnt to associate a *Vi* phrase with the word, *to*, followed by allowable verbs; and to associate *Vi* and the supporting noun and preposition phrase structures with a *to*-infinitive clause. The matching parse for the sentence, *I can nt bear to see her like this*, is composed of a subject noun phrase (*Na*), a main verb phrase (*V*) and the *to*-infinitive clause (*Ti*). The parser correctly recognised the subject noun phrase as the first person singular personal pronoun, *I*. The main verb phrase correctly consists of the modal auxiliary verb *can*, the negative *nt* (*not*), and the main verb *bear* thus demonstrating that the parser is able to process negatives. Additional to the parser correctly recognising the *to*-infinitive clause, it is also evident that the parser is able to correctly process third person personal pronouns as well as first person personal pronouns based on whether they are nominative or accusative. This distinction is made via using different syntactic tags and it appears the parser has learnt the behaviour of each tag. If the personal pronoun is third person and accusative (e.g., *his* or *her*) then it cannot be made the subject of a clause, however if it is nominative (e.g., *he,she,we* or *they*) then it is allowed to function as the subject of a clause or sentence. The parser is consistently able to make the correct linguistic predictions regarding personal pronouns, an impressive characteristic considering only 2.4% of its training data consisted of *Na* phrases.

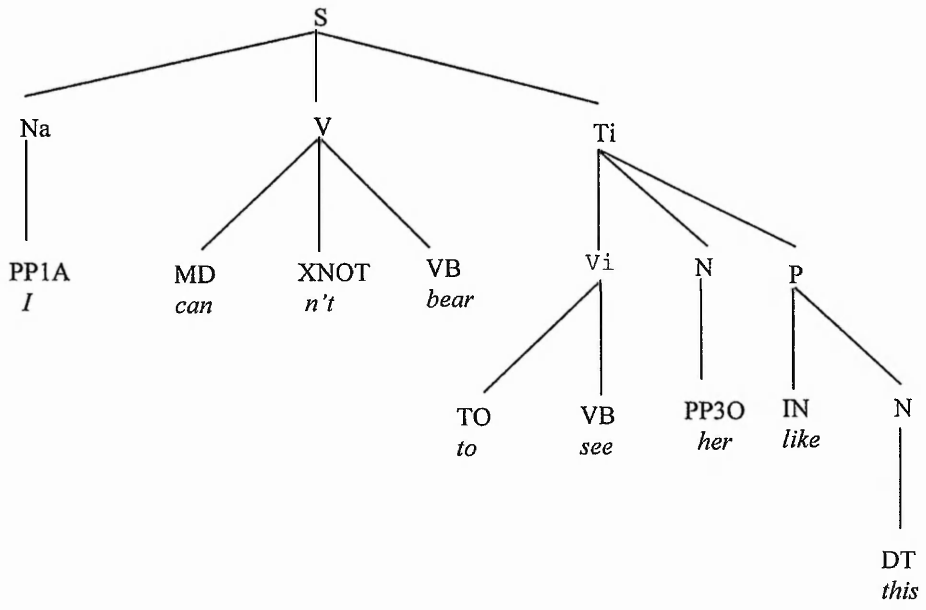


Figure 7.5 : The parser's matching parse for *I can n't bear to see her like this.*

The matching parse shown in figure 7.6 demonstrates that the parser allows adjective phrases to function as complements to the subject noun phrase, and the auxiliary verb links the adjective phrase to the subject. However, note that in this particular sentence, *Brandon* is marked as a standard noun phrase (*N*) rather than the subject noun phrase (*Na*). This illustrates the limitations of the LPC in that names cannot be marked as the subject noun phrase.

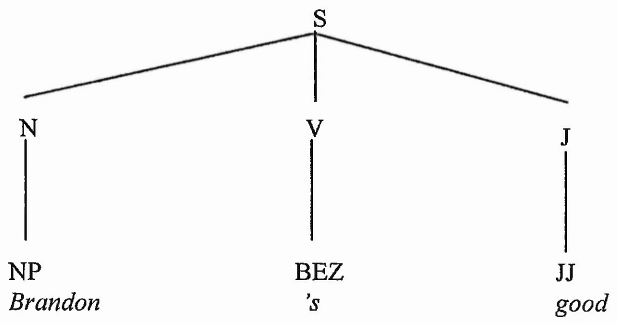
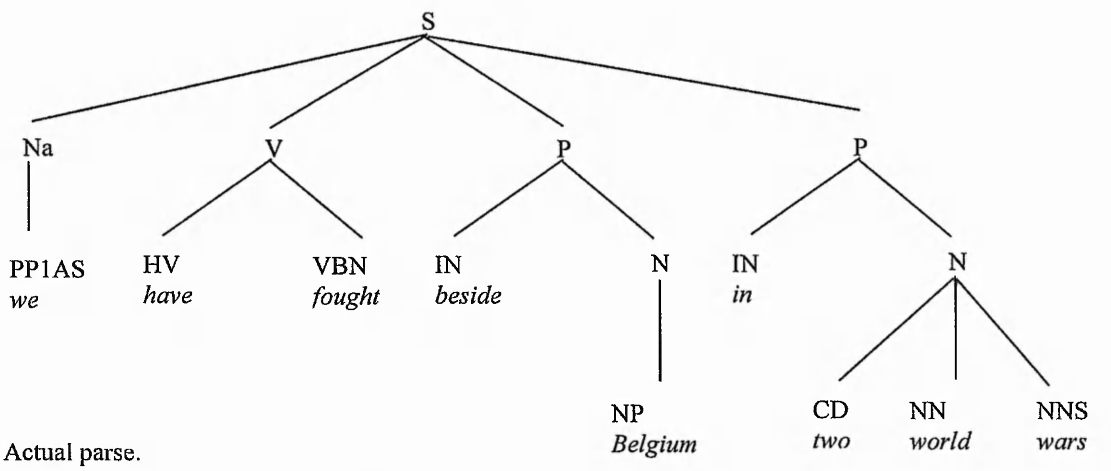


Figure 7.6 : The parser's matching parse for *Brandon's good.*

The mismatch shown in figure 7.7 is due to differing preposition attachment preferences. The LPC parse illustrated in 7.7(a) shows the second preposition phrase, *in two world wars*, as functioning as a complement to the verb phrase. However, the parser prefers to attach this preposition phrase as a post-modifier to the first preposition phrase, *beside Belgium*. This is illustrated in figure 7.7(b). This is a globally ambiguous sentence and the parser has provided a plausible alternative structure for the syntactic information provided e.g., consider the prepositional phrase, *in the European mainland.* The parser's attachment in 7.7(b) would be correct if the intention was to elaborate on Belgium rather than specify where the war was fought. It demonstrates a limitation of parsing based mostly on syntactically tagged words. Such attachment ambiguities require semantic information to be resolved and even then they may remain ambiguous. The structural attachment preference made by the parser in 7.7(b) is made repeatedly in the LPC and thus reflects attachment inconsistencies at the syntactic level in the corpus.

The results in this section have shown that the parser is able to parse simple sentence structures that consist of : noun phrases (including subject nouns), verb phrases (including auxiliary verbs and negatives), preposition phrases, adverbial phrases and adjective phrases. They demonstrate that it had correctly learnt to distinguish between the different uses of the word *to* i.e., as a to-infinitive in a non-finite verb phrase and as a preposition within a preposition phrase. The parser was also able to correctly associate to-infinitive clauses (*Ti*) with non-finite verb phrases (*Vi*). Although the parser produced an incorrect preposition attachment, it provided a semantically plausible alternative attachment.

(a) Target parse.



(b) Actual parse.

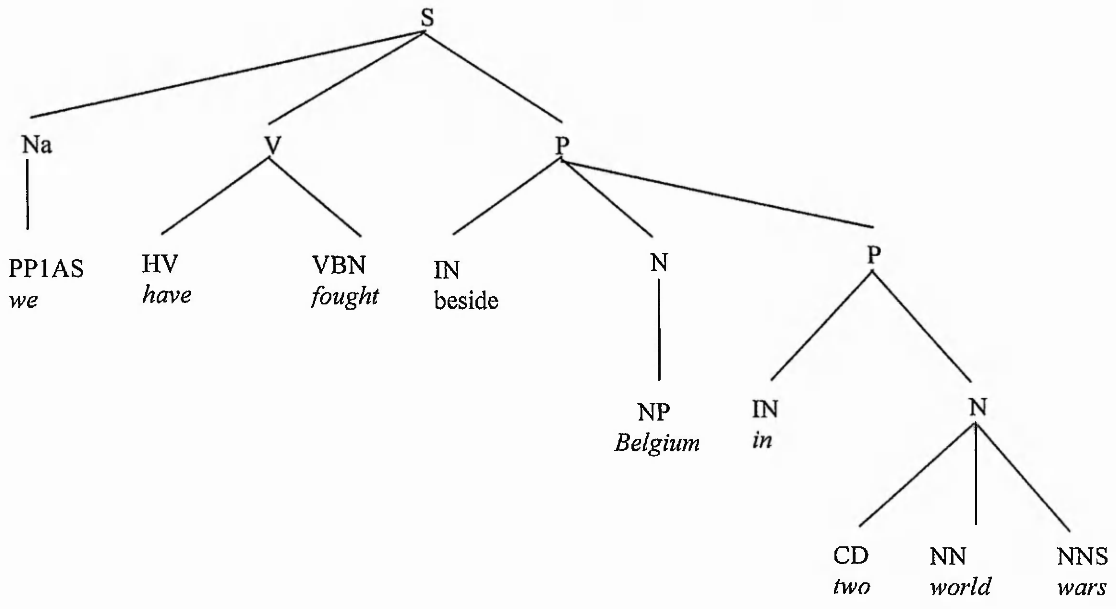


Figure 7.7 : (a) The target parse for *we have fought beside Belgium in two world wars.* (b) The parser's actual parse for the sentence.

7.3.3. Multiple Clause Structures And Constituent Coordination

A compound sentence structure is one that consists of two main clauses, joined by a coordinating conjunction. Coordinating conjunctions are words such as *and*, *but*, *or/either*, *neither/nor*, and *yet*. The word tag, *CC*, cover all these words in the LPC and are thus treated in the same way. Each clause is of equal importance and provides information of equal value.

Complex sentence structures are those that contain a subordinate clause as well as a main clause. A subordinate clause is one that contains special information about the main clause. It will usually be introduced by a subordinating conjunction such as *that*, *than*, *as*, *until*, *unless* and *although*. The word tag, *CS*, cover all these words in the LPC and are thus treated in the same way. When one clause is of principle importance and the other clause provides information about the principle one we have a complex sentence structure that consists of a single main clause and one subordinate clause.

This section analyses the matching and mismatching parse structures to assess the ability of the parser to perform constituent coordination using coordinating conjunctions and subordinating conjunctions. A number of modified LPC and artificially generated sentences will also be analysed after the matching and mismatching structures, to examine the parsers ability to process particular coordinate structures. Such sentences will be referred to as 'artificial' when used.

Coordination Using Coordinating Conjunctions

The matching parse in Figure 7.8 shows that the first clause consists of an *N* dominating *It* (personal pronoun in third person); a *V* dominating the auxiliary verb *was* and the main verb *done*. The second clause is the coordinated structure that consists of a basic sentence structure of *N V N*. This coordinated sentence is connected to the first clause using the coordinating conjunction *and*, represented by the word tag *CC*. The parser correctly groups the adjoining N and V phrases with the S+ structure to form the compound sentence structure (S&). This demonstrates the ability of the parser to recognise coordinating conjunctions and correctly perform constituent coordination – an advantage over other connectionist parsers such as Henderson's localist parser [14]. It also demonstrates the ability to correctly categorise third person personal pronouns that are neuter as *N* phrases rather than *Na* phrases.

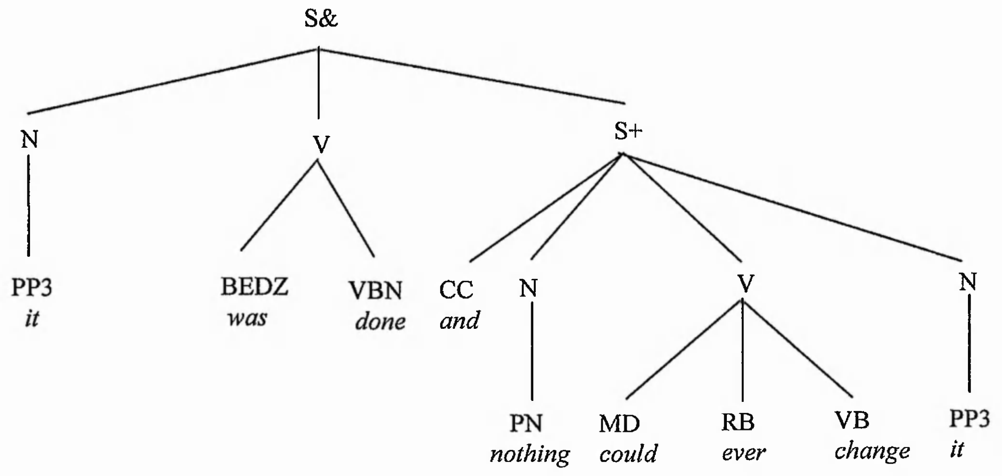


Figure 7.8 : The parser's matching parse for *it was done and nothing could ever change it.*

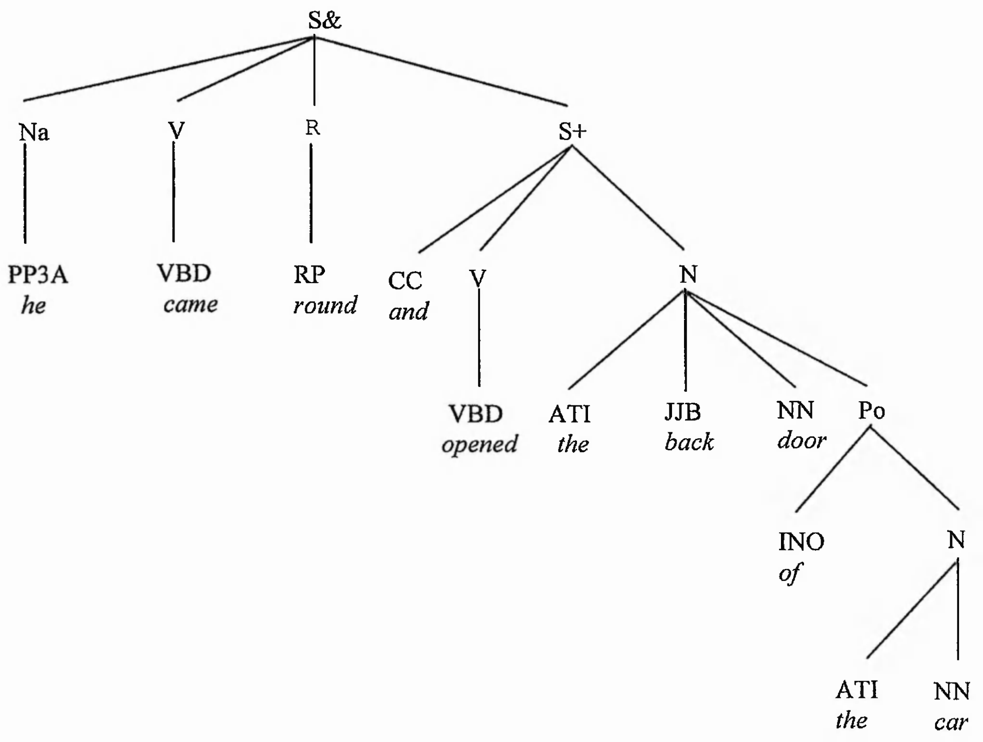


Figure 7.9 : The parser's matching parse for *he came round and opened the back door of the car.*

Figure 7.9 shows the matching parse for *he came round and opened the back door of the car*. This parse again illustrates the ability of the parser to join two main clauses with a coordinating conjunction. The first clause consists of the subject noun phrase, *Na*, dominating the third person singular subject *he*; a *V* dominating the past tense verb *came* and a supporting adverbial phrase, *R*, dominating the adverbial particle word *round.* The second main clause (i.e., the coordinated sentence *S*+) is joined with a coordinating conjunction. The coordinated sentence consists of a basic sentence structure of *V N*. However, the object noun phrase is more complex than that in figure 7.8. It consists of a post-modifying preposition phrase beginning with the preposition *of.* This demonstrates the parser's ability to correctly attach a preposition phrase to the appropriate object noun phrase. The parser correctly groups the adjoining *N V R* structure with the S+ structure to form a compound sentence structure (S&). This parse shows that the parser is able to correctly recognise the subject of a verb and attach the appropriate phrasal symbol to it.

As shown by the matching parse in figure 7.10, the first clause is correctly segmented into *Na, V, P* phrases. The subject, singular third person personal pronoun *she*, is dominated by *Na*; the past tense verb, *looked*, is designated as the main verb and is dominated by *V to* represent the main verb phrase group; the prepositional phrase, *into the bright* *eyes*, is dominated by *P* and correctly functions as an adverbial phrase.

The second clause, *they were expressionless*, is segmented into *Na, V, J* phrases and is joined to the first clause with a coordinating conjunction to form the compound sentence. The subject, plural third person personal pronoun *they*, is correctly dominated by a subject noun phrase symbol (*Na*); the progressive tense verb, *were*, is designated as the main verb of the second clause and is dominated by *V*; and the adjective phrase, *expressionless*, is dominated by *J* and correctly functions as a complement to the subject noun phrase for the clause. The matching parse shown in figure 7.10 therefore demonstrates the parser's ability to join main clauses together; recognise noun phrases functioning as the subject of a clause; allow prepositional phrases to function as an adverbial phrase; and to allow adjective phrases to act as complements to the subject noun phrase.

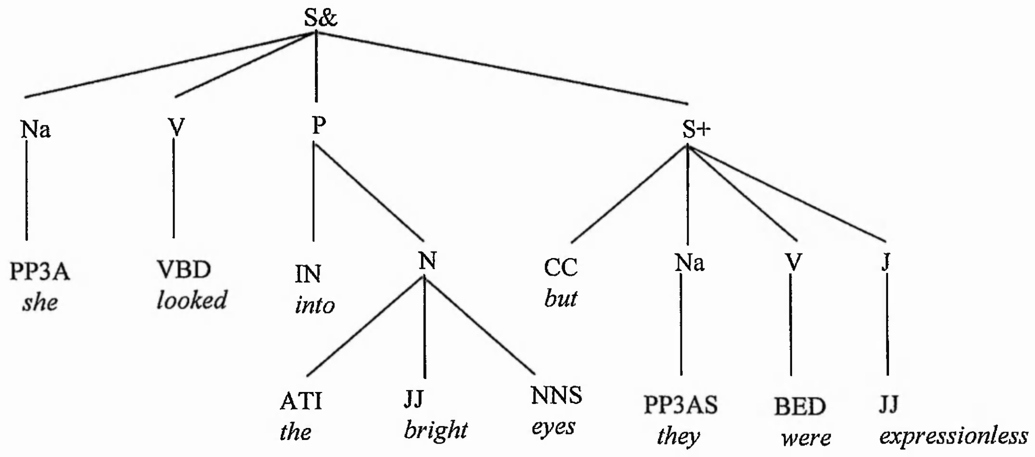
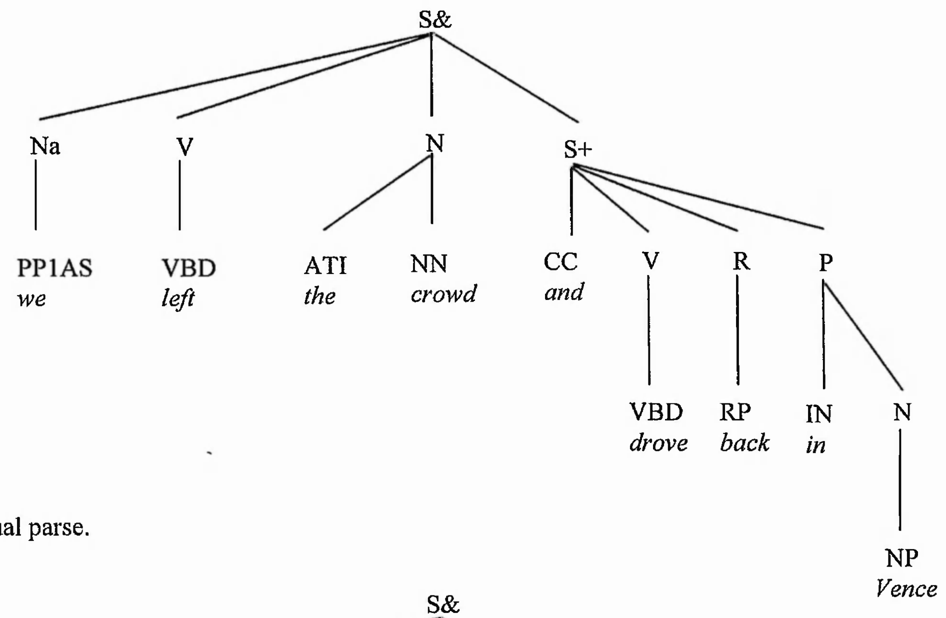


Figure 7.10 : The parser's matching parse for *she looked into the bright eyes but they were expressionless.*

Figure 7.11(a) shows the LPC parse structure for the sentence, *we left the crowd and drove back in Vence.* The sentence consists of two clauses joined together by a coordinating conjunction to form a compound sentence structure (*S&*). Figure 7.11(b) illustrates that the parser is able to correctly match the structure of the first main clause with that in the LPC parse. The second clause is a coordinated sentence (*S+*) that is joined by the coordinating conjunction, *and.* As figure 7.11(b) illustrates, the parser's structure for this clause differs from that of the LPC parse. The LPC structure separates the main verb and adverb into separate phrases whereas the parser joins the adverb to the verb phrase so that *back* acts as a post-modifier to the main verb. Although it can be postulated that either interpretation is syntactically plausible, the parser has made an incorrect attachment considering the content of the preposition, *in Vence.* All occurrences of *RP* in the parser's training data are attached to *R* phrase groups regardless of the context of the adverb. It indicates that both the RLD and Recogniser networks have not fully learnt a rule to correctly group and classify adverbial phrases. However, if the preposition phrase had consisted of, *in the car*, then the parser would have produced a syntactically and semantically plausible parse in figure 7.11(b).

(a) Target parse.



(b) Actual parse.

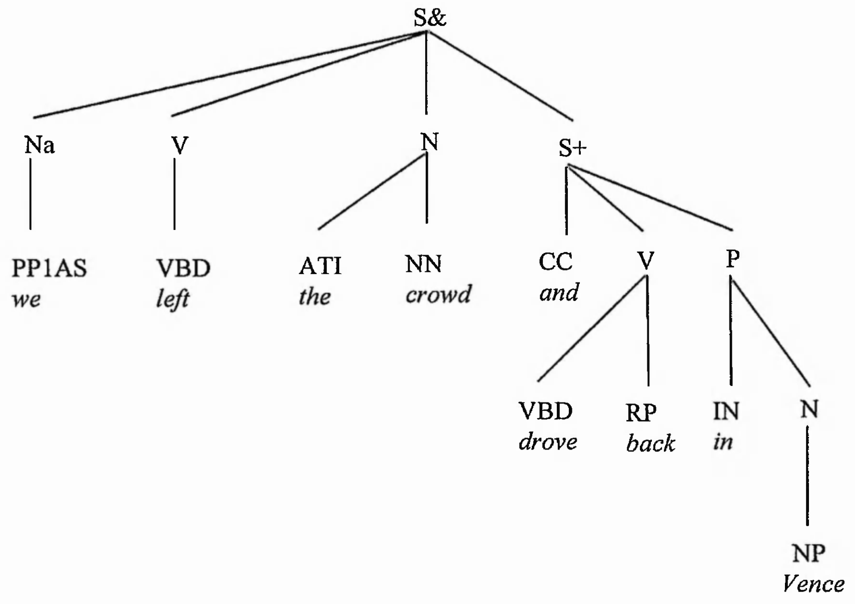


Figure 7.11 : (a) The target parse for *we left the crowd and drove back in Vence.* (b) The parser's actual parse for the sentence.

In figure 7.12(b) the parser demonstrates a preference for attaching the right-most preposition phrase to the nearest noun phrase. It can be seen that rather than attaching the preposition phrase, *in the water*, directly to *S+* so that it functions as an adverbial phrase, the parser attaches the prepositional phrase as a post-modifier to the noun phrase, *his hand.*

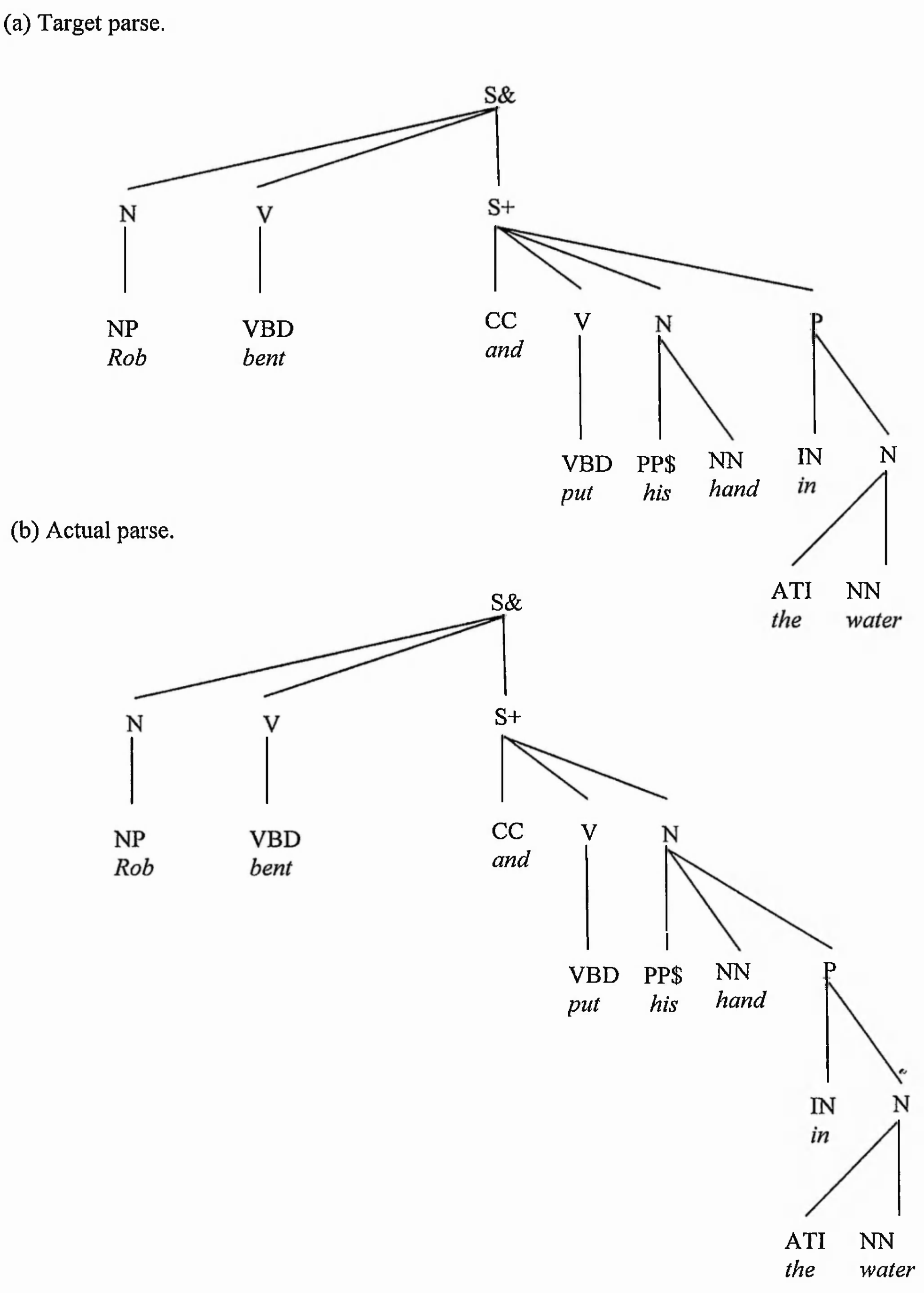


Figure 7.12 : (a) The target parse for *Rob bent and put his hand in the water*. (b) The parser's actual parse for the sentence.

This behaviour is similar to Minimal Attachment preferences discussed by Frazier [114] and a linguistic feature consistently exhibited by the parser. However, the parser's output is a syntactically plausible alternative and very close to the corresponding LPC parse. For example, if the past tense verb was *clenched* rather than *put* then parse 7.12(b) would be both syntactically and semantically valid.

The above matching and mismatching parses have demonstrated that the parser is able to correctly join clauses of equal importance together using coordinating conjunctions. However, although any phrase type can be coordinated using a coordinating conjunction it is only been established that the parser is able to attach coordinate *S* structures. The maximum number of coordinate structures allowed by the parser has also yet to be determined. The question that needs to be answered is therefore : how complex can compound phrase structures be before the parser begins to fail? A number of artificially generated sentences will be used to answer these questions. Rather than test the parser on every phrase type, the following experiments will focus on compound noun phrases to establish whether the parser can join phrase types other than *S* with a coordinating conjunction. Compound noun phrase structures are significantly more common than any other compound phrase structure throughout the LPC and it is therefore expected that the parser has learnt to recognise such structures.

Consider the following *artificially generated* compound noun phrases :

(S7.1) wisdom and knowledge.
(S7.2) a simple and informative book.
(S7.3) a very simple and informative book.
(S7.4) the computer with disks or a manual book.
(S7.5) the computer with disks or a manual book or a mouse.

Sentence S7.1 shows a simple compound noun phrase structure i.e., two nouns joined together by a coordinating conjunction. The desired output of the parser for this artificial sentence is illustrated in labelled bracketing format below :

[S [N [NN& wisdom_**NN [NN+** and_**CC** knowledge_**NN NN+] NN&] N] ._. S]**

The parser is able to match this parse and correctly perform word level coordination (i.e., *NN+*) as defined by Garside et al [147] and as exhibited throughout the LPC. The principle for word level coordination is that it must take place if two or more words inside the same constituent are linked by a coordinating conjunction. The parser performs this task via grouping *and knowledge* as a coordinated noun (*NN+*) and joining this to *wisdom* to form the correct compound noun structure (*NN&*). The compound noun structure is then attached to a noun phrase.

Sentence S7.2 illustrates the coordination of adjectives and the main noun within a compound noun phrase. The main noun, *book*, is pre-modified by the indefinite article, *a*, and the adjectives, *simple* and *informative*. The output of the parser is as follows :

[S [N& a_AT simple_**JJ [N+** and_**CC** informative_**JJ** book_**NN N+] N&] S]**

The parser is able to match the above structure and thus forms the appropriate compound noun phrase structure via grouping the coordinating conjunction, *and,* with the second adjective and the main noun to form the second conjoin of a compound noun phrase (*N+*). It then correctly attaches the indefinite article and first adjective to *N+* to form the compound noun phrase structure (*N&*).

However, the parser begins to fail when the qualifier, *very,* is added to support the adjectives as shown in sentence S7.3. This forms a noun phrase structure that contains an embedded coordinated adjective phrase. The desired LPC parse is :

[S [N a_**AT [J&** very_**QL** simple_**JJ [J+** and_**CC** informative_**JJ J+] J&]** book_**NN N] S]**

The addition of a single qualifier to the compound noun phrase has a detrimental effect on the parser's behaviour. The parser forms the following parse structure for sentence S7.3 :

[S [N a_**AT** very_**QL N] [Na-** simple_**JJ [N+** and_**CC** informative_**JJ** book_**NN N+] Na-] S]**

It can be seen that the parser is unable to separate the compound adjective phrase from the noun, *book.* It therefore groups the noun with the second clause of the adjective phrase to form *N+*. The presence of *very* causes the parser to group the adjective, *simple*, with *N+* to create a subject noun phase that is marked as a second conjoin of a compound subject noun phrase (*Na-*). This is an obvious error caused by the failure of the phrase segmentation module. The desired behaviour is for the parser to have grouped ' *very simple*' with J+ to form a compound adjective phrase structure (J&) thus initially ignoring the contribution of the main noun *book.* The main noun, compound adjective phrase and the indefinite article should then have been grouped to form a noun phrase. This demonstrates that the parser has significant difficulty in segmenting embedded adjective phrases that contain qualifiers from noun phrases.

Compound phrase structures that contain more than one coordinating conjunction also reveal the parser's limitations. Consider the sentences S7.4 and S7.5.

According to the coordination constraints in the LPC defined by Garside, the desired LPC parse should be :

[S [N the_**ATI** computer_**NN [P** with_**IN [N&** disks_**NNS [N+** or_**CC** a_**AT** manual_**NN** book_**NN**
N+] N&] P] N] S]

The parser is able to match the above structure. However, when a further conjoin of the compound noun phrase is appended to the end of the sentence, the parser is unable to identify it as such. This is illustrated in the parser's response to sentence S7.5. Sentence S7.5 only differs from S7.4 in that it contains a further noun phrase attached to the end of the sentence via a coordinating conjunction.

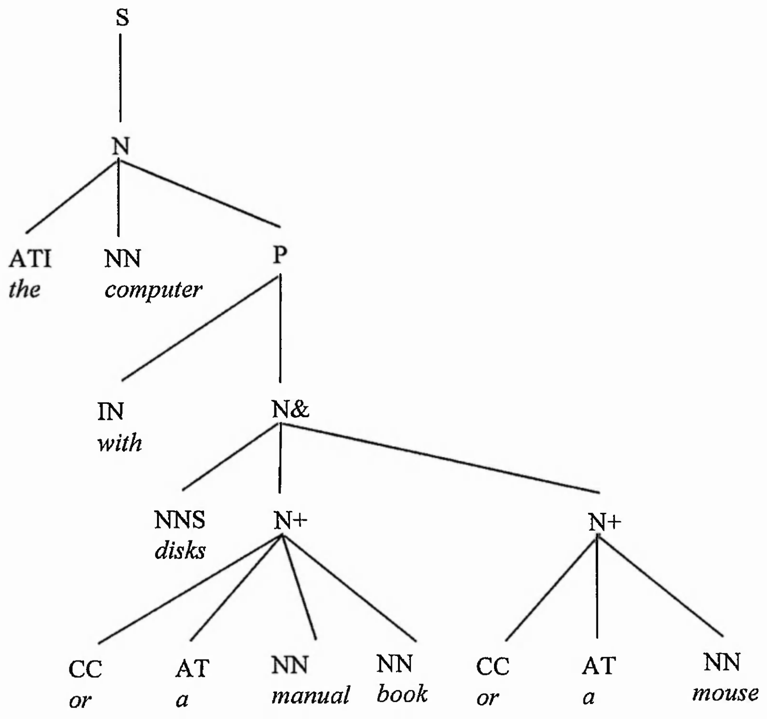
Figure 7.13(a) shows the desired parse for sentence S7.5 and figure 7.13(b) illustrates the actual output of the parser. The parses are depicted in parse tree format for clarity. As figure 7.13(b) illustrates, the parser divides the sentence into two main clauses rather than simply attaching the additional noun phrase as a subsequent conjoin (i.e., *N*+) to the compound noun phrase. The parser has created two compound *S* structures for S7.5. The first *S&* structure acts as the root of the parse tree and is an acceptable decision made by the parser as it has also formed a corresponding coordinated sentence structure (*S*+). However, the second *S&* structure violates the LPC principles in that there are no subsequent *S* structures attached to it and there is already a compound *S* structure functioning as the root of the tree. It is actually used to dominate the main noun phrase, *the computer with disks*. Although the *S*+ structure matches the root of the parse tree, it does not match the coordinated noun phrases functioning below it. A compound noun phrase structure is required to dominate the *N*+ structures. Further experiments with sentences containing two coordinating conjunctions produced similar results.

It is clear from the experiments performed with coordinating conjunctions that the parser is able to process sentences containing one coordinated phrase structure. If more than one coordinated phrase exists then the parser begins to make significant structural errors. Also, some pre-modifiers to the coordinated phrase may affect the behaviour of the parser as demonstrated with the addition of qualifiers to compound noun phrase structures.

Coordination Using Subordinating Conjunctions

The subordinating conjunction itself is usually the first word of the subordinate clause and the clause is commonly a finite adverbial clauses (*Fa*) or finite nominal clauses (*Fn*). The parser has been successfully trained to recognise all 22 instances of subordinating conjunctions functioning as the first word of an *Fa* or *Fn* clause in the training data. However, during the sentence-level test process the parser was unable to match all 20 of the LPC parses from the constrained test sample that contained a subordinating conjunction. Although the parser was able to recognise the subordinating conjunction as the first word of the subordinate clause, it was unable to fully match the *Fn* or *Fa* clause structure in which it functioned.

(a) Target parse.



(b) Actual parse.

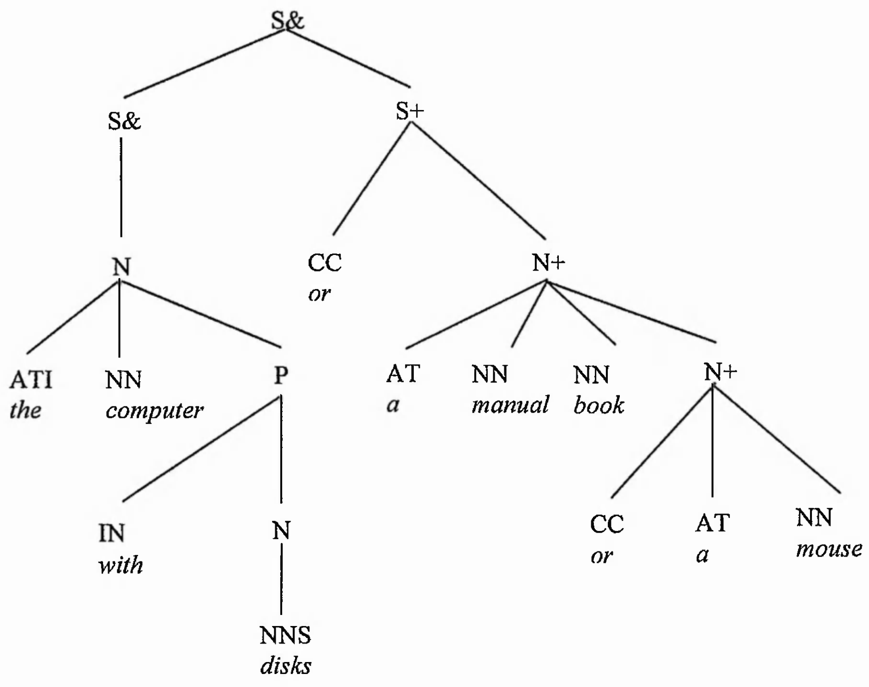


Figure 7.13 : (a) The target parse for *the computer with disks or a manual book or a mouse* (artificial sentence S7.5). (b) The parser's output demonstrating the inability to process two coordinating conjunctions.

For example, consider the mismatching parse illustrated in Figure 7.14. It can be seen from the LPC parse in 7.14(a) that the sentence contains two clauses : a finite nominal clause (*Fn*) beginning with *that*, and a past participle clause (*Tn*). The parse produced by the parser is remarkably close to that in the LPC. The only structural difference between the parses is that the parser has been unable to associate the single past participle tag, *VBN*, with a past participle verb phrase tag, *Vn*. It has mistakenly grouped *VBN* with the preposition phrase (*P*) to form a *Vn* phrase.

This has lead to the parser to ignore the *Tn* association and violate the LPC principle that *Vn* phrases must be attached to *Tn* clauses. However, this is not a significant structural error as the preposition phrase, i.e., *at local level*, still functions as an adverbial phrase to the main verb of the clause, i.e., *agreed*. If the grammatical constraints encoded in the LPC imply that a preposition phrase cannot directly function next to a past tense lexical verb then the parser has violated these constraints. However, this is of no detriment to the whole syntactic structure of the sentence. The past participle verb phrase, *Vn*, functions as a post-modifying phrase to the main noun, *payments*, that belongs to the noun phrase of the main preposition phrase within the finite nominal clause. The parser is able to recognise that *Fn* clauses begin with the subordinating conjunction, *that_CS*, and function in the position of an object noun phrase. Even though the parser's attempt deviates slightly from the LPC parse in figure 7.14(a) it shows that it is able to parse complex sentences by correctly joining subordinate clauses and in this case the semantic interpretation is unaffected by such minor deviations.

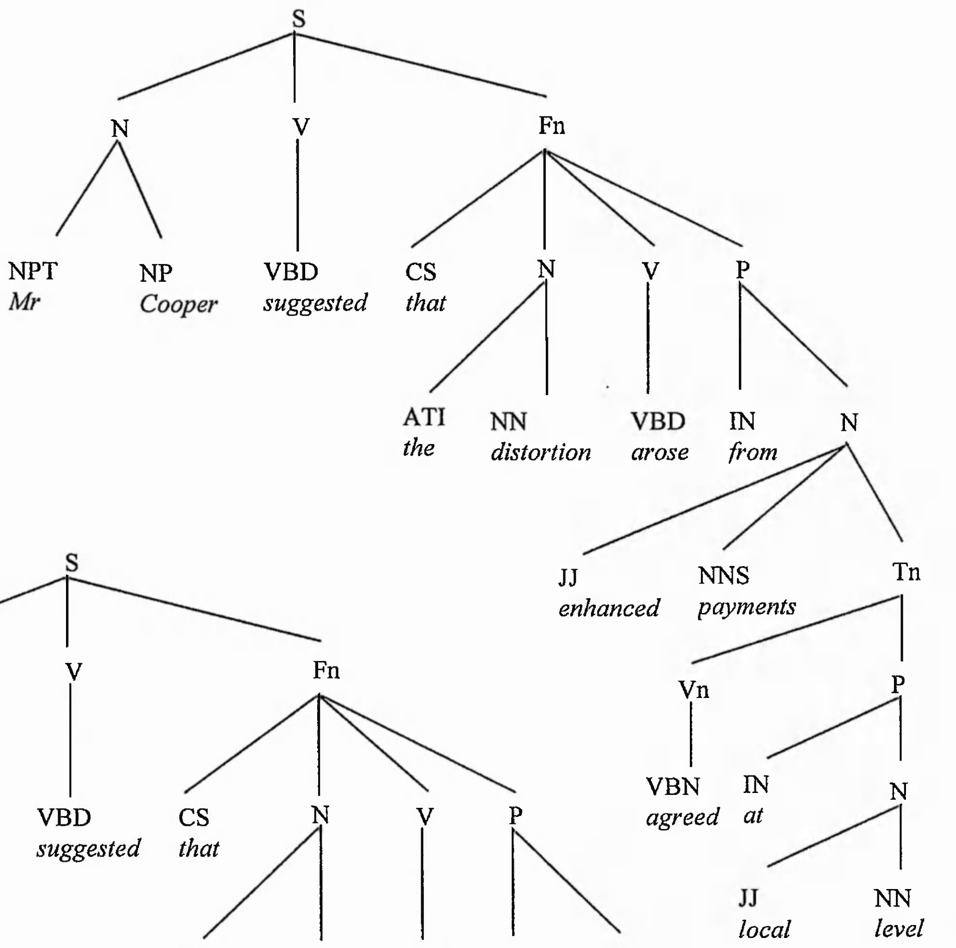
The intention of these experiments is therefore to establish the point at which the parser begins to fail when coordinating subordinate *Fn* or *Fa* clauses. Consider the following sentences :

(S7.6) check that the plug is connected.
(S7.7) check first that the plug is connected.
(S7.8) check that the plug is properly connected.

Sentence S7.8 is an actual LPC sentence and sentences S7.6 and S7.7 are derivatives of S7.8. Sentence S7.6 consists of a simple finite nominal clause preceded by a verb phrase. The present tense verb, *check*, forms the verb phrase. The finite nominal clause begins with the subordinating conjunction, *that*, and contains the noun phrase, *the plug*, and the verb phrase, *is connected.* The output of the parser produces the correct syntactic structure for this sentence S7.6 demonstrating that it is able to process simple subordinate clauses. The desired LPC parse and thus the parser's output is as follows :

[S [V check_**VB V] [Fn** that_**CS [N** the_**ATI** plug_**NN N] [V** is_**BEZ** connected_**VBN V] Fn] S]**

(a) Target parse.



(b) Actual parse.

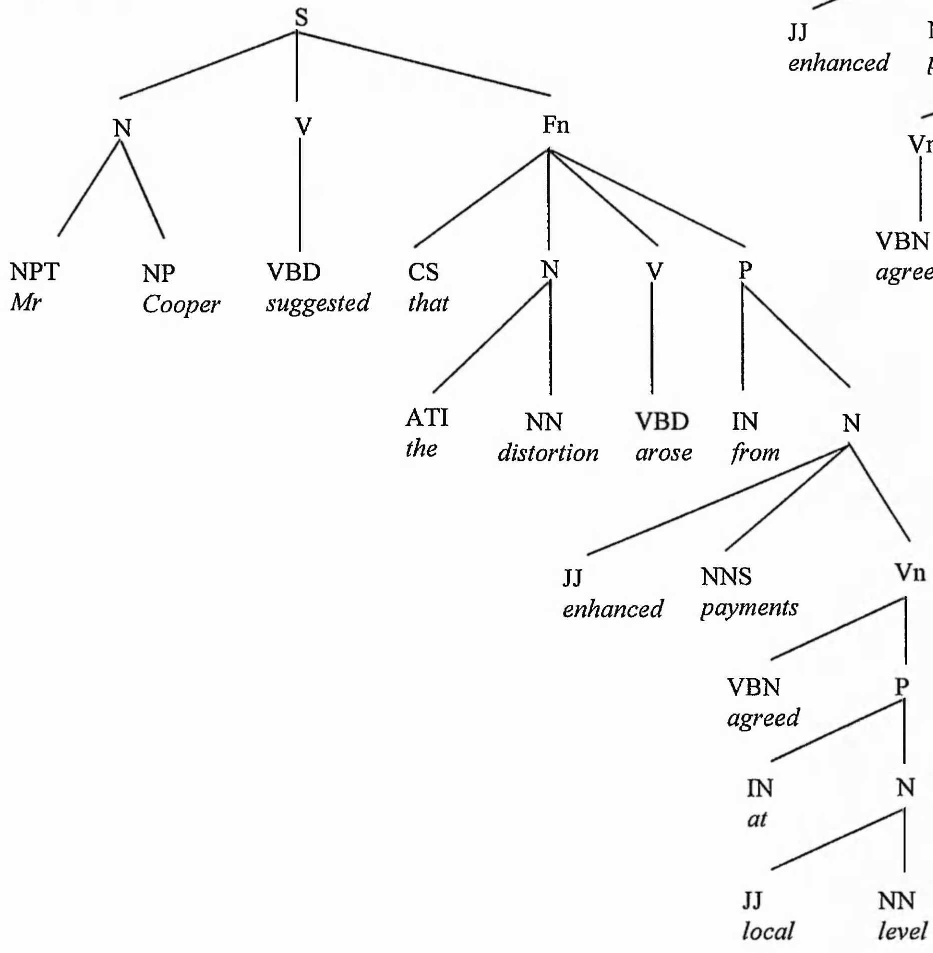


Figure 7.14 : (a) The target parse for *Mr Cooper suggested that the distortion arose from enhanced payments agreed at local level*. (b) The parser's actual parse for the sentence.

Sentence S7.7 differs slightly from S7.6 in that an adverbial phrase that consists of the single adverb, *first*, precedes the finite nominal clause. The desired LPC parse structure is thus :

[S [V check_**VB V] [R** first_**RB R] [Fn** that_**CS [N** the_**ATI** plug_**NN N] [V** is_**BEZ** connected_**VBN V] Fn] S]**

This slight modification causes problems for the parser as shown below :

[S [Rq check_**VB** first_**RB Rq] [Fn** that_**CS [N** the_**ATI** plug_**NN N] [V** is_**BEZ** connected_**VBN V] Fn] S]**

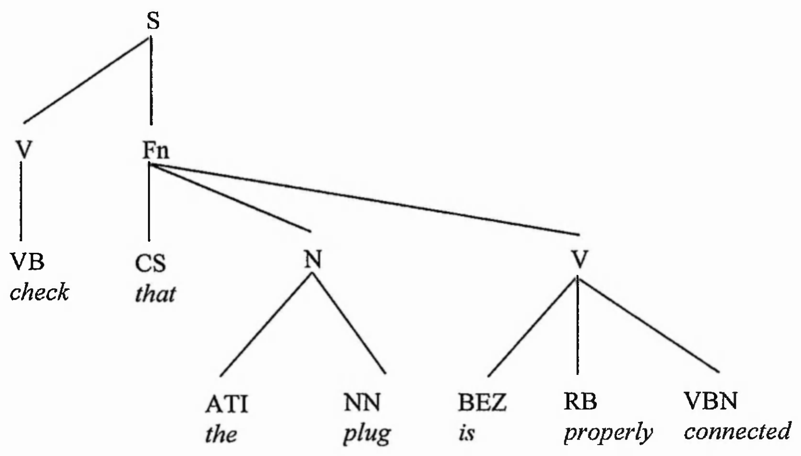
Rather than segmenting the main verb and adverb into two separate phrases, the parser incorrectly groups '*check first*' into a *Rq* phrase which represents an adverbial phrase beginning with a WH-word. This is an error clearly made by the RLD phrase segmentation module. This indicates that the parser has difficulty in segmenting adverbs and verbs into separate phrases in particular contexts. However, the structure of the *Fn* clause is unaffected and the parser has correctly coordinated the clause with a subordinating conjunction.

Sentence S7.8 inserts the adverbial phrase into the subordinate *Fn* clause rather than outside it to assess its affect on the syntactic structure of the sentence. The target LPC parse for sentence S7.8 is illustrated in figure 7.15(a). As figure 7.15(b) shows, this modification catastrophically affects the behaviour of the parser as it is unable to segment the sentence into the appropriate phrase and clause structures. Although the parser does not fail it produces a syntactically incorrect verb phrase structure, grouping all input words into a single verb phrase. This behaviour illustrates the structural sensitivity of the parser and its tendency to become increasingly abstract when it is unable to correctly segment the sentence. The above examples in principle demonstrate that the parser *is* able to perform constituent coordination with subordinating conjunctions. It is the complexity of the coordinated clauses and whether the parser has learnt to correctly recognise the subordinate clauses that determines the success of the coordinating operation. For instance, as illustrated previously, the parser has learnt to competently recognise and process *Ti* clauses as there were 59 instances of *Ti* clauses in training set. However, the parser is limited at processing *Tn* clauses as the coverage of these clauses within the training set is comparatively poor i.e., only 16 instances of *Tn* clauses. The expected behaviour of the parser is therefore that it will perform more competently on subordinate clauses containing a *Ti* clause than those containing a *Tn* clause. To demonstrate this point, consider the manually created sentences that are based on S7.8 :

(S7.9) check that the plug is connected to use the computer.

(S7.10) check that there are four research posts in AI.

(a) Target parse.



(b) Actual parse.

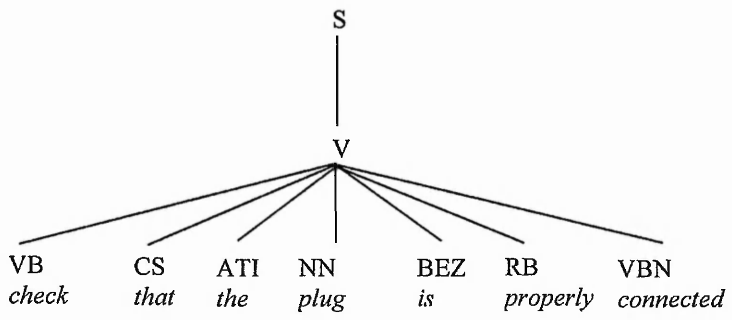


Figure 7.15 : (a) The LPC parse for *check that the plug is properly connected* (sentence S7.6). (b) The parser's output demonstrating the inability of the parser to process subordinate Fn clause structures containing a general adverb.

Sentence S7.9 has the *Ti* clause, *to use the computer,* added to sentence S7.6. when presented with this sentence, the parser is able to automatically detect this as a *Ti* clause embedded within the *Fn* clause. This demonstrates that the parser is also able to process complex sentence structures containing two or more clauses. The LPC parse and thus the parser's matching output for sentence S7.9 is :

> **[S [V** check_**VB V] [Fn** that_**CS [N** the_**ATI** plug_**NN N] [V** is_**BEZ** connected_**VBN V] [Ti [Vi** to_**TO** use_**VB Vi] [N** the_**ATI** computer_**NN N] Ti] Fn] S]**

However, the parser's output for sentence S7.10 illustrates that the parser has difficulty recognising *Tn* clauses that are embedded within the coordinated finite nominal clause due to the lack of coverage in the training data. Figure 7.16(a) illustrates the target LPC parse for sentence S7.10 and 7.16(b) shows the parser's output for the sentence. It can be seen from figure 7.16(a) that the embedded *Tn* clause contains a past tense verb phrase (*Vn*) and a prepositional phrase (*P*). The *Vn* phrase consists of the single past tense verb, *concerned.*

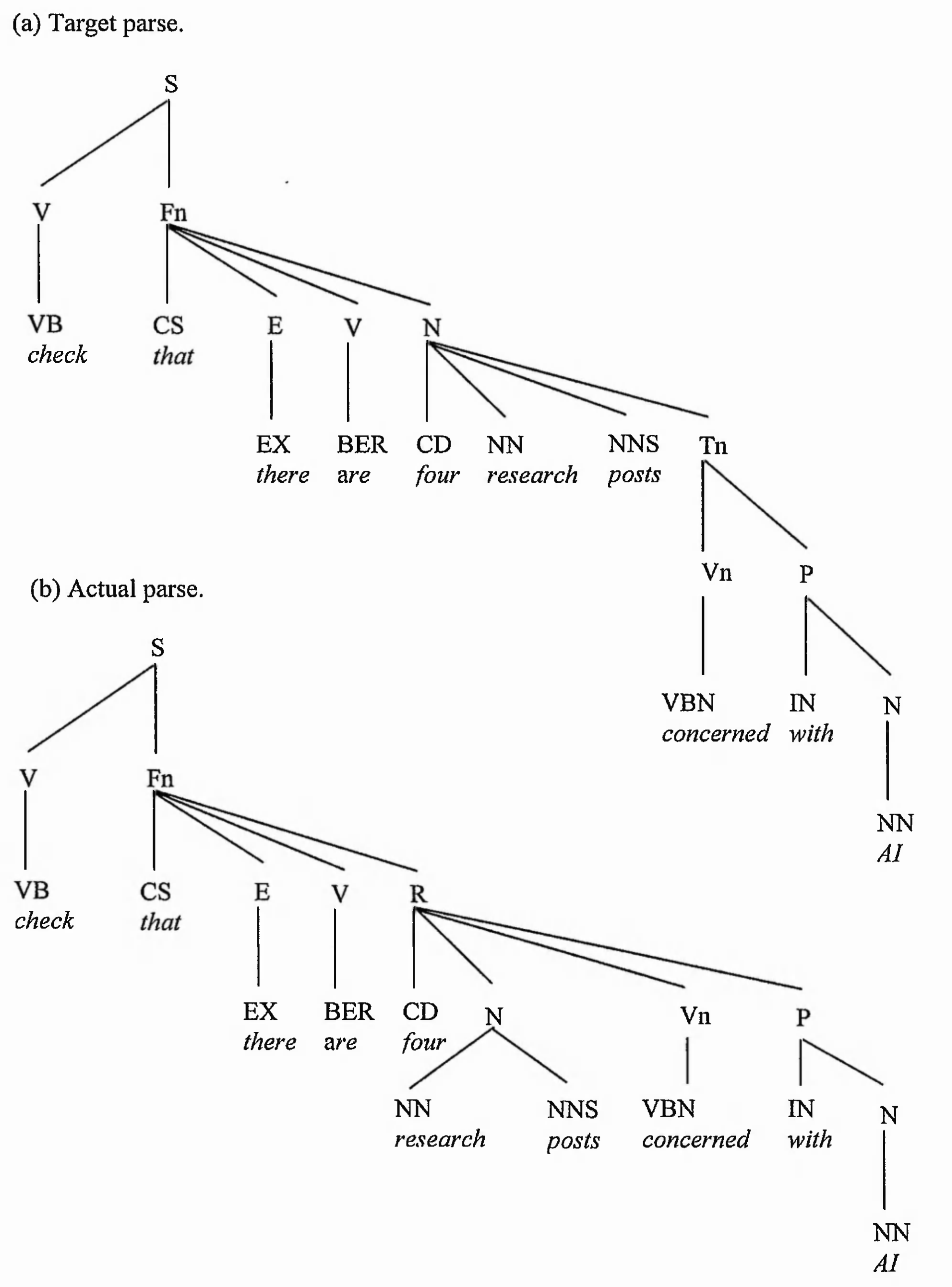


Figure 7.16 : (a) The target parse for *check that there are four research posts concerned with AI* (sentence S7.10). (b) The parser's output illustrating its inability to fully recognise embedded Tn clauses.

The preposition phrase consists of the preposition, *with*, and a noun phrase containing the main noun, *AI*. The parse is able to construct the *Vn* phrase and the *P* phrase, however it is unable to attach it to a corresponding *Tn* clause as shown in figure 7.16(b). Consequently, the parser groups *research posts* into a simple *N* phrase and then attaches *four* with *N*, *Vn* and *P* to form a syntactically incorrect adverbial phrase.

This section has demonstrated the ability of the parser to parse compound and complex sentence structures by performing constituent coordination using both coordinating and subordinating conjunctions. It was shown that the parser is capable of processing compound *S*, *N* and *J* structures using coordinating conjunctions and the ability to perform word-level coordination was also shown. The parser is able to distinguish between the types of pronouns that can be used as a subject noun phrase and those that cannot. Although some incorrect preposition attachments were made, the parser was able to provide semantically plausible alternative structural attachments. This is an impressive characteristic considering the parser has no access to semantic information.

A major limitation of the parser is that if there are more than two main clauses to be coordinated then the number of structural errors made by the parser increases. Also, the inclusion of pre-modifiers to a subordinate clause can affect the behaviour of the parser as demonstrated with the addition of qualifiers to compound noun phrase structures. Another major limitation encountered during the analysis is that it was evident that the RLD and Recogniser networks have not successfully learnt a simple rule to segment and classify adverbial phrases. The Recogniser network is consequently very sensitive to the position of adverbial phrases and the RLD has a tendency to incorrectly group adverbial phrases with verb phrases. For example, inserting a general adverb into a subordinate clause can have a catastrophic effect on the resulting parse. Further problems are encountered when the parser is expected to associate a past-participle verb phrase (*Vn*) with a past-participle clause (*Tn*). The parser enables a *Vn* to function as a *Tn* clause thus ignoring *Tn* clauses altogether.

7.3.4. Centre-Embedded Structures

Sentence structures that contain embedded structures typically consists of one or more phrase or clause that intervenes between two or more words that belong to the same clause. This creates problems for a syntactic parser as it must have the ability to preserve information about any long-distance dependencies in order to correctly process sentences containing embedding.

The parser demonstrates its ability to recognise embedded phrases in the matching LPC parse shown in figure 7.17. It can be seen from figure 7.17 that the parser is able to preserve the main noun, *mistress*, whilst it processes the embedded genitive phrase, *any mans*. It also presents an interesting structure found in the LPC in that the adverb, *never*, and the verb, *be*, act as premodifiers to the main noun, *mistress*, of the noun phrase rather than as the main verb phrase.

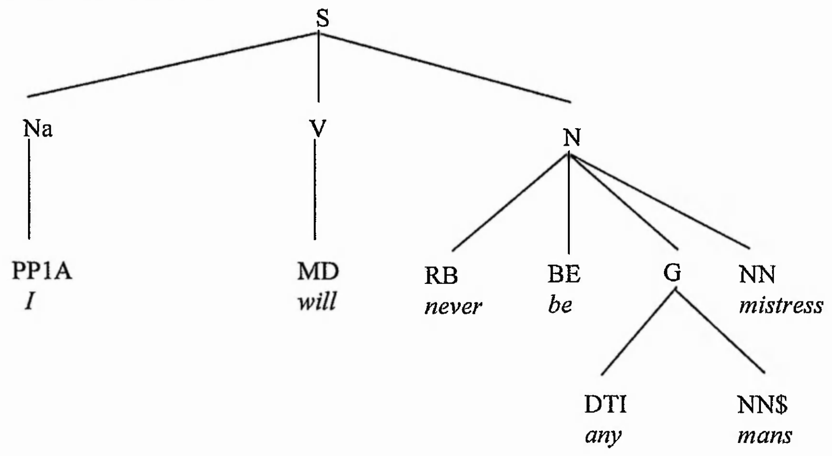
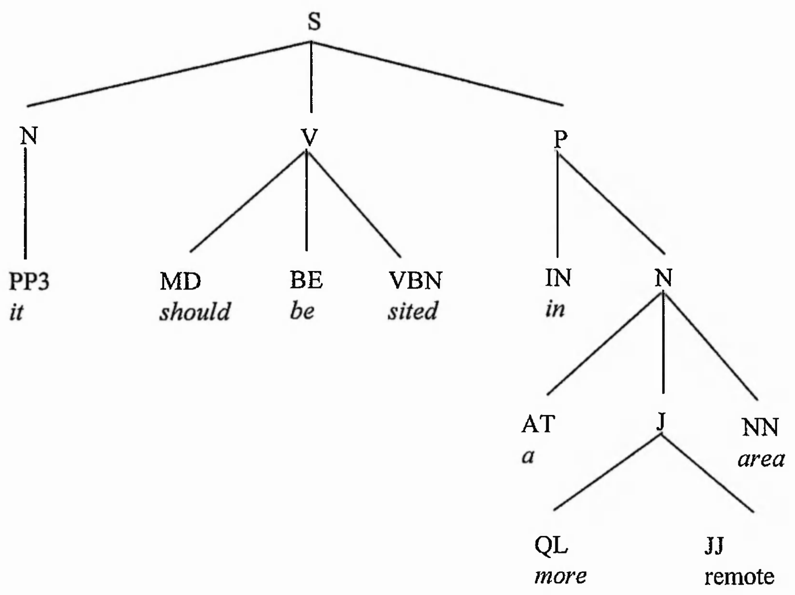


Figure 7.17 : The parser's matching parse for *I will never be any mans mistress.*

The modal auxiliary verb, *will,* is separated from the adverb and verb and appears to act as the main verb. An alternative parse would be for the modal verb to function in the first position of the verb phrase, followed by the adverb and then the verb. The noun phrase would therefore consist of the genitive phrase and the main noun, *mistress.* However, the parse for the sentence, *I will never be any mans mistress* matches that in the LPC and thus represents the structural preferences that have been learnt from the linguistic constraints defined in the LPC.

However, the parser relies upon its generalisation ability when processing unique sentence structures with embedded phrases and this is demonstrated in the mismatching parse shown in figure 7.18. It can be seen from figure 7.18(b) that the parser is unable to fully decompose the noun phrase that is attached to the prepositional phrase. The structural mismatch is not detrimental to the structure as a whole and the coarse semantic interpretation of both sentences would be identical. The adjective phrase illustrated in figure 7.18(a) is embedded within the noun phrase and the fact that the adjective phrase belongs to the noun phrase is still reflected in 7.18(b). Rather than reducing the qualifier, *more*, and the adjective, *remote*, into an adjective phrase that separates the indefinite article, *a*, from the main noun, *area*, the parser simply groups all the words into a simple noun phrase.

(a) Target parse.



(b) Actual parse.

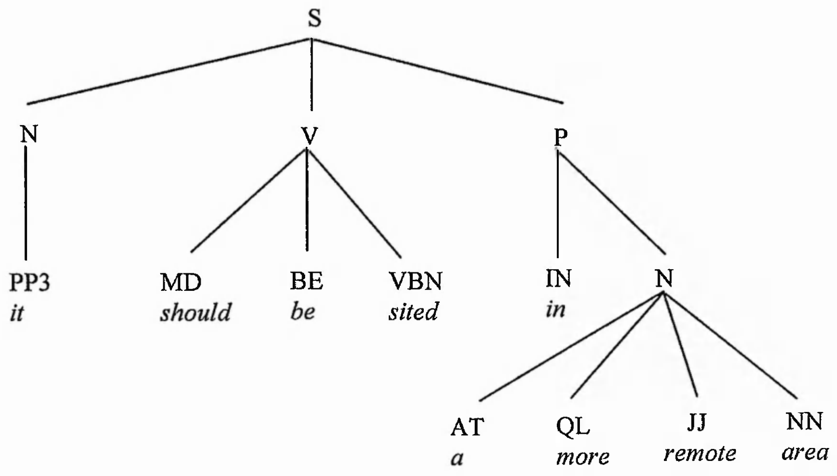


Figure 7.18 : (a) The target parse for *It should be sited in a more remote area.* (b) The parser's actual parse for the sentence.

Although the parser was able to preserve information in order to process the embedded genitive phrase in figure 7.17, serious limitations are encountered when the parser expected to preserve more than one word. For example, consider the following non-LPC sentences :

(S7.11) the dogs who chased the cat are hungry.

(S7.12) I saw a boy who dropped the model cry

In sentence S7.11, the number agreement between the subject noun phrase, *the dogs,* and the auxiliary verb, *are*, can only be established when the parser has processed the embedded clause, *who chased the cat.* As the parser processes the sentence from Right-to-Left, it must store *are hungry* as context whilst it processes the embedded finite relative clause. However, the parser makes an incorrect structural preference to attach *are hungry* to the embedded clause as illustrated in figure 7.19.

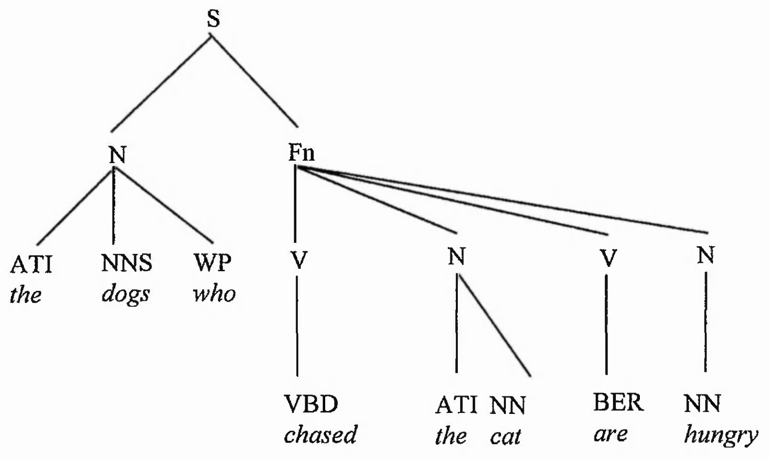


Figure 7.19 : The parser illustrating that it is unable to correctly segment the embedded clause for sentence S7.11.

It also indicates that parser makes similar structural attachment preferences to those defined in Frazier's [114] Minimal Attachment and Late Closure principles in that rather than preserving the auxiliary verb phrase and noun phrase it is attaching them both to the nearest clause. Minimal Attachment preferences are consistently made by the parser during test experiments and account for numerous structural errors.

Clearly, using the principles defined in the LPC, the embedded clause, *who chased the cat*, should be dominated by *Fr* to represent a finite relative clause (the word *who* is usually associated with the first word of a finite relative clause in the LPC). The auxiliary verb phrase, *are*, and the noun phrase, *hungry*, should be attached to the *S* node at the same level as the finite relative clause and the subject noun phrase. This would yield the following LPC structure :

> **[S [N** the_**ATI** dogs_**NNS N] [Fr [Nq** who_**WP Nq] [V** chased_**VBD V] [N** the_**ATI** cat_**NN N] Fr] [V** are_**BER V] [N** hungry_**NN N] S]**

Although the parser is unable to attach *who* to a finite relative clause in sentence S7.11, it is able to correctly attach *who* to the finite relative clause in sentence S7.12 as illustrated in figure 7.20. The parser is able to segment the sentence into two clauses. However, it is unable to attach the verb phrase, *cry*, at the same level in the tree as the main clause, *I saw a boy*, as the parser again prefers to attach it to the 'nearest' or current clause it is processing.

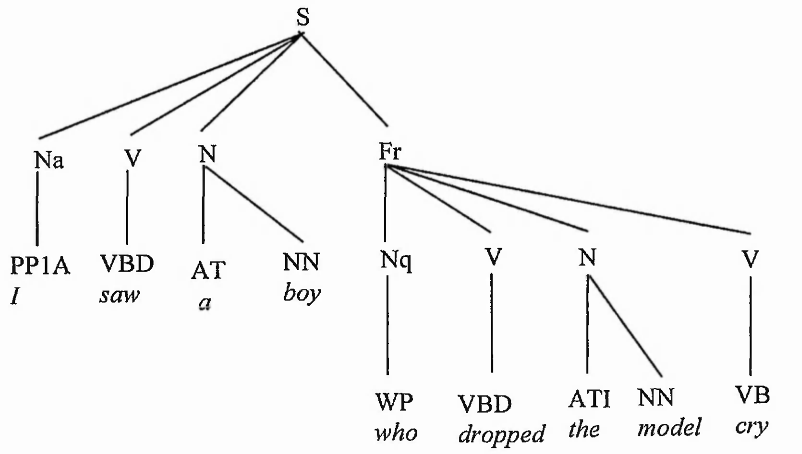


Figure 7.20 : The parser correctly attaches *who* to the embedded clause.

The main reason for the parser failing to correctly segment embedded clauses and establish long-distance dependencies is that although there are embedded phrases within the training data, there are no embedded clauses that requires the parser to preserve information in this way. Sentence structures in the training data that do contain *Fn, Fr* or *Fa* clauses contain 'right-heavy' embedded clauses that do not require information to the right of the clause to be attached at the same level as the clause node.

For the parser to be reasonably be expected to process embedded clauses it must be trained with such examples. However, the parser has proven that it is able to correctly process embedded phrases provided that the number of words/tags to be preserved is not greater than one. Also, the parser has demonstrated its robustness when processing unfamiliar embedded phrases. For example, when presented with a noun phrase containing an embedded adjective phrase the parser was unable to correctly group the adjectives into an adjective phrase. The parser alternatively grouped the adjectives and nouns to form a simpler noun phrase. Although the structure differed from the target LPC parse, it presented a semantically plausible structure. This generalisation property presents an inherent advantage of using connectionist networks for parsing natural language.

7.3.5. Mismatches Due to Recogniser Error

The mismatching parses made by the parser in previous sections have been caused by the failure of the RLD and LRD networks. However, this section discusses the mismatching parses that have been caused solely by the Recogniser network to assess the affect these errors have on the resulting semantic interpretation of the syntactic structure. The low coverage of phrases in the Recogniser network's training data is made apparent by the recognition error made by the parser for the mismatching parse illustrated in figure 7.21(b).

It can be seen from figure 7.21(a) that *were* must be attached to the constituent tag for a simple verb phrase, *V*. However, in the Recogniser network's training data there are only seven instances where this association is made and in each instance the context is obviously different from that in the LPC structure shown in figure 7.21(a). The parser is committed to attaching *Vr* to the past progressive tense verb, *were*. A *Vr* phrase is a verb phrase that allows the parser to process common inverted constructs whereby the position of an auxiliary verb and a subject noun phrase is swapped (e.g., *Has Jon finished?*). The error made by the Recogniser network is significant in that each of the 8 instances of *Vr* phrases found within its training data is preceded by *Vo* and *N* phrases, yet it still classifies *were* as a *Vr* phrase.

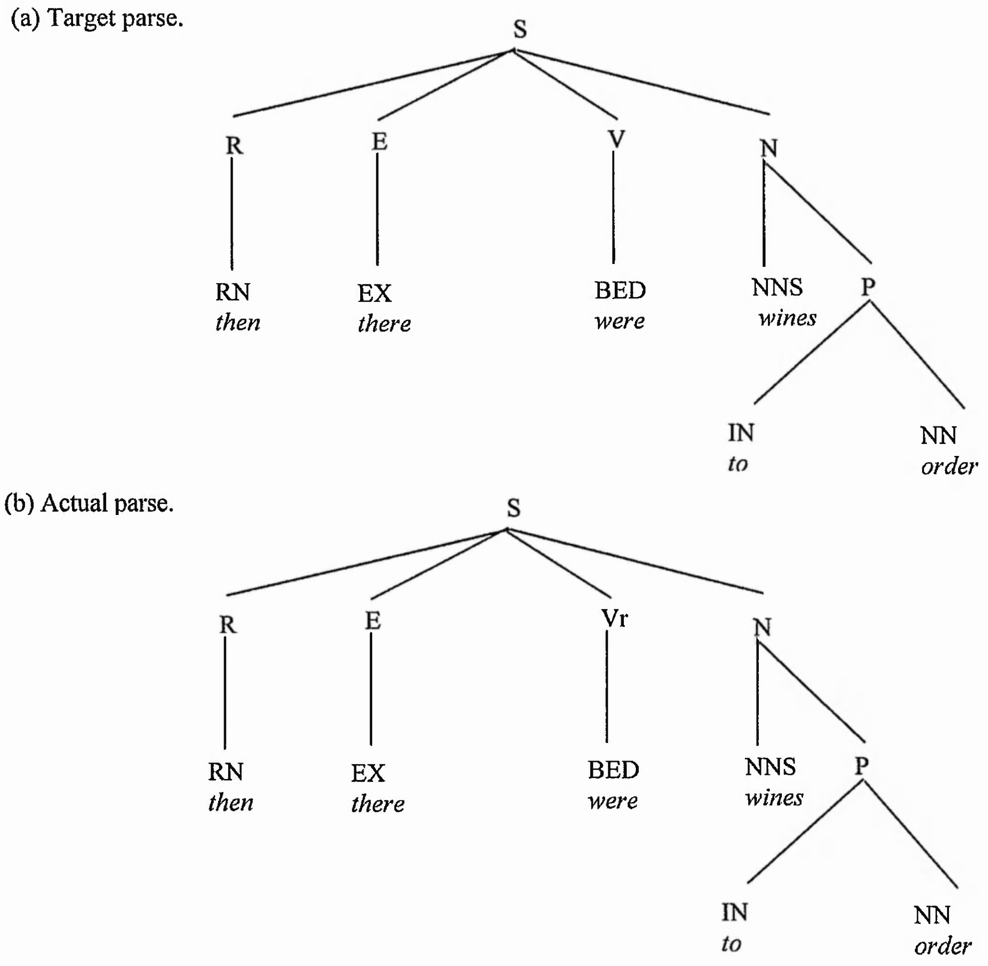


Figure 7.21 : (a) The target parse for *then there were wines to order.* (b) The parser's actual parse for the sentence.

The mismatching parses illustrated in figures 7.22 and 7.23 are both caused by simple errors made the Recogniser network. Both parses are very close to their target parse and demonstrates that the parser is able to 'recover' from minor recognition failures and parse according to the grammatical constraints learnt from the LPC.

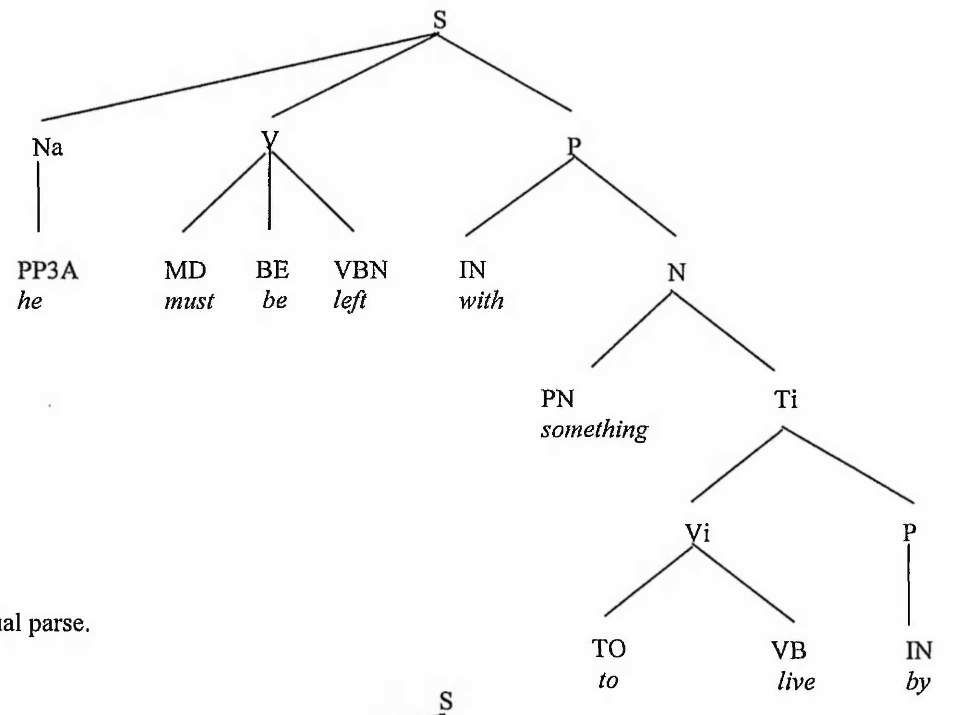
As shown in figures 7.22(a) and (b), the mismatching parse for the sentence, *he must be left with something to live by*, is caused by a single error made by the Recogniser network during the first phrase reduction. Fortunately, this error did not cause any further errors in the parsing process. The sentence structure consists of the subject noun phrase, *I*; a verb phrase consisting of the modal auxiliary verbs, *must be*, acting as pre-modifiers to the main past tense verb, *left*; and a main preposition phrase *P*. The main preposition phrase begins with the preposition, *with*, and is supported by a noun phrase consisting of the nominal pronoun, *something* and a *to*-infinitive clause (*Ti*). The *to*-infinitive clause consists of a *to*-infinitive verb phrase (*Vi*) and a prepositional phrase. The *Vi* phrase groups the words *to live* and the prepositional phrase dominates the preposition, *by*. It is on this preposition the Recogniser network fails to recognise it as belonging to a standard *P* phrase group. As the parser parses from Right-to-Left, this is the first phrase segmented by the phrase segmentation module. The Recogniser network incorrectly recognises *by* as belonging to a preposition phrase beginning with the preposition *of*.

The minor recognition error made by the parser for the sentence, *they may have to call up the reinforcement of the common market*, (figure 7.23) again did not cause any further errors in the parsing process. The sentence structure consists of three main parts : a subject noun phrase, a main verb phrase and a *to*-infinitive clause.

The subject noun phrase consists of the nominative personal pronoun (third person plural) *they*. The main verb phrase contains the modal auxiliary verb *may* pre-modifying the main verb, *have*. The word *have* is being used as the main verb to express obligation. Further information is provided by the *to*-infinitive clause (*Ti*). The *Ti* clause consists of three main parts : a *to*-infinitive verb phrase, an adverbial phrase and an object noun phrase. The *to*-infinitive verb phrase (*Vi*) contains the word, *to*, and the verb, *call*. The adverbial phrase consists of the single adverb, *up*. It is at this point where the parser makes the recognition error, classifying *up* as a prepositional phrase rather than an adverbial phrase. A plausible explanation for this behaviour is that preposition phrases can function as adverbial phrases therefore the parser is confusing the adverbial particle, *up_RP* with a prepositional phrase due to its position.

However, the recognition error does not to hinder the parsing process as it is able to continue parsing in-line with its target LPC parse. The object noun phrase is composed of the definite article, *the*, the main noun, *reinforcement*, and a post-modifying prepositional phrase.

(a) Target parse.



(b) Actual parse.

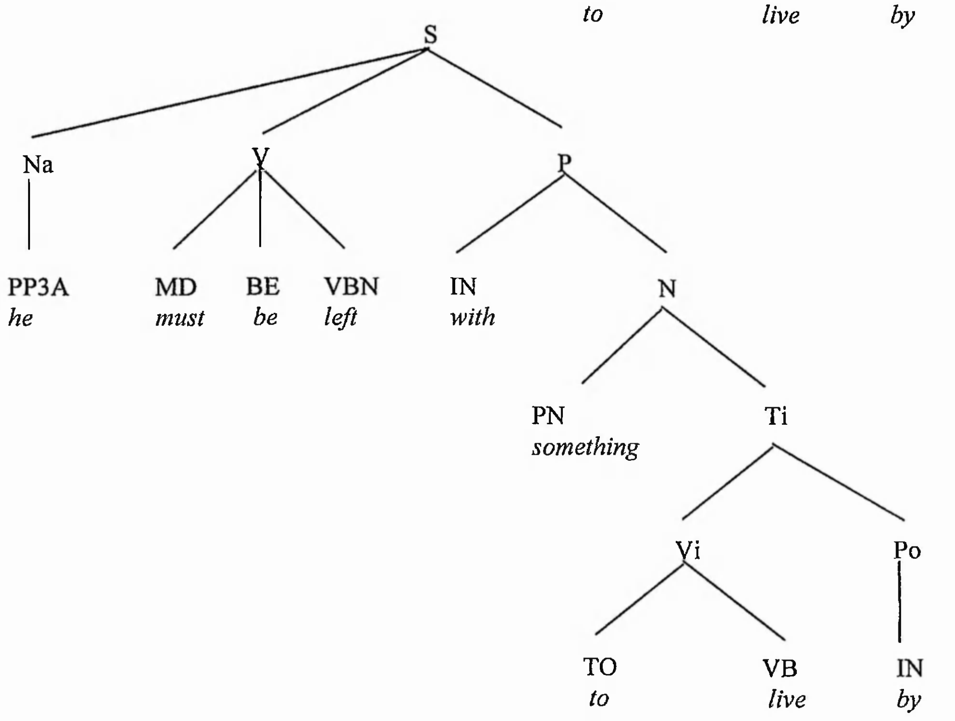


Figure 7.22 : (a) The target parse for *he must be left with something to live by*. (b) The parser's actual parse for the sentence.

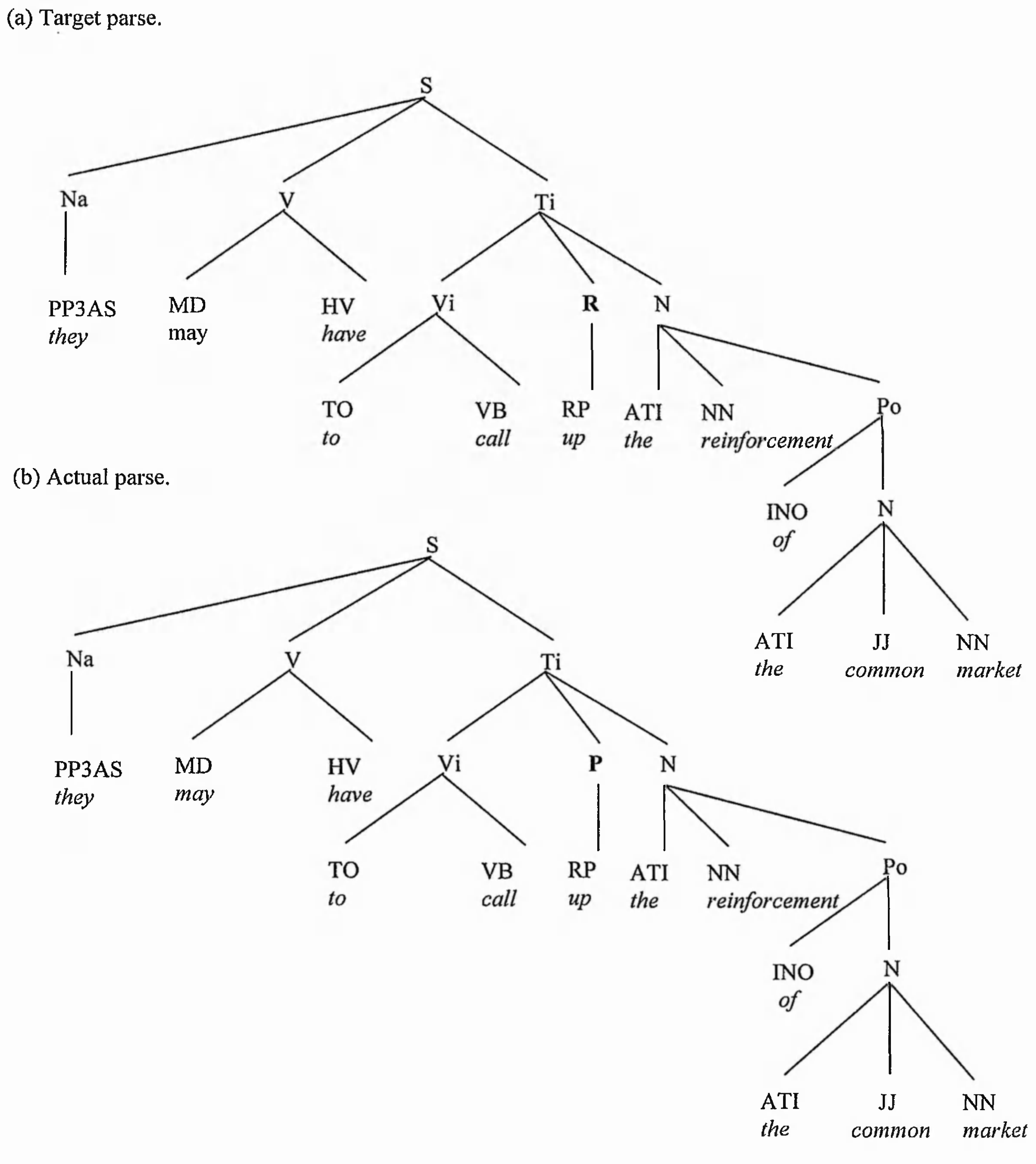


Figure 7.23 : (a) The target parse for *they may have to call up the reinforcement of the common market.* (b) The parser's actual parse for the sentence.

The post-modifying prepositional phrase begins with the preposition, *of* followed by an object noun phrase. This object noun phrase consists of a similar structure to that found in figure 7.18. However, the noun phrase in figure 7.18(a) possessed a qualifier pre-modifying the adjective causing the instantiation of an embedded adjective phrase within the noun phrase. The parser ignored the significance of the qualifier in figure 7.18(b) and simply grouped the definite article, qualifier, adjective and noun into a noun phrase group. The reason for this can be seen in the right-most object noun phrase, *the common market*, found in figure 7.23. When there are no modifiers to the adjective then the adjective is simply grouped with its neighbouring constituents to form the noun phrase. These structures are significantly more common than those noun phrases that contain embedded adjective phrases, therefore it is natural for the parser to generalise in the manner illustrated in figure 7.18(b).

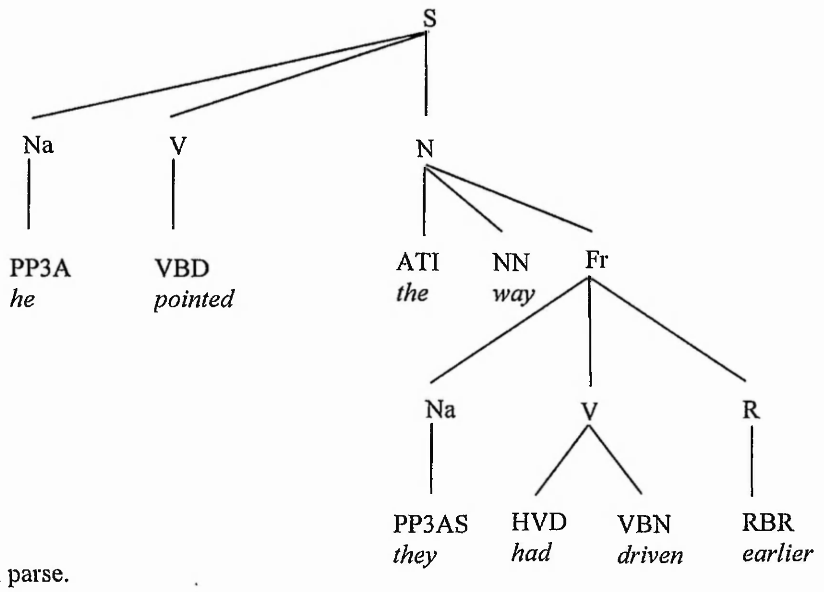
The parse structures in 7.22(b) and 7.23(b) again demonstrate that the parser can competently process *to*-infinitive clauses. Even though the Recogniser network has learnt to recognise 216 instances of standard prepositional phrases and 81 instances of prepositional phrases beginning with *of* during the training phase, these close though incorrect parses illustrates the need to increase the generalisation of the Recogniser network by increasing the level of language coverage or improving the position invariance of the input pattern. This will be discussed further in Chapter 8.

The next structural comparison in this assessment demonstrates the detrimental affect an error made by the Recogniser network can have on the whole parsing process. Figure 7.24(a) shows the LPC parse for the sentence, *he pointed the way they had driven earlier*. The sentence structure is composed of a subject noun phrase (*Na*) consisting of a singular third person personal pronoun; a main verb phrase (*V*) consisting of the past tense verb *pointed*, and a noun phrase (*N*) that contains a relative clause (*Fr*) acting as a post-modifier to the main noun, *way*. The *Fr* clause consists of a subject noun phrase, verb phrase and adverbial phrase. As the parser processes input words from Right-to-Left, it will first process this clause and it is during this stage that the Recogniser network makes the critical error of recognising the third person plural pronoun *they* as belonging to a noun phrase beginning with a WH-word i.e., *Nq* rather than a subject noun phrase group. This error is significant as the *Nq* always belongs to a finite nominal clause (*Fn*), therefore the parser incorrectly creates a *Fn* clause later in the parsing process as illustrated by figure 7.24(b).

The recognition error also causes the phrase segmentation modules to fail as the main noun, *way*, is subsequently attached to the *Fn* clause. Although the structure produced by the parser for the main noun phrase is incorrect, it is envisaged that via extending the language coverage of the Recogniser network further syntactic errors can be significantly reduced.

However, this section has shown that errors made by the Recogniser network does not necessarily result in an syntactically and semantically incorrect structure. Although the Recogniser network confused adverbs with prepositions, this error is understandable in that adverbial and prepositional phrases can function in the same position within a sentence.

(a) Target parse.



(b) Actual parse.

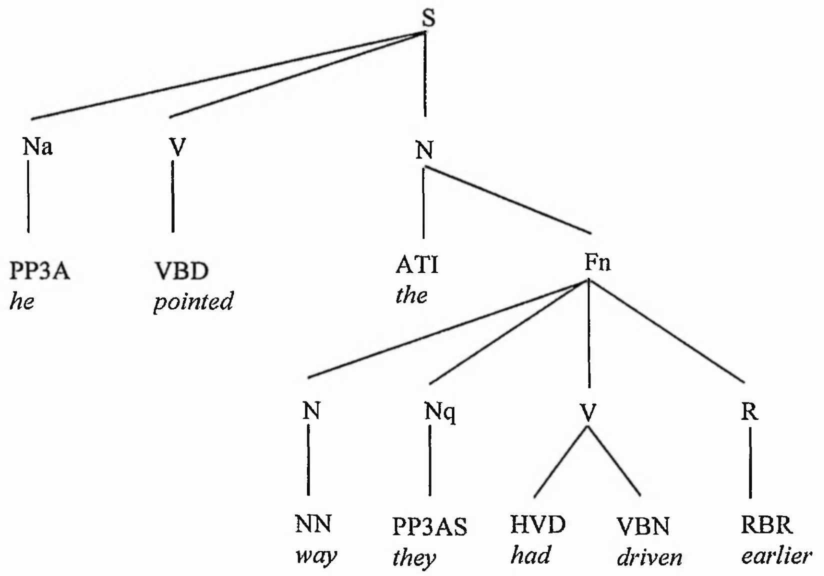


Figure 7.24 : (a) The target parse for *he pointed the way they had driven earlier.* (b) The parser's actual parse for the sentence.

The general nature of the recognition errors are minor in that the Recogniser network tends to misclassify the correct variant of a phrase rather than a gross mismatch. This usually preserves the meaning and also allows the segmentation modules to continue processing in the desired manner (also demonstrating the resilience of the RLD and LRD networks). However, the errors made do represent a limitation of the phrase invariant recognition technique.

7.3.6. Mismatches Due to Punctuation Removal

Punctuation has been removed from all the sentences processed by the parser during training and testing. The consequence of removing punctuation is apparent in the parser's attempt at matching the LPC parse for the sentence *I must; therefore I can* shown in figure 7.25. The LPC sentence structure illustrated in figure 7.25(a) shows that the sentence consists of two main clauses.

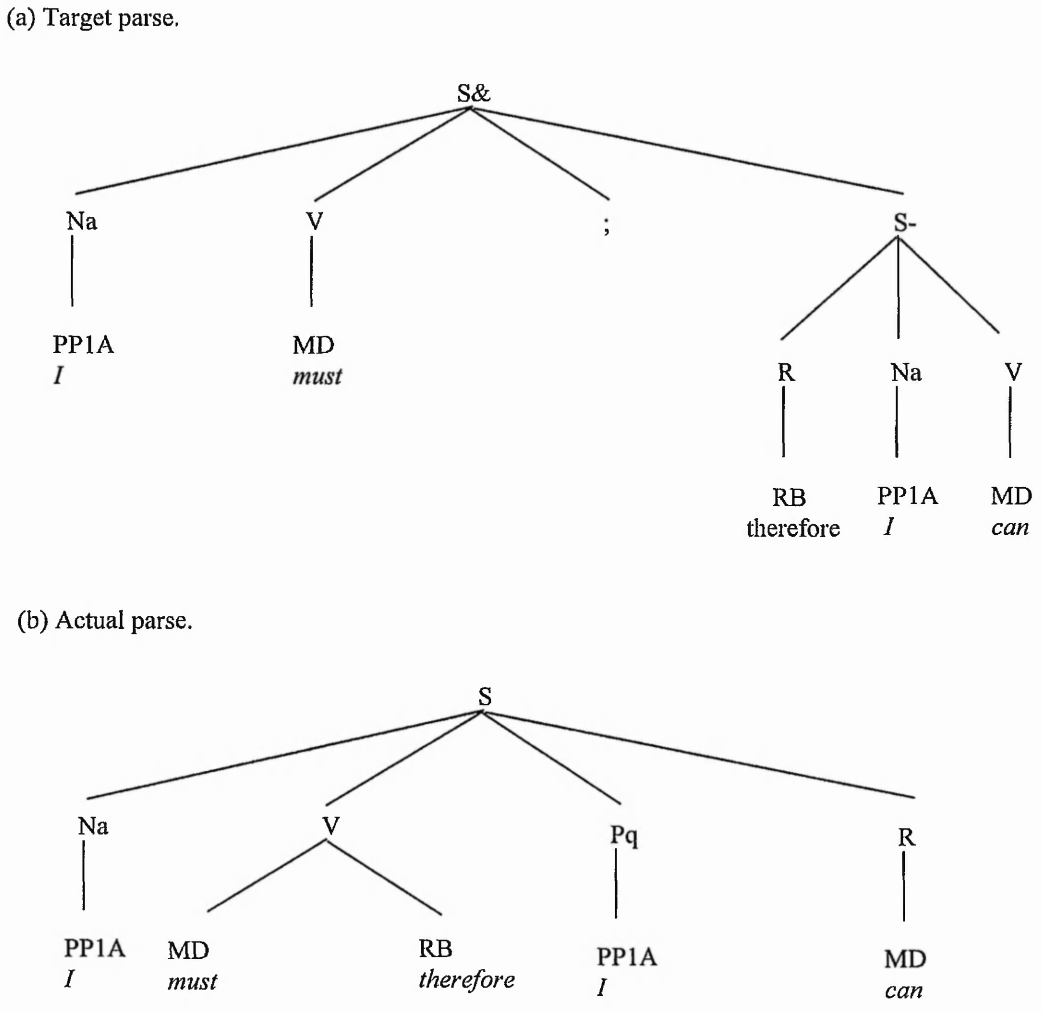


Figure 7.25 : (a) The target parse for *I must ; therefore I can.* (b) The parser's actual parse for the sentence.

Rather than being joined by a coordinating conjunction, the clauses are separated by a semi-colon therefore the second clause is dominated by the constituent tag for a 'coordinated sentence without a coordinating conjunction' i.e., *S-*. As the parser has no notion of punctuation it is acceptable for the parser to fail on sentences that rely on punctuation to separate clauses. It can be seen from figure 7.25(b) that the parser is unable to separate the sentence into two main clauses and makes serious classification errors. The right-most modal auxiliary verb, *can*, is grouped as an adverbial phrase and is therefore dominated by *R* in the parse tree. This has a knock-on effect on the classification of the personal pronoun, *I*. The parser is unable to associate the pronoun with *Na* in the context of an adverbial phrase immediately succeeding it and a general adverb immediately preceding it. Due to the previous incorrect classification and the missing punctuation, the parser cannot recognise *I* as functioning within a clause. In this context, it therefore recognises *I* as incorrectly belonging to a prepositional phrase with a wh-word (*Pq*). These errors, made by the Recogniser network, cause the phrase segmentation modules to fail. The modal auxiliary, *must*, functions as the main verb and should alone be associated with *V*, however the parser groups *must therefore* into the *V* phrase. The adverb, *therefore*, incorrectly functions as a post-modifier to the auxiliary verb. A consolation is that the parser is able to correctly recognise the subject noun phrase at the beginning of the sentence.

The LPC sentence in figure 7.25 is the only sentence in the constrained test sample that contains a semi-colon. After a further analysis of matching and mismatching parses, the removal of commas, colons and quotes did not appear to affect the performance of the parser (see Appendix K and L). Although the occurrences of semi-colons in the LPC is rare, further research to assess the significance of punctuation is required.

7.4. Comparisons With Other LOB Parsers

At present there is no known parser that can accurately parse the entire LOB corpus. Although a direct comparison with other LOB-based parsing systems (symbolic, connectionist or statistical) cannot be made due to different annotations, corpus samples and statistical measures used, one can compare the model in terms of precision and coverage i.e., the size and composition of training and test sentences used together with the complexity of LOB sentence structures that can be processed by the system[24].

Sampson et al's [94] HMM and simulated annealing-based system, APRIL, attempted to parse sentences from the LOB corpus after 'training' the system on a database of manually parsed English text, also extracted from the LOB corpus. The method in which Sampson compares APRIL's resulting parse with the target parse, is that for each input word he compares the chains of non-terminal symbols between the terminal nodes and the root (S) node in order to compute the number of non-terminals which match each other and occur in the same order. An average is then made over all input words. The consequence of averaging over words means that high-level non-terminals (those nearer the root node) dominate many words and thus contribute more to the overall cost than lower-level non-terminal symbols. The resulting cost is a percentage of how close the actual parse is to the target parse. Although Sampson reported a 75.3% accuracy rate, this was only for 50 test sentences and no information was given as to the structural composition of this sample.

Atwell and Pocock [149] developed an HMM parser that used pre-parsed sentences from the Lancaster Treebank

[24] However, it must also be noted that after assessing current research on corpus-based parsing models, much focus has moved towards annotating more substantial corpora such as the Brown Corpus [144], Penn Treebank [145] and the British National Corpus (BNC) [178].

(based on the LOB corpus) as training data. The smallest training set consisted of 30 pre-parsed sentences, and the largest consisted of 292 pre-parsed sentences. Once trained, attempts were made to parse the training data. Atwall reported disappointing results. It could only parse 19 of the 30 pre-parsed training sentences, and increasing the training set reduced the accuracy further. This was apparently due to further ambiguity being added to the training set. However, Atwall did report better levels of accuracy for sentences not in the training data (i.e., generalisation), and stated that this accuracy increased as the number of training samples increased. Many problems were encountered in terms of testing times, and little information was given as to how similar the test sentences were to the training data. The model itself was computationally inadequate and insufficient to process realistic subsets of natural language.

De Marcken [176] developed a rule-based parser that attempts to rapidly parse simple phrase structures contained within sentences in the LOB corpus rather than parsing entire sentences i.e., accuracy is favoured over completeness. Three tables of context-free rules are used to build phrases in a bottom-up fashion and each grammar rule contains a category, starting and ending locations and a collection of feature and tree information. The first is the 'rule-action table' which specifies what action a rule in a certain state should take if it encounters a phrase with a given set of categories and features. The second is the 'single-phrase-action' table that contains attachment rules which determine whether adjoining phrases can be combined to form higher level linguistic structures. Finally, a 'paired-phrase-action' table is used to specify actions to take if two certain phrases adjoin each other. A symbolic queue holds each input item and when a phrase is popped off rules are examined to assess whether the phrase can be attached to another phrase or whether to create a new rule. For a rule to fire on a phrase, the rule must be at the starting position of the phrase. Actions that can be taken by the parser are : shift; create a phrase from all phrases so far; or both, creating a phrase and continuing the rule to recognise a larger phrase. New phrases and rules are placed on the queue only after all actions resulting from a given pop of the queue have been taken. The ordering of their placement has a dramatic effect on how the parse proceeds. New rules are placed higher than new phrases. Although no statistical measure of accuracy is provided, the performance of the system is judged in terms of the number of phrases it can create in response to 624 randomly chosen sentences. However, the correctness of these phrases could not be determined and the actual grammatical formalism was not described. De Marcken conceded that the parser was unable to perform complete parses for many of the 624 sentences and the system was unable to process complex sentences that contained coordinating or subordinating conjunctions.

Wyard and Nightingale [177] defined a connectionist system, called HODYNE, that simply accepts or rejects tuples of tags in the LOB corpus. It performs no structural analysis of the sentences. HODYNE is essentially a single layer high-order network containing a variable number of input units and two output units, one output unit for 'yes' (accept) and one for 'no' (reject). The number of input units is generally large and is dependent on the lengths of the tuples selected and the number of syntactic categories used. For processing the LOB corpus, the number of input units is dependent on the maximum number tuples in a sentence of the LOB sample (i.e., one unit for each tuple). The output units simply calculate the weighted sum of their inputs without the squashing function or a threshold. The unit with the highest activation is selected. The training procedure used is similar that defined for linear networks by Rosenblatt

[1]. All sentences in the LOB are assumed to be grammatical and thus a number of LOB sentences are made ungrammatical to present HODYNE with both positive and negative examples during training. Three training and test sets were used for the entire LOB corpus based on sentence length, one each for sentences between 1 and 20 words, 1 and 30 words and 1 and 100 words. The maximum number of input units used by HODYNE after training was reported to be 15,139 units. HODYNE had learnt between 80-85% of each training set. Wyard and Nightingale reported a recognition rate of 79.5% for the entire LOB corpus. However, HODYNE performed worse than a comparable pairwise statistical model. The statistical model simply consisted of a 153*153 matrix (as there are 153 tags in the LOB corpus) containing the number of times each possible tag-pair had occurred in the training set of only grammatical strings. The LOB sentence is judged to be grammatical if its value exceeded a threshold. This model achieved a recognition rate of 81.4% for the entire LOB corpus and consistently outperformed HODYNE in similar experiments. Although the recognition rate of HODYNE is impressive it is questionable as to its advantage over traditional pairwise statistical models and also to its usefulness as a module within a natural language understanding system. HODYNE provides no information about how elements within a sentence relate to one another and thus does not perform syntactic analysis per se. It is thus difficult to assess how such a model could process the syntactic problems and dependencies associated with language and how it could also interface or integrate with a semantic module.

In contrast to the above LOB-based parsing systems, the modular hybrid system developed in this work holds the following advantages :

- Wide language coverage. The parser learnt to parse 96.9% of its 654 training sentences, matching 62.2% with its LPC target parse. It generalised to 93.4% of its 687 test sentences, correctly matching 49% with its target LPC parse. However, 63.6% of the 305 mismatching parses contained between 1 and 3 module errors with 75 of the mismatching parses having only 1 module error. The output from the parser for these sentence structures was thus typically 'close' to the LPC parse.

- The parser is consistently able to perform constituent coordination to process complex sentence structures.

- An incremental structural representation of the full sentence structure is output by the parser thus allowing a semantic module to process it further or for semantic information to be combined within the parser itself.

- The generality of the architecture loosely couples the grammatical formalism with the parser thus it is not predisposed to a particular grammatical formalism. This widens the scope of the parser and its future uses.

7.5. Summary

This chapter has presented the test results of the parser's performance on sentences belonging to the training and test samples. It demonstrated that the parser has learnt a large coverage of linguistic features from the LPC and is consequently able to parse according to these constraints. The parser matched 62.2% of the LPC parses for the training sample but only 49% of the LPC parses for the test sample. However, 63.6% of the 305 mismatching parses contained between 1 and 3 module errors thus the output of the parser for these sentence structures were typically close to the LPC parse and in some cases where there was syntactic ambiguity, the result was a valid parse in the absence of semantic information.

A subset of 10 matching parses and 10 mismatching parses was used to assess the behavioural characteristics of the parser. Also, a number of LPC sentence structures were manually modified and some sentence structures were created to test the parser on specific syntactic constructions. The assessment of these sentence structures revealed the following characteristics and computational adequacies :

- The parser is able to parse simple sentence structures that consist of : noun phrases (including subject nouns), finite and non-finite verb phrases (including auxiliary verbs and negatives), preposition phrases, adverbial phrases and adjective phrases.

- The parser parses compound and complex sentence structures by performing constituent coordination. It is able to perform coordination using coordinating conjunctions and subordinating conjunctions. The parser is also capable of performing word-level coordination.

- Although some incorrect preposition attachments were made, the parser is able to provide semantically plausible alternative structural attachments. This is an impressive characteristic considering it has no access to semantic information.

- Embedded phrases are correctly processed by the parser provided that the number of words/tags to be remembered is not greater than one.

- When presented with unfamiliar context and phrases the parser is able to generalise to produce a 'coarser' syntactic grouping that preserves the semantic interpretation of the sentence.

- to-infinitive clauses (*Ti*) are correctly and consistently associated with non-finite verb phrases (*Vi*). The different uses of the word *to* is also recognised by the parser i.e., when it is used as a preposition or as part of a verb phrase.

- The parser is able to distinguish between the types of pronouns that can be used as a subject noun phrase and those that cannot.

A number of limitations emerged from the analysis of the test results, the main inadequacies being :

- The parser behaves in a strictly deterministic manner and there is no mechanism to recover from failures made by either the segmentation or recognition module. If either module fails then the generalisations made by the other module are crucial to the success of the parse.

- If there are more than two main clauses to be coordinated then the number of structural errors made by the parser increases. Also, the inclusion of pre-modifiers to a subordinate clause can affect the behaviour of the parser as demonstrated with the addition of qualifiers to compound noun phrase structures.

- Problems are encountered when the parser is expected to associate a past-participle verb phrase (*Vn*) with a past-participle clause (*Tn*). The parser enables a *Vn* to function as a *Tn* clause thus ignoring *Tn* clauses altogether.

- Embedded clauses cannot be correctly processed unless there is only one word/tag to the right of an embedded phrase.

However, the inability to process embedded clauses is due to the poor representation of such structures in the training data. Similarly, the representation of *Tn* clauses in the training data is also poor. For example, there are only 16 instances of *Tn* clauses in the training data compared to 59 instances of *Ti* clauses. The magnitude of recognition errors could be significantly reduced by increasing the coverage of clauses and phrases in the Recogniser network's training data.

Overall, the corpus-based connectionist parsing model is a robust parser that exhibits high levels of generalisations to compensate for errors made by other connectionist modules and for changes in context. It integrates the advantages of connectionist parallel processing with those of traditional symbolic representations and manipulation to parse a realistic subset of natural occurring English text, based upon the linguistic constraints it automatically acquired during the training phase.

Chapter 8

Conclusions

8.1. Introduction

This dissertation has described a natural language parsing architecture that is trainable and thus able to automatically learn linguistic constraints directly from annotated text corpora without the explicit use of grammar rules. The model is a new type of hybrid architecture that relies on the integration of current symbolic representation and manipulation techniques with a novel approach to shift-reduce parsing using modular connectionist architectures. The method of phrase segmentation and recognition developed provides a powerful technique for sequentially processing long and complex sentences. The symbolic components allow information to be stored in an easily interpretable and manipulable manner and provides the basis for organising the parse. This method of integration also allows the automatic parsing of naturally occurring English text without the use of phrase structure rules thus satisfying the main aim of the thesis.

The parsing model developed is adaptable to the grammatical framework and thus to the underlying linguistic theory used to annotate the corpus. It relies upon the connectionist modules of the system to learn the linguistic constraints, embedded within the corpus, during the training phase. Consequently, unlike a number of previous localist and distributed connectionist parsers, there is no constraint imposed on the type of grammatical framework or linguistic theory used. This also satisfies the second main objective of the thesis. Additionally, since the input symbols are processed sequentially, the length of the input sentence is theoretically unrestricted thus providing another advantage over previous connectionist parsers. This novel integration of connectionist and symbolic techniques yields a robust shift-reduce natural language parsing mechanism that supports massively parallel processing. The results presented in Chapter 7 also demonstrate that the works stated aims have indeed been achieved.

The approach to parsing developed here contributes to the field of Artificial Intelligence in two fundamental respects : as a configurable[25] model for the domain-independent learning of syntactic structure, and as a step toward constructing a natural language understanding system that is able to process unconstrained natural language texts. More broadly, the research makes a number of contributions to the development of techniques for achieving intelligent behaviour using modular connectionist networks that possess distributed representations of constraints.

The following sections present a number of specific contributions made to the field of connectionist parsing and a number of improvements that can be made to the parsing model and thereby suggests further work for the research.

[25] Configurable in that it can be configured according to the linguistic theory or grammatical formalism used.

8.2. Contributions

The model integrates the advantages of both connectionist and symbolic processing techniques to formulate a novel hybrid shift-reduce parsing system. The system exhibits interesting linguistic behaviour and makes a number of important contributions to the field of connectionist parsing that need to be considered.

8.2.1. Modular Hybrid Architecture

In contrast to previous parsers implemented within a connectionist framework, the grammatical knowledge of the hybrid parsing model is not constrained to one type of formalism. Although sentences extracted from the LPC were used as the basis for training and test data, the architecture is equally suited to learning its grammatical knowledge from a corpus that has been annotated using an alternative linguistic theory such as HPSG or GB-Theory. For example, if a large corpus of text was processed by a number of human linguistic experts using a common syntactic notation (rather than using some other artificial parser based on a specific linguistic theory) then the hybrid architecture is still suited to learning the resulting linguistic constraints embedded within the corpus. An advancement here is therefore a trainable parsing system that is able to learn syntactic structure from large bodies of naturally occurring sentence examples that have been annotated with syntactic word tags and constituent tags to denote syntactic structure. It also demonstrates the potential of connectionist architectures to be applied to large-scale parsing tasks.

The modular hybrid architecture developed is motivated by the results of the preliminary investigations presented in Chapter 3. The aim of the preliminary investigations was to decide on the most appropriate role of feed-forward MLP networks within the parsing process. During the investigations, limitations were encountered when attempting to use a single feed-forward MLP network to identify and validate phrases generated from a large context-free grammar. MLP architectures used in this way are tightly coupled with the grammar and amendments to the grammar results in corresponding changes to the architecture and the training and test data. Feed-forward MLP architectures are also limited to either using fixed length temporal input windows or using a predetermined maximum sentence length. This is reflected in the limitations found in connectionist parsers that are based solely on feed-forward MLP architectures [2b,4,11,12,55,130,131]. This led to adopting a modular connectionist approach to parsing using a mixture of feed-forward MLP and recurrent MLP connectionist networks, each with its own dedicated task thus dividing the overall parsing problem in to simpler tasks.

Parsing within a shift-reduce framework was divided into two fundamental tasks : phrase segmentation and phrase recognition. Phrase segmentation was further decomposed into two sub-tasks : right-to-left phrase delimiting and left-to-right phrase delimiting. The task of the right-to-left delimiter (RLD) is to recognise the beginning of a syntactic phrase, and the left-to-right delimiter (LRD) to recognise the end of a syntactic phrase. The number of inputs processed by either delimiter before the beginning or end of a phrase is encountered is variable and not known *a prior*, therefore it was essential that the connectionist network assigned to either task was able to sequentially process linguistic input. As mentioned previously, the SRN has proved very successful at temporal processing and enabling

MLP networks to establish temporal relationships over sequential inputs. However, SRNs have limited memory and this is particularly problematic when there is no target output vector for several time steps. This results in the loss of earlier input information. TASRNs were considered to overcome this problem. They ensure that meaningful targets are available at every processing stage and thereby minimise the loss of previous inputs. The TASRN compared favourably over the SRN for both the RLD and LRD delimiter tasks. TASRNs allow the advantage of sequentially processing linguistic input within connectionist networks and reduces the memory limitations associated with SRNs.

Feed-forward MLP networks trained with Back-propagation proved adequate for performing simple phrase recognition tasks. However, as illustrated in Chapter 3, problems are encountered when expecting a single MLP to recognise the position of the phrase and to reject invalid phrases. This motivated the decision to employ a feed-forward MLP network trained with Back-propagation to act as a simple phrase structure recogniser whose output activations are subject to a 'nearest match' computation to find the closest valid phrase for a group of input symbols. This removed the problem of phrase validation. The position of the phrase within the MLP network's input window is also fixed thus eliminating the complexities associated with position variant phrase recognition. Context from the left-hand side *and* right-hand side of the extracted phrase was made available to the MLP network to allow essential context for recognition.

The manner in which the parsing problem has been decomposed and modularised aids generalisation. For example, if one connectionist module makes an incorrect prediction then the other connectionist modules are often able to successfully generalise in order to continue the structural processing without complete failure. This 'graceful degradation' in performance is characterised by the small number of parse failures and high number of structural mismatches reported for the test data.

The inadequacies associated with existing connectionist distributed representations of hierarchical structure (e.g., RAAM) have severely limited the success of other modular connectionist parsers [62, 63, 126, 127]. Although the development of connectionist variable binding techniques and representation of complex structure have provided valuable insight into how symbolic computation and storage can be performed with localist and distributed architectures, much research still needs to be done to improve memory capacity and generalisation capabilities. At present, well-established symbolic techniques still provide an easily manipulable and interpretive method of dynamic variable binding and storage for parsing purposes. Consequently, three core symbolic structures were employed to enable the hybrid model to be applied to large language domains. A symbolic linked list is used to store tag information, a stack structure is used as a Parse-Stack to store parse state information and the resulting parse tree, and finally a Last-In-First-Out stack structure is used as an Input-stack to store the current input state. These symbolic components of the parser reflect the hybrid nature of the parser with subsymbolic and parallel processing within the connectionist networks and communication at a symbolic level between them.

In contrast with the traditional left-to-right processing approach, the parser processes each sentence from right-to-left.

The advantage of parsing in a right-associative manner is that the complicated structures that cause structural ambiguity (locally) are commonly located to the right of the sentence (within the predicate) rather than to the left where subject noun phrases are commonly located. This means that the ambiguity can be resolved earlier. Also, there are no construction specific heuristics built into the parser to guide its decisions and thus the attachment preferences made by the parser are based on those prevalent in the LPC sample used for training the parser.

8.2.2. Language Coverage and Generalisation

Connectionist parsers that are restricted to using linguistic- or semantic-based grammars as the basis of their grammatical framework inherently risk low coverage because the rule-based constraints limit the number of acceptable grammatical structures. The corpus-based approach taken in this research implements a data-orientated framework that uses a training phase to essentially determine the search procedure that produces best results on the training corpus. The LPC contains a variety of naturally occurring sentences from which structural information can be extracted. This presents an advantage over connectionist parsers that are trained with a hand-crafted grammar as hand-crafted grammars either allow too many structures that rarely naturally occur, or they are too restrictive and do not allow structures that do naturally occur.

The parser successfully parsed 96.9% of the 654 training sentences and managed to match 62.2% of them with the equivalent LPC parse. The parser failed to parse only 20 (3.1%) of the 654 sentences. Although a thorough large-scale linguistic analysis of the parser's behaviour is beyond the scope of the dissertation, the resulting level of the parser's language coverage was assessed, using the constrained test sample. The parser successfully matched 337 of the 687 test sentences with the corresponding LPC structures i.e., 49%. It also produced alternative linguistic structures for 305 of the 687 test sentences i.e., 44.4%. These mismatching parses indicate that a number of constraints were not represented in the training data and some may be contradicted elsewhere in the corpus, particularly if semantic information was used to determine the parse. However, it is also a significant indication that if a connectionist module fails somewhere within the parsing process it does not result in catastrophic failure. For example, 63.6% of the 305 mismatching parses contained between 1 and 3 module errors with 75 of the mismatching parses having only 1 module error. The output from the parser for these sentence structures was thus typically 'close' to the LPC parse. Clearly, these mismatches demonstrated that the other connectionist modules generalise, thus compensating for the failings made by other connectionist modules. The successful linguistic generalisations made by the parser are also apparent when considering the particularly low failure rate i.e., the parser only failed to parse 45 (6.6%) of the test sentences. Generally, it is clear that the hybrid parser has successfully learnt a large number of general linguistic constraints embedded in the LPC.

During the assessment, the parser demonstrated its ability to parse simple sentence structures that consist of : noun phrases, finite and non-finite verb phrases, preposition phrases, adverbial phrases and adjective phrases. It adequately parses compound and complex sentence structures by performing constituent coordination using either coordinating conjunctions and subordinating conjunctions. The parser is also capable of performing word-level coordination thus showing that it had successfully learnt many of the linguistic constraints embedded in the LPC.

Although some incorrect preposition attachments were made, the parser is able to provide semantically plausible alternative structural attachments and exhibited behaviour that conformed to Minimal Attachment and Late Closure preferences, without those preferences being explicitly built in.

The parser also proved to be robust when presented with unfamiliar sentence structures. It was evident that the parser generalises in order to produce a 'coarser' syntactic grouping when processing unfamiliar noun phrase structures or known phrases embedded within an unfamiliar context. This coarser syntactic grouping commonly results in the preservation of the overall semantic interpretation of the sentence, as the parser is able to recognise the major phrase group even when unable to recognise the exact phrase variant. This is a distinct performance advantage over other LOB parsers and systems that use hand-crafted grammars which tend to fail when processing sentence structures that do not adhere strictly to the grammar rules.

8.3. Future Work

A number of limitations of the hybrid corpus-based parsing model were found and these need to be addressed in future work. For example, the RLD and LRD networks were unable to fully learn their training data within the processing constraints imposed. Also, in order to reduce the processing requirement, a complexity restriction was placed upon the training and test sentences thus reducing the coverage of natural language structures. The performance of the Recognition module exhibited limited generalisation which restricted the ability of the module (largely due to limited training coverage) to process unconstrained natural language texts. Integrating semantic information into the model needs to be explored in future research in order to extend the hybrid corpus-based parsing model to a more complete approach to natural language understanding. This section will discuss a number of directions for further work.

8.3.1. Improving The Segmentation Module

The segmentation module consists of the two TASRN networks for the task of sequentially segmenting phrases from the input sentence. The RLD module proved the most problematic and two fundamental improvements have been identified. The first is to use the advantages of emerging fast hardware technologies to improve the training times and flexibility of architectural configurations used. Secondly, the complexity constraint on the RLD network sequences can be adapted so that unconstrained texts can be processed. These improvements will be discussed in detail.

Improving Training Times and Flexibility of Experiments

The RLD module had the largest training set and proved the most difficult to train. A complexity constraint was applied to the maximum RLD sequence length in order to reduce training set size and thus the overall training times. However, training experiments were performed on SunSPARC 10/20 Servers and a P100MHZ Pentium PC and training times averaged 2 hours per epoch for the various architectural configurations used. This reflected the limitations of the computational resources available. The objective of the training experiments was therefore to learn as much of the training data as possible rather than to find the optimum number of hidden units. The TASRN hidden unit configuration that provided optimal performance for the RLD task, with respect to the time constraints imposed, was 165 hidden units. This configuration managed to learn 92.2% of the 4,060 training sequences and produce pure test sequence generalisations of 85% for the constrained test sample. However, the recent wide-scale availability and accessibility of fast single processor machines (Intel Pentium 450MHZ) will enable faster simulations to be performed and a more thorough assessment of the RLD module to be undertaken. Also, adapting the parallel CNAPS machine, used in a number of experiments for the Recogniser network, for TASRN and SRN networks would also allow further training experiments to be performed within reasonable time limits.

Processing Unconstrained Language Using A Sequential Temporal Input Window

The complexity constraint applied to the training and test sentences is based upon the number of symbols the RLD network can process before it recognises the beginning of a valid syntactic phrase. This limits the level of embedding allowed within the sentence. The limit imposed was 9 input symbols including look-back symbols i.e., if the RLD network's look-back limit is four symbols then the RLD network must encounter the actual beginning of a phrase by the *fifth* input symbol, as the remaining four symbols will be look-back symbols. This largely reduces the size of the training sets at the expense of limiting the parsers capability to process arbitrarily complex sentences.

The limitations imposed by the complexity constraint can be overcome by introducing an additional reset operation so that the RLD network could theoretically process arbitrarily complex sentence structures whilst using the complexity constraint. This could be achieved by resetting the RLD network's context units if it cannot signal the beginning of a syntactic phrase after processing 9 input symbols. However, rather than starting at the end (right-most symbol) of the sentence again, the RLD network starts at the input symbol to the left of the right-most symbol thus if there are *n* input symbols then process symbol *n-1* rather than symbol *n*. If the next 9 input symbols have been processed and the RLD network still cannot signal the beginning of a phrase, then reset the context units and start at input symbol *n-2* and so on. This is analogous to the 'shifting' of a temporal input window across the input sentence, however, the input symbols are being processed sequentially thus forming a sequential temporal input window.

8.3.2. Improving The Recognition Module

Even though the Recogniser network had learnt all of its training data, it was responsible for a large number of errors during testing. For example, the Recogniser network contributes to 13 of the 20 parse failures shown in Appendix J. The Recogniser network proved sensitive to position and context, therefore these generalisation failures represent a limitation of the phrase invariant recognition technique.

Two methods of overcoming these limitations have therefore been identified. The first is to adjust the position of the null symbols used to pad out the look-back and phrase fields in order to further reduce the possibility of position variance and the second is to increase the coverage of phrases in the training data.

Position of Null Padding in Recogniser Network's Input Layer

The input layer is fixed at 15 input symbols consisting of 4 look-back symbols, 10 phrasal symbols and 1 look-ahead symbol. Although the position of the look-back, phrase and look-ahead is fixed within the Recogniser network's input layer, position variance is introduced when null padding is added to the look-back and phrase fields. At present, null symbols are added to the right of the look-back symbols and phrasal symbols when the appropriate number of input symbols are unavailable. This may cause unnecessary variance of a symbols position within the input layer.

Null Padding In The Look-Back Field

When null padding is required for the look-back field, the beginning of sentence (or BOS) marker is the last available input symbol and will always be present in the look-back field. The BOS marker occurs in the look-back context for 56.2% of the phrases in the training data, and 55.4% of the constrained test data. Using the current padding scheme, variance is avoided in many instances. For example, consider the typical input phrases from the Recogniser network's training data (actual phrase is in bold type) :

(1) . *RB PP3A* ^ ***VBD*** ^^^^^^^^^ *Ti*

(2) . *RB* ^^ ***PP3A*** ^^^^^^^^^ *V*

(3) . *PP1A BEDZ* ^ ***IN N*** ^^^^^^^^.

(4) . *PP1A* ^^ ***BEDZ*** ^^^^^^^^^ *P*

where the first four symbols constitute the look-back symbols. It can be seen from (1) and (2) that the position of RB has been preserved whilst PP3A has functioned as a look-back symbol and then within a phrase. Similarities can be seen in (3) and (4) for PP1A. In these instances, padding to the left of the look-back symbols would cause unnecessary variance.

However, theoretically variance could occur within the look-back when padding to the right of the look-back symbols. For example, variance would occur if the following look-back configuration were present within the Recogniser network's training data (obviously across different input sentences) :

(5) . *Y Z* ^ *VBG RB* ^^^^^^^^^.

(6) . *X Y Z VBG RB* ^^^^^^^^^.

Here, *X*, *Y* and *Z* would actually be word tags that can function in the respective positions within the look-back field. Padding to the left rather than to the right of the look-back symbols would fix tags adjacent to a phrase that only requires two look-back symbols. Although this type of look-back configuration has not been encountered during the reported tests, a more thorough linguistic and statistical analysis of the training and test data is required to assess whether such configurations do occur and whether it would be beneficial to pad to the left of the look-back symbols rather than to the right.

Null Padding In The Phrase Field

Although null symbols are also added to the right of the phrasal symbols when required, the phrases within the LPC appear 'right-heavy', especially structures that contain coordinated sentence structures. For example, consider the typical input phrases for compound sentence structures (*S&*) from the Recogniser network's training data :

(7) . ^^^*N V N S+* ^^^^^^^.

(8) . ^^^*Na V N P S- S+* ^^^^.

Clearly, the presence of *S+* determines whether the desired output is *S* or *S&* and the position of *S+* is always to the right-hand side of the phrase. However, phrase lengths vary and thus when padding to the right of the phrase, position variance of *S+* occurs unnecessarily. To overcome this problem, padding to the left of the phrase rather than the right needs to be investigated :

(9) . ^^^^^^^^^^^*N V N S+* .

(10) . ^^^^^^^ *Na V N P S- S+*.

In (9) and (10) the position of S+ remains static and thus removes the variance problem in phrases (7) to (8). It is envisaged that this could improve the rate of phrase recognition at least for complex structures containing coordinated clauses and requires further investigation.

Increase Language Coverage

The position invariant recognition technique is sensitive to phrase position and context. A wide coverage of phrases represented in a variety of contexts within the training data is therefore essential. The current training sample for the Recogniser network consists of only 2,588 input phrases and contains 72 unique phrase groups after all natural replication has been removed. There is a considerable natural bias in the training sample towards the noun phrase group. There are 825 unique noun phrases in the training data thus representing nearly 33% of the entire training set. The verb phrase group represents only 17% of the training data with 444 unique types of verb phrases and the preposition phrase group represents nearly half that at 8% with 216 unique preposition phrases. Adverbial phrases and adjective phrases represent approximately 4% of the training sample. As there is one per sentence, unique higher level sentence structures, represented by the *S* phrase group, naturally occur considerably less than the noun, verb and preposition phrase groups with 174 unique types of sentence. The remaining phrases that occur in the training sample are derivatives of these phrase groups and the complete composition of the Recogniser network's training sample can be seen in Appendix F.

The low coverage of phrases in the Recogniser network's training data is made apparent by the recognition errors made during testing. For example, the network had difficulty recognising simple verb phrase structures (see figure 7.21) due to poor coverage in the training data. It commonly confused *V* with *Vr* phrases, and even basic preposition phrases, *P*, with preposition phrases beginning with *of*, i.e., *Po* phrases.

The coverage of *Wh*-phrases in the training data was also poor and this clearly affected its performance on the test data. There were a total of 32 sentences in the training data that contained *Wh*-phrases and the parser was able to match only 6 of these sentences with its corresponding LPC parse. The constrained test sample contained 35 sentences with *Wh*-phrases and the parser was only able to match 3 of these with the target LPC parse. The Recogniser network also confused *Wh-phrases* with subject noun phrases when the subject was embedded within a subordinate clause. However, this could be considered a sensible mistake as a *Wh*-words commonly function in the position of the subject noun.

It is envisaged that by increasing the training sample for the Recogniser network, significant improvements in generalisation will be achieved. For example, a larger training sample consisting of an even sample of 13,524 phrases was extracted from the LPC to train the Recogniser network. The network was able to learn all phrases after 500 epochs (RMS-error of 0.003) using 60 hidden units. It was consequently able to produce a pure generalisation rate of 93.1% of 3,966 test patterns thus demonstrating the potential improvements in phrase recognition that can be achieved.

8.3.3. Semantic Interpretation Using Incremental Case-Role Assignments

A major limitation of the hybrid parsing model is that it makes firm commitments to the structures it forms and is currently unable to recover from incorrect decisions made or from complete parse failures. Although people also make firm commitments to particular syntactic structures (i.e., when processing garden-path sentences[51]), they are able to correct decisions once the appropriate inputs have been received to disambiguate the structure. However, this correction may require semantic information. Currently, the parser relies upon processing sentences from Right-to-Left and using look-back symbols to reduce the probability of making incorrect decisions that require only syntactic information to resolve. There is no semantic processing performed by the parser and this must be addressed to extend the hybrid corpus-based parsing model to a more complete approach to natural language understanding.

Case-role assignments [92,140] in sentences provide a general approach to how syntactic and semantic interpretation can be combined. Traditionally, grammar rules are written to describe syntactic rather than semantic regularities. However, the structures that the rules produce correspond to semantic relations rather than to strictly syntactic ones. The case used by a Case grammar describe relationships between verbs and their arguments. This contrasts with the grammatical notion of surface case rather than surface structure. A given surface case can describe a variety of semantic cases, the common ones being : *agent* - instigator of the action; *instrument* - cause of the event or object used in causing the event; and *dative* - entity affected by the action.

M^{c}Clelland and Kawamoto [5] described an interesting connectionist system that took a partial surface parse as input (leaving certain attachment decisions unspecified) and generated a case-level representation from it. The model learnt through the presentation of correct surface-structure/case-structure pairs. However, the system was limited to fixed width input sentences, a small vocabulary, simple sentence structures and a limited set of roles. Although the system had numerous limitations it demonstrated how case-role assignments could be performed using MLP-based network architectures and how semantic micro-features could be assigned to input words. More recently, Miikkulainen [127] successfully developed a modular connectionist parser (SPEC – see Chapter 2) that was able to produce case-role assignments for each clause within a given input sentence. The collection of these case-role vectors determined the parse for the sentence Although Miikkulainen trained SPEC with a simple context-free grammar and exhibited inherent memory constraints imposed by RAAM and the number of case-roles used as his output representations was fixed at three, it demonstrated how case-role assignments can be performed using modular architectures.

To extend the current model to perform case-role assignments for each clause encountered, an additional process would first have to be defined that would be able to assign semantic micro-features as well as the LPC word tag to each word within the sentence. A TASRN network with a sequential temporal input window could be used to perform this task whereby the input to the network is the current input word and previous context, and its output is either a null symbol or a corresponding word tag and set of semantic micro-features. If the output of the network is a null symbol then additional inputs are required before the appropriate tag and semantic micro-features can be output. Once the TASRN network has output the appropriate information for a word, it's context units are reset and it begins processing the next word and so on. These tags can then be passed to the RLD network and the LRD network as before for further processing. However, it is envisaged that an existing connectionist module could be extended or an additional connectionist module incorporated to produce the case-role assignments for each clause encountered thus incrementally forming case-role assignments. The output of the parser would therefore be a syntactic structure and a set of case-role assignments. The manipulation and storage of the words, micro-features and tags would remain symbolic thus preserving the hybrid nature of the system and avoiding any limitation that would be imposed using connectionist structure encoders such as RAAM.

Appendix A

Notation

The following notation is used to describe the functionality and necessary principles of connectionist networks that are relevant to the work presented in this thesis. Note that not all symbols are meaningful for the networks described, and that in some cases subscripts and superscripts may be omitted (e.g., p is often unnecessary). The notation used is as follows :

$i(j,k)$	The unit i,j or k.
X^p	The pth input pattern vector.
x_i^p	The ith unit of the pth input pattern vector.
$YNET^p$	Net input to a set of units when the pth input pattern vector is presented.
$ynet_i^p$	Net input to unit i when the pth input pattern vector is presented. This is calculated as : $$ynet_i^p = b_j + \sum_i x_i w_{ij}$$
O^p	The output of the network when the pth input pattern vector is presented.
o_i^p	The ith output unit of the network when the pth input pattern vector is presented.
D^p	The desired output of the network when the pth input pattern vector is presented.
d_i^p	The desired output of the ith output unit when the pth input pattern vector is presented.
E^p	The output error of the network when the pth input pattern vector is presented.
A^p	The activations on the output of the network when the pth input pattern vector is presented.
a_i^p	The activation value of the ith unit when the pth input pattern vector is presented.
W	The matrix of connection weights.
w_i	The weight of the connections which feed into unit i.
w_{ij}	The weight of the connection from unit j to unit i.
f_i	The activation function associated with unit i.
η	The learning rate for all weight values.
η_{ij}	The learning rate associated with weight w_{ij}.
B	The biases to the units of the units of a network.
b_i	The bias on unit i.

θ_i	Threshold for activation on unit *i*.
Δw_{ij}	The change in w_{ij}: $\Delta w_{ij} = [w_{ij}(\text{new}) - w_{ij}(\text{old})]$
α	A momentum term which adds to the current gradient a certain fraction of the previous weight Change producing the new weight change. This has a snow ball effect on weight changes,effectively increasing the learning rate.

Appendix B

The Back-propagation Learning Algorithm And Pattern-Error Sensitive Learning Rates

The Back-propagation learning algorithm [2a] is a systematic method for training multi-layered perceptron (MLP) networks. The theory behind this solution is that the errors for the units of the hidden layers are determined by back-propagating the errors of the units of the output layer. Back-propagation can also be considered as a generalisation of the delta rule [70] for non-linear activation functions and MLP networks. The following expresses the Back-propagation algorithm in its simplest form using the notation defined in Appendix A. For a fully detailed explanation of the Back-propagation algorithm and the corresponding chain rule see [2a,73,156,157].

The Back-propagation algorithm essentially consists of two phases :

- **Forward Pass**. This is where the input vector is presented and propagated forward through the network to compute the output values a_i^p for each output unit. The activation of each unit in the hidden or output layer is a differentiable function of the total input to that unit, given by :

  (EB.1)

$$a_i^p = f(ynet_i^p)$$

  where :

  (EB.2)

$$ynet_i^p = b_j + \sum_i x_i w_{ij}$$

  The sigmoid activation function, *f*, is defined as :

  (EB.3)

$$a_i^p = f(ynet_i^p) = \frac{1}{1 + e^{-ynet_i^p}}$$

- **Reverse Pass**. This error signal is passed to each unit in the network and appropriate weight changes are calculated.

If the unit is an output unit, the error signal is given by :

(EB.4)

$$\delta_i^p = (d_i^p - a_i^p) f'(ynet_i^p)$$

In this case, the derivative of the sigmoid, *f'*, is equal to :

(EB.5)

$$f'(ynet_i^p) = a_i^p (1 - a_i^p)$$

Such that the error signal for an output unit can be written as :

(EB.6)

$$\delta_i^p = (d_i^p - a_i^p) a_i^p (1 - a_i^p)$$

The error signal for a hidden unit is determined recursively in terms of the error signals of the output units to which it directly connects and the weights of those connections. For all hidden units, *H*, the error signal is calculated as :

(EB.7)

$$\delta_j^p = f'(ynet_j^p) \sum_{i=1}^{O} \delta_i^p w_{ji} = a_j^p (1 - a_j^p) \sum_{i=1}^{O} \delta_i^p w_{ji}$$

where i = output unit i, j = hidden unit j and O = total number of output units.

The change in weight (in each weight layer) is dependent upon the past weight change and is calculated as :

(EB.8)

$$\Delta w_{ij}(t+1) = \eta \delta_i^p a_j^p + \alpha \Delta w_{ij}(t)$$

Where t indexes the presentation number and α is a constant that determines the effect of the previous weight change and η is the constant of proportionality (the learning rate). However, when using the pattern-error sensitive learning rate, η is adapted according to the output error for the current pattern and is thus calculated as :

(EB.9)

$$\eta = \frac{\sum_{i=0}^{q-1} |e_i|}{q} * c1 + c2$$

where e$_i$ is the error for output unit *i*, *q* is the total number of output units and *c1* and *c2* are constants which allow adjustment of the range of η. *c1* = 0.6 and *c2* = 0.2 were chosen to give 0.2 $\leq$ 0 $\geq$ 0.8. Therefore between 0.2 and 0.8, η is linearly dependent upon the average absolute output error. Although the current emphasis here is on the average absolute output error, formula EB.9 could be adapted to refer to the error on an individual output unit or units and thus narrow the focus to specific output unit(s) with high errors.

Appendix C

A Context-free Grammar for The English Language

Terminal Symbols

adjective
adverb
am
are
been
being
can
demphasizer
det
did
do
does
emphasizer
future_modal
had
has
have
having
in
intransitive_verb
is
it
mass_noun
mass_quantifier
mental_state_noun
moved
moving
much
not
noun
number
ordinal
pl_noun
prep
pres_2ndpers
present_infinitive_verb
pronoun
proper_noun
quantifier
ref_prep
relative_quantifier
relative_time_prep
sg_noun

stative_verb
that
to
too
v_ed
v_ing
was
were
wh_pronoun
with

Non-Terminal Symbols

<ADJG>
<ADJL>
<ADV_1>
<ADV_2>
<ADV_3>
<ADV_4>
<ADV_PH>
<ADVG>
<ADVINF>
<ADVINF2>
<ADVL>
<AEMPL>
<AEMPL2>
<AUXDO>
<AUXHAVE>
<AUXVG_1>
<AVP_1>
<DECL_SENT>
<IMPR_SENT>
<IMPR_SENT2>
<INFVPG>
<INTRVG>
<MAINVG_1>
<MAINVG_2>
<MASQL>
<NEGATEP>
<NOUNL>
<NP>
<NPG_1>
<NPG>
<ORDG>
<PP>
<PPL>
<PRESECPERSVG>
<QUEST_VG>
<QUEST>
<S>
<STATVG>
<V_INGP>
<VP>
<VPG>

Production Rules

<S>	→	<DECL_SENT>
<S>	→	<IMPR_SENT>
<S>	→	<QUEST>
<DECL_SENT>	→	<NP> <VP>
<DECL_SENT>	→	v_ing <DECL_SENT>
<DECL_SENT>	→	<ADV_PH> <DECL_SENT>
<NP>	→	<NPG>
<NP>	→	<NPG> <PPL>
<PPL>	→	<PP>
<PPL>	→	<PP> <PPL>
<PP>	→	prep <NP>
<PP>	→	ref_prep <VP>
<NPG>	→	det pl_noun
<NPG>	→	pl_noun
<NPG>	→	<ADJL> pl_noun
<NPG>	→	det sg_noun
<NPG>	→	det <ADJL> sg_noun
<NPG>	→	pronoun
<NPG>	→	proper_noun
<NPG>	→	<ADJL> proper_noun
<NPG>	→	det <NPG_1>
<NPG>	→	<NPG_1>
<NPG>	→	<ADVL> <NPG_1>
<NPG>	→	det <ADJL> <NPG_1>
<NPG_1>	→	mental_state_noun
<NPG_1>	→	v_ing
<NPG_1>	→	mass_noun
<ADJL>	→	<ADJG>
<ADJL>	→	<ADJG> <ADJL>
<ADJG>	→	adjective
<ADJG>	→	<AEMPL> adjective
<ADJG>	→	adverb <AVP_1>
<ADJG>	→	<AEMPL> adverb <AVP_1>
<ADJG>	→	noun v_ed
<ADJG>	→	<AEMPL> noun v_ed
<ADJG>	→	v_ed
<ADJG>	→	v_ing
<ADJG>	→	<AEMPL> v_ed
<ADJG>	→	quantifier
<ADJG>	→	<AEMPL> quantifier
<ADJG>	→	ordinal
<ADJG>	→	<ORDG> number
<ADJG>	→	mass_noun
<ADJG>	→	<MASQL> mass_noun
<ADJG>	→	<NOUNL> v_ed
<ADJG>	→	<NOUNL> v_ing
<AEMPL>	→	emphasizer
<AEMPL>	→	emphasizer <AEMPL>
<AEMPL2>	→	demphasizer
<AEMPL2>	→	demphasizer <AEMPL2>
<AVP_1>	→	v_ed
<AVP_1>	→	v_ing
<MASQL>	→	mass_quantifier
<MASQL>	→	mass_quantifier <MASQL>

<ORDG>	→	ordinal
<ORDG>	→	ordinal <ORDG>
<NOUNL>	→	noun
<NOUNL>	→	noun <NOUNL>
<AUXDO>	→	do
<AUXDO>	→	does
<AUXDO>	→	did
<AUXHAVE>	→	have
<AUXHAVE>	→	has
<AUXHAVE>	→	had
<AUXVG_1>	→	can
<AUXVG_1>	→	am
<AUXVG_1>	→	are
<AUXVG_1>	→	is
<AUXVG_1>	→	was
<AUXVG_1>	→	were
<MAINVG_1>	→	being
<MAINVG_1>	→	having
<MAINVG_2>	→	moved
<MAINVG_2>	→	moving
<VPG>	→	<AUXDO> present_infinitive_verb
<VPG>	→	<AUXDO> not present_infinitive_verb
<VPG>	→	present_infinitive_verb
<VPG>	→	v_ed
<VPG>	→	<AUXHAVE> been <AVP_1>
<VPG>	→	<AUXHAVE> been not <AVP_1>
<VPG>	→	<AUXHAVE> not been not <AVP_1>
<VPG>	→	<AUXHAVE> not been <AVP_1>
<VPG>	→	<AUXHAVE> v_ed
<VPG>	→	<AUXHAVE> not v_ed
<VPG>	→	<AUXVG_1> <AVP_1>
<VPG>	→	<AUXVG_1> not <AVP_1>
<VPG>	→	<AUXVG_1> being v_ed
<VPG>	→	<AUXVG_1> not being v_ed
<VPG>	→	<AUXVG_1> being not v_ed
<VPG>	→	<AUXVG_1> not being not v_ed
<VPG>	→	future_modal present_infinitive_verb
<VPG>	→	future_modal not present_infinitive_verb
<VPG>	→	future_modal have v_ed
<VPG>	→	future_modal not have v_ed
<VPG>	→	future_modal have been <AVP_1>
<VPG>	→	future_modal not have been <AVP_1>
<VPG>	→	future_modal have been not <AVP_1>
<VPG>	→	future_modal not have been not <AVP_1>
<VPG>	→	future_modal be <AVP_1>
<VPG>	→	future_modal not be <AVP_1>
<VPG>	→	future_modal not be not <AVP_1>
<VPG>	→	future_modal be being v_ed
<VPG>	→	future_modal not be being v_ed
<VPG>	→	future_modal be being not v_ed
<VPG>	→	future_modal not be being not v_ed
<ADVL>	→	<ADVG>
<ADVL>	→	<ADVG> <ADVL>
<ADVG>	→	relative_time_prep <ADV_1>
<ADVG>	→	prep <NP>

<ADVG>	→	relative_direction_prep
<ADVG>	→	<ADV_2> mental_state_noun
<ADVG>	→	<V_INGP>
<ADVG>	→	<ADV_3> <V_INGP>
<ADVG>	→	adverb
<ADVG>	→	<AEMPL> adverb
<ADVG>	→	<AEMPL2> adverb
<ADVG>	→	relative_quantifier adverb
<ADVG>	→	emphasizer much relative_quantifier adverb
<ADVG>	→	adverb <ADV_4>
<ADVG>	→	<ADVINF> <INFVPG>
<ADVG>	→	<ADV_5> adverb <ADV_4>
<ADVINF>	→	to
<ADVINF2>	→	too
<ADV_1>	→	v_ing
<ADV_1>	→	<DECL_SENT>
<ADV_2>	→	in
<ADV_2>	→	with
<V_INGP>	→	<MAINVG_1>
<V_INGP>	→	having moved
<V_INGP>	→	having <NEGATEP> moved
<V_INGP>	→	having been <MAINVG_2>
<V_INGP>	→	having <NEGATEP> been <MAINVG_2>
<V_INGP>	→	having <NEGATEP> been <NEGATEP> <MAINVG_2>
<V_INGP>	→	having been <NEGATEP> <MAINVG_2>
<NEGATEP>	→	not
<NEGATEP>	→	not <NEGATEP>
<VP>	→	<AUXVG_1> <ADJG>
<VP>	→	<ADVL> <AUXVG_1> <ADJG>
<VP>	→	<AUXVG_1> <ADJG> <ADVL>
<VP>	→	<ADVL> <AUXVG_1> <ADJG> <ADVL>
<VP>	→	<INTRVG>
<VP>	→	<ADVL> <INTRVG>
<VP>	→	<INTRVG> <ADVL>
<VP>	→	<ADVL> <INTRVG> <ADVL>
<VP>	→	<STATVG>
<VP>	→	<ADVL> <STATVG>
<VP>	→	<STATVG> <ADVL>
<VP>	→	<ADVL> <STATVG> <ADVL>
<VP>	→	<STATVG> <DECL_SENT>
<VP>	→	<ADVL> <STATVG> <DECL_SENT>
<VP>	→	<STATVG> <ADVL> <DECL_SENT>
<VP>	→	<ADVL> <STATVG> <ADVL> <DECL_SENT>
<VP>	→	<STATVG> <NP>
<VP>	→	<STATVG> <NP> <NP>
<VP>	→	<ADVL> <STATVG> <NP>
<VP>	→	<ADVL> <STATVG> <NP> <NP>
<VP>	→	<STATVG> <ADVL> <NP>
<VP>	→	<STATVG> <ADVL> <NP> <NP>
<VP>	→	<ADVL> <STATVG> <ADVL> <NP>
<VP>	→	<ADVL> <STATVG> <ADVL> <NP> <NP>
<INTRVG>	→	intransitive_verb
<STATVG>	→	stative_verb
<IMPR_SENT>	→	<PRESECPERSVG>
<IMPR_SENT>	→	do <PRESECPERSVG>

<IMPR_SENT>	→	<PRESECPERSVG> <ADVG>
<IMPR_SENT>	→	do <PRESECPERSVG> <ADVG>
<IMPR_SENT>	→	<PRESECPERSVG> <IMPR_SENT2>
<IMPR_SENT>	→	do <PRESECPERSVG> <IMPR_SENT2>
<IMPR_SENT>	→	<PRESECPERSVG> <ADVG> <IMPR_SENT2>
<IMPR_SENT>	→	do <PRESECPERSVG> <ADVG> <IMPR_SENT2>
<IMPR_SENT>	→	<PRESECPERSVG> <PP>
<IMPR_SENT>	→	do <PRESECPERSVG> <PP>
<IMPR_SENT>	→	do <PRESECPERSVG> <ADVG> <PP>
<IMPR_SENT>	→	<PRESECPERSVG> <ADVG> <PP>
<IMPR_SENT>	→	<PRESECPERSVG> <IMPR_SENT2> <PP>
<IMPR_SENT>	→	do <PRESECPERSVG> <IMPR_SENT2> <PP>
<IMPR_SENT>	→	do <PRESECPERSVG> <ADVG> <IMPR_SENT2> <PP>
<IMPR_SENT>	→	<PRESECPERSVG> <ADVG> <IMPR_SENT2> <PP>
<IMPR_SENT>	→	do <PRESECPERSVG> <PP> <ADVG>
<IMPR_SENT>	→	<PRESECPERSVG> <PP> <ADVG>
<IMPR_SENT>	→	do <PRESECPERSVG> <ADVG> <PP> <ADVG>
<IMPR_SENT>	→	<PRESECPERSVG> <ADVG> <PP> <ADVG>
<IMPR_SENT>	→	<PRESECPERSVG> <IMPR_SENT2> <PP> <ADVG>
<IMPR_SENT>	→	do <PRESECPERSVG> <IMPR_SENT2> <PP> <ADVG>
<IMPR_SENT>	→	do <PRESECPERSVG> <ADVG> <IMPR_SENT2> <PP> <ADVG>
<IMPR_SENT>	→	<PRESECPERSVG> <ADVG> <IMPR_SENT2> <PP> <ADVG>
<IMPR_SENT2>	→	<PP>
<IMPR_SENT2>	→	<NP>
<PRESECPERSVG>	→	pres_2ndpers
<QUEST>	→	wh_pronoun <QUEST_VG>
<QUEST>	→	wh_pronoun <QUEST_VG> <IMPR_SENT2>
<QUEST>	→	wh_pronoun <QUEST_VG> <ADVG>
<QUEST>	→	wh_pronoun <QUEST_VG> <IMPR_SENT2> <ADVG>
<QUEST>	→	wh_pronoun is it that <DECL_SENT>
<QUEST_VG>	→	<AUXVG_1> <NP>
<QUEST_VG>	→	v_ed <NP>
<QUEST_VG>	→	<AUXHAVE> <NP> been <AVP_1>
<QUEST_VG>	→	<AUXHAVE> <NP> not been <AVP_1>
<QUEST_VG>	→	<AUXHAVE> <NP> been not <AVP_1>
<QUEST_VG>	→	<AUXHAVE> <NP> not been not <AVP_1>
<QUEST_VG>	→	<AUXHAVE> <NP> v-ed
<QUEST_VG>	→	<AUXHAVE> <NP> not v-ed
<QUEST_VG>	→	<AUXVG_1> <NP> <AVP_1>
<QUEST_VG>	→	<AUXVG_1> <NP> not <AVP_1>
<QUEST_VG>	→	<AUXVG_1> <NP> being v-ed
<QUEST_VG>	→	<AUXVG_1> <NP> not being v-ed
<QUEST_VG>	→	<AUXVG_1> <NP> being not v-ed
<QUEST_VG>	→	<AUXVG_1> <NP> not being not v-ed
<QUEST_VG>	→	future_modal <NP> present_infinitive_verb
<QUEST_VG>	→	future_modal <NP> not present_infinitive_verb
<QUEST_VG>	→	future_modal <NP> have v-ed
<QUEST_VG>	→	future_modal <NP> not have v-ed
<QUEST_VG>	→	future_modal <NP> have not v-ed
<QUEST_VG>	→	future_modal <NP> not have not v-ed
<QUEST_VG>	→	future_modal <NP> be <AVP_1>
<QUEST_VG>	→	future_modal <NP> not be <AVP_1>
<QUEST_VG>	→	future_modal <NP> be not <AVP_1>
<QUEST_VG>	→	future_modal <NP> not be not <AVP_1>
<QUEST_VG>	→	future_modal <NP> be being v-ed

<QUEST_VG>	→	future_modal <NP> not be being v-ed
<QUEST_VG>	→	future_modal <NP> be being not v-ed
<QUEST_VG>	→	future_modal <NP> not be being not v-ed
<QUEST_VG>	→	future_modal <NP> be not being v-ed
<QUEST_VG>	→	future_modal <NP> not be not being v-ed
<QUEST_VG>	→	future_modal <NP> be not being not v-ed
<QUEST_VG>	→	future_modal <NP> not be not being not v-ed

Appendix D

The Word Tags Used In The LPC

The word tags can be sub-divided into five seperate groups : nouns, verbs, prepositions, conjunctions and punctuation. Although punctuation has been omitted, a bit representation has been defined for future use. The following tables lists the tag, description and bit representation within the encoding scheme defined in Chapter 4 for each tag in each group.

83 Word Tags for Nouns

Tag	Description	Bit Representation
ABL	pre-qualifier in a noun phrase (QUITE, RATHER, SUCH)	1 0 0 0 0 0 0 1
ABN	pre-quantifier in a noun phrase (ALL, HALF)	1 0 0 0 0 0 1 0
AP	post-determiner (FEW, FEWER, FORMER)	1 0 0 0 0 0 1 1
AP$	OTHER'S	1 0 0 0 0 1 0 0
APS	OTHERS	1 0 0 0 0 1 0 1
APS$	OTHERS'	1 0 0 0 0 1 1 0
AT	singular article (A, AN, EVERY)	1 0 0 0 0 1 1 1
ATI	singular or plural article (THE, NO)	1 0 0 0 1 0 0 0
CD	cardinal number (2, 3, etc; TWO, THREE, THOUSAND)	1 0 0 0 1 0 0 1
CS$	cardinal number + genitive	1 0 0 0 1 0 1 0
CD-CD	hyphenated pair of cardinal numbers (e.g. 1988-90)	1 0 0 0 1 0 1 1
CD1	ONE	1 0 0 0 1 1 0 0
CD1$	ONE'S	1 0 0 0 1 1 0 1
CD1S	ONES	1 0 0 0 1 1 1 0
CDS	cardinal number + plural (TENS, MILLIONS, DOZENS, etc)	1 0 0 0 1 1 1 1
DT	singular determiner (ANOTHER, EACH, THAT, THIS)	1 0 0 1 0 0 0 0
DT$	singular determiner + genitive (ANOTHER'S)	1 0 0 1 0 0 0 1
DTI	determiner neutral for number (ANY, ENOUGH, SOME)	1 0 0 1 0 0 1 0
DTS	plural determiner (THESE, THOSE)	1 0 0 1 0 0 1 1
DTX	determiner / double conjunction (EITHER, NEITHER)	1 0 0 1 0 1 0 0
EX	existential THERE	1 0 0 1 0 1 0 1
JJ	adjective (general)	1 0 0 1 0 1 1 0
JJB	attributive adjective	1 0 0 1 0 1 1 1
JNP	adjective with word-initial cap; e.g. WELSH, KEYNESIAN	1 0 0 1 1 0 0 0
JJR	comparative adjective	1 0 0 1 1 0 0 1
JJT	superlative adjective	1 0 0 1 1 0 1 0
NC	cited word as singular noun (e.g. "LED is a verb")	1 0 0 1 1 0 1 1
NN	singular common noun	1 0 0 1 1 1 0 0
NNP	singular common noun; word-initial cap; e.g. LONDONER	1 0 0 1 1 1 0 1
NNPS	plural common noun; word-initial cap; e.g. LONDONERS	1 0 0 1 1 1 1 0
NNPS$	plural common noun; word-init. cap; genitive LONDONERS'	1 0 0 1 1 1 1 1
NNP$	singular common noun; word-init.cap; gen; e.g. LONDONER'S	1 0 1 0 0 0 0 0
NNS	plural common noun	1 0 1 0 0 0 0 1
NNS$	plural common noun + genitive	1 0 1 0 0 0 1 0
NNU	singular unit of measurement (e.g. IN. KG.)	1 0 1 0 0 0 1 1
NNUS	plural unit of measurement (e.g. INS. KGS.)	1 0 1 0 0 1 0 0
NNUS$	plural unit of measurement + genitive	1 0 1 0 0 1 0 1
NP	singular proper noun	1 0 1 0 0 1 1 0
NPS	plural proper noun	1 0 1 0 0 1 1 1
NPS$	plural proper noun + genitive	1 0 1 0 1 0 0 0
NP$	singular proper noun + genitive	1 0 1 0 1 0 0 1
NPL	singular locative noun; word-initial cap.; e.g. ISLAND	1 0 1 0 1 0 1 0

Tag	Description	Bit Representation
NPLS	plural locative noun; word-initial cap.; e.g. ISLANDS	1 0 1 0 1 0 1 1
NPLS$	plural locative noun; word-init. cap; + gen.; e.g. ISLANDS'	1 0 1 0 1 1 0 0
NPL$	singular locative noun; word-init. cap; + gen.; e.g. ISLAND'S	1 0 1 0 1 1 0 1
NPT	singular titular noun; word-initial cap.; e.g. DR.	1 0 1 0 1 1 1 0
NPTS	plural titular noun; word-initial cap.; e.g. MESSRS.	1 0 1 0 1 1 1 1
NPTS$	plural titular noun; word-init. cap.; + gen.; e.g. QUEENS'	1 0 1 1 0 0 0 0
NR	singular adverbial noun (JANUARY, MONDAY, EAST)	1 0 1 1 0 0 0 1
NR$	singular adverbial noun + genitive	1 0 1 1 0 0 1 0
NRS	plural adverbial noun	1 0 1 1 0 0 1 1
OD	ordinal number (1ST, 2ND, etc; FIRST, SECOND, etc)	1 0 1 1 0 1 0 0
PN	nominal pronoun (ANYBODY, ANYONE, EVERYONE etc)	1 0 1 1 0 1 0 1
PN$	nominal pronoun + genitive	1 0 1 1 0 1 1 0
PP$	possessive determiner (MY, YOUR, etc)	1 0 1 1 0 1 1 1
PPS$	possessive pronoun (MINE, YOURS, etc)	1 0 1 1 1 0 0 0
PP1A	personal pronoun, 1st pers sing nom (I)	1 0 1 1 1 0 0 1
PP1AS	personal pronoun, 1st pers plur nom (WE)	1 0 1 1 1 0 1 0
PP1O	personal pronoun, 1st pers sing acc (ME)	1 0 1 1 1 0 1 1
PP1OS	personal pronoun, 1st pers plur acc (US, 'S)	1 0 1 1 1 1 0 0
PP2	personal pronoun, 2nd pers (YOU, THOU, THEE, YE)	1 0 1 1 1 1 0 1
PP3	personal pronoun, 3rd pers sing nom+acc (IT)	1 0 1 1 1 1 1 0
PP3A	personal pronoun, 3rd pers sing nom (HE, SHE)	1 0 1 1 1 1 1 1
PP3AS	personal pronoun, 3rd pers plur nom (THEY)	1 1 0 0 0 0 0 0
PP3O	personal pronoun, 3rd pers plur acc (HIM, HER)	1 1 0 0 0 0 0 1
PP3OS	personal pronoun, 3rd pers plur acc (THEM, 'EM)	1 1 0 0 0 0 1 0
PPL	singular reflexive pronoun	1 1 0 0 0 0 1 1
PPLS	plural reflexive pronoun	1 1 0 0 0 1 0 0
QL	qualifier (AS, AWFULLY, LESS, MORE, SO, TOO, VERY, etc)	1 1 0 0 0 1 0 1
QLP	post-qualifier (ENOUGH, INDEED)	1 1 0 0 0 1 1 0
WDT	WH-determiner (WHAT, WHATEVER,WHICH)	1 1 0 0 0 1 1 1
WP	WH-pronoun, nom+acc (WHO, WHOEVER, THAT)	1 1 0 0 1 0 0 0
WP$	WH-pronoun, genitive (WHOSE)	1 1 0 0 1 0 0 1
WPA	WH-pronoun, nom (WHOSOEVER)	1 1 0 0 1 0 1 0
WPO	WH-pronoun, acc (WHOM, WHOMSOEVER)	1 1 0 0 1 0 1 1
PP$$	possessive pronoun (MINE,YOURS etc)	1 1 0 0 1 1 0 0
NN$	singular common noun + genitive	1 1 0 0 1 1 0 1
NPT$	titular noun with w.i.c + genitive	1 1 0 0 1 1 1 0
WDTR	WH-determiner - relative e.g. WHICH	1 1 0 0 1 1 1 1
WP$R	WH-pronoun - relative - gen e.g. WHOSE	1 1 0 1 0 0 0 0
WPOR	WH-pronoun - relative - acc e.g. WHOM	1 1 0 1 0 0 0 1
WPR	WH-pronoun - relative - nom+acc e.g. THAT,relative WHO	1 1 0 1 0 0 1 0
WRB	WH-verb (HOW,WHEN)	1 1 0 1 0 0 1 1

4 Word Tags for Prepositions

Tag	Description	Bit Representation
IN	preposition (general)	1 0 0 1
INF	FOR as a preposition	1 0 1 0
INO	OF as a preposition	1 0 1 1
INW	WITH as a preposition	1 1 0 0

33 Word Tags for Verbs

Tag	Description	Bit Representation
BE	BE	1 0 0 0 0 0 1
BED	WERE	1 0 0 0 0 1 0
BEDZ	WAS	1 0 0 0 0 1 1
BEG	BEING	1 0 0 0 1 0 0
BEM	AM	1 0 0 0 1 0 1
BEN	BEEN	1 0 0 0 1 1 0
BER	ARE, 'RE	1 0 0 0 1 1 1

Tag	Description	Bit Representation
BEZ	IS, 'S	1 0 0 1 0 0 0
DO	DO	1 0 0 1 0 0 1
DOD	DID	1 0 0 1 0 1 0
DOZ	DOES	1 0 0 1 0 1 1
HV	HAVE	1 0 0 1 1 0 0
HVD	HAD, 'D (past tense)	1 0 0 1 1 0 1
HVG	HAVING	1 0 0 1 1 1 0
HVN	HAD (past participle)	1 0 0 1 1 1 1
HVZ	HAS, 'S	1 0 1 0 0 0 0
MD	modal auxiliary	1 0 1 0 0 0 1
RB	adverb (general)	1 0 1 0 0 1 0
RB$	adverb + genitive (ELSE'S)	1 0 1 0 0 1 1
RBR	comparative adverb	1 0 1 0 1 0 0
RBT	superlative adverb	1 0 1 0 1 0 1
RI	adverb (homograph of preposition: BELOW, NEAR, etc)	1 0 1 0 1 1 0
RN	nominal adverb (HERE, NOW, THERE, THEN, etc)	1 0 1 0 1 1 1
RP	adverbial particle (BACK, DOWN, OFF, etc)	1 0 1 1 0 0 0
TO	infinitival TO	1 0 1 1 0 0 1
UH	interjection	1 0 1 1 0 1 0
VB	base form of lexical verb (uninflected present tense, infinitive)	1 0 1 1 0 1 1
VBD	past tense of lexical verb	1 0 1 1 1 0 0
VBG	present participle or gerund of lexical verb	1 0 1 1 1 0 1
VBN	past participle of lexical verb	1 0 1 1 1 1 0
VBZ	3rd person singular of verb	1 0 1 1 1 1 1
WRB	WH-adverb (HOW, WHEN, WHERE, etc)	1 1 0 0 0 0 0
XNOT	NOT, N'T	1 1 0 0 0 0 1

3 Word Tags for Conjunctions

Tag	Description	Bit Representation
ABX	pre-quantifier / double conjunction (e.g. BOTH)	1 0 1
CC	coordinating conjunction (e.g. AND, AND/OR, BUT, OR, YET)	1 1 0
CS	subordinating conjunction (e.g. AFTER, ALTHOUGH, etc)	1 1 1

20 Word Tags for Punctuation

Tag	Description	Bit Representation
^	null	0 0 0 0 0
ZZ	letter of the alphabet (E, X, etc).	1 0 0 0 1
!	exclamation mark (!)	1 0 0 1 0
&FO	formula	1 0 0 1 1
&FW	foreign word	1 0 1 0 0
(	left bracket	1 0 1 0 1
[	left bracket	1 0 1 0 1
)	right bracket	1 0 1 1 0
]	right bracket	1 0 1 1 0
*'	begin quote	1 0 1 1 1
*"	begin quote	1 0 1 1 1
**'	end quote	1 1 0 0 0
**"	end quote	1 1 0 0 0
-	dash	1 1 0 0 1
,	comma	1 1 0 1 0
?	question mark	1 1 0 1 1
...	ellipsis	1 1 1 0 0
:	colon	1 1 1 0 1
;	semicolon	1 1 1 1 0
.	full stop	1 1 1 1 1

Appendix E

The Constituent Tags Used In The LPC

The constituent tags can be sub-divided into five main groups : sentence tags, finite clause tags, non-finite and verbless clause tags, major phrase tags, and minor phrase tags. The following table lists the tag, description and bit representation within the encoding scheme defined in Chapter 4 for each tag in each group.

3 Sentence Tags

Tag	**Description**	**Bit Representation**
S	sentence.	1 0 1
Sq	piece of direct quotation.	1 1 0
Si	interpolated sentence.	1 1 1

5 Finite Clause Tags

Tag	**Description**	**Bit Representation**
F	finite subordinate clause i.e. a clause which contains a finite verb.	1 0 0 1
Fa	finite adverbial clause(e.g. finite subordinate clause of time etc)	1 0 1 0
Fc	comparative clause, normally beginning with `than' or `as'.	1 0 1 1
Fn	finite nominal clause (subord clause func in pos of Noun PH)	1 1 0 0
Fr	relative clause - whether restrictive or non-restrictive.	1 1 0 1

9 Non-finite And Verbless Clause Tags

Tag	**Description**	**Bit Representation**
T	nonfinite clause.	1 0 0 0 1
Ti	to-infinitive clause.	1 0 0 1 0
Tg	-ing clause.	1 0 0 1 1
Tn	past participle clause.	1 0 1 0 0
Tb	`bare infinitive clause'.	1 0 1 0 1
Tf	subject of the infinitive which is introduced by `for'.	1 0 1 1 0
W	nonfinite or verbless clause that is introduced by with.	1 0 1 1 1
L	verbless clause that is not introduced by subordinating conjunction.	1 1 0 0 0

17 Major Phrase Tags

Tag	**Description**	**Bit Representation**
V	A finite `verb phrase' i.e. one that excludes objects, complements	1 0 0 0 0 1
Vo	Used when a verb phrase is split into two parts by subj-aux inv. o=operator	1 0 0 0 1 0
Vr	Used when a verb phrase is split into two parts by subj-aux inversion. r=remaindr	1 0 0 0 1 1
Vi	Label for nonfinite verb phrases(VP). i.e VP's that are VP's of Ti,Tg and Tn.	1 0 0 1 0 0
Vg	Label for nonfinite verb phrases(VP). i.e VP's that are VP's of Ti,Tg and Tn.	1 0 0 1 0 1
Vn	Label for nonfinite verb phrases(VP). i.e VP's that are VP's of Ti,Tg and Tn.	1 0 0 1 1 0
N	Label for a noun phrase, whether it is a single word or a sequence of words.	1 0 0 1 1 1
Na	A noun phrase marked as subject of the verb.	1 0 1 0 0 0
Nq	A wh- noun phrase, such as `who', `which', `which car', `what time'.	1 0 1 0 0 1
J	An adjective phrase such as `happy', `very tall' etc.	1 0 1 0 1 0
Jq	A phrase beginning with a wh-word e.g. `How old'.	1 0 1 0 1 1

Tag	Description	Bit Representation
P	A prepositional phrase, e.g. `in London' or `on arriving at the station'.	1 0 1 1 0 0
Pq	A prepositional phrase with a wh-word, e.g. `on whose behalf', `in which case'.	1 0 1 1 0 1
Po	A prepositional phrase beginning with the preposition 'of'.	1 0 1 1 1 0
Poq	A prepositional phrase beginning with the preposition 'of' with wh-word?	1 1 0 0 0 1
R	An adverb phrase, e.g. `there',`quickly' or a sequence such as `quite often' etc	1 0 1 1 1 1
Rq	an adverb phrase beginning with a wh-word, e.g. `How do you feel?', or `how long'	1 1 0 0 0 0

7 Minor Phrase Tags

Tag	Description	Bit Representation
M	`numeric phrase' when such an expression is part of a noun phrase.	1 0 0 1
D	`determiner phrase'.	1 0 1 0
Dq	determiner phrase beginning with a wh-word.	1 0 1 1
G	genitive phrase - phrase with two or more words acting as the genitive in a noun.	1 1 0 0
X	negative word 'not' when acting as an independent element of a clause.	1 1 0 1
E	label used for existential `there' i.e 'There' is nothing wrong.	1 1 1 0
U	tag used for an exclamatory word such as `Oh','yes', or 'no'.	1 1 1 1

Appendix F

Composition of The Training Sample

The training sample consisted of 654 LPC sentences and contained 2,588 unique syntactic phrases. These are listed below.

Phrase	Description	Occurrences	% Training Data
N	Noun phrase.	825	31.88
V	Finite verb phrase.	444	17.16
P	Prepositional phrase.	216	8.35
S	Sentence.	174	6.72
R	Adverb phrase	111	4.29
J	Adjective phrase.	90	3.48
Po	Prepositional phrase beginning with 'of'.	81	3.13
Na	Noun phrase marked as subject of the verb.	63	2.43
Vi	Non-finite verb phrase of Ti clause.	63	2.43
Ti	to-infinitive clause.	59	2.28
Fn	Finite nominal clause	32	1.24
Nq	Wh- noun phrase.	32	1.24
Fr	Relative clause.	28	1.08
Vg	Non-finite verb phrase of Tg clause.	26	1.00
S+	Second or subsequent conjoin of a compound sentence.	25	0.97
Tg	-ing clause.	25	0.97
S&	Compound sentence.	22	0.85
N+	Second or subsequent conjoin of a compound Noun phrase.	18	0.70
Vn	Non-finite verb phrase of Tn clause	18	0.70
N&	Compound Noun phrase	17	0.66
Tn	Past participle clause.	16	0.62
Fa	Finite adverbial clause	15	0.58
G	Genitive phrase - phrase with two or more words acting as the genitive in a noun.	10	0.39
S-	A conjoin when it does not begin with a coordinating conjunction (main category being sentence).	10	0.39
U	tag used for an exclamatory word such as `Oh','yes', or 'no'.	10	0.39
NN+	Second or subsequent conjoin of a compound singular common noun	9	0.35
Sq	piece of direct quotation.	9	0.35
Tb	`bare infinitive clause'.	9	0.35
JJ&	Compound adjective (general)	8	0.31
JJ+	Second or subsequent conjoin of a compound adjective (general)	8	0.31

Phrase	Description	Occurrences	% Training Data
NN&	Compound singular common noun	8	0.31
Rq	Adverb phrase beginning with a wh-word	8	0.31
Si	interpolated sentence.	8	0.31
RB=	Idiom tags for adverb (general)	7	0.27
Vr	Used when a verb phrase is split into two parts by subj-aux inversion.r=remaindr	7	0.27
E	Existential `there'.	6	0.23
IN=	Idiom tags for preposition (general)	6	0.23
Fc	Comparative clause, normally beginning with `than' or `as'.	5	0.19
NNS&	Compound plural common noun	5	0.19
NNS+	Second or subsequent conjoin of a compound plural common noun	5	0.19
Vo	Used when a verb phrase is split into two parts by subj-aux inversion.o=operator	4	0.15
J&	Compound adjective phrase	3	0.12
J+	Second or subsequent conjoin of a compound adjective phrase	3	0.12
JJ-	A conjoin when it does not begin with a coordinating conjunction (main category being adjective).	3	0.12
P+	Second or subsequent conjoin of a compound prepositional phrase	3	0.12
Ti&	Compound to-infinitive clause.	3	0.12
Ti+	Second or subsequent conjoin of a compound to-infinitive clause.	3	0.12
NNU=	Idiom tags for singular unit of measurement.	2	0.08
P&	Compound prepositional phrase	2	0.08
W	Non-finite or verbless clause that is introduced by 'with'.	2	0.08
AP=	Idiom tags for post-determiner	1	0.04
IN&	Compound preposition (general)	1	0.04
IN+	Second or subsequent conjoin of a compound preposition (general)	1	0.04
J-	A conjoin when it does not begin with a coordinating conjunction (main category being an adjective phrase)	1	0.04
JJB/JJ&	A JJB without JJ&	1	0.04
L-	A conjoin when it does not begin with a coordinating conjunction (main category being verbless clause that is not introduced by with or by a subordinating conjunction).	1	0.04
L&	Compound verbless clause that is not introduced by with or by a subordinating conjunction.	1	0.04
Na&	Compound noun phrase marked as subject of the verb.	1	0.04
NN-	A conjoin when it does not begin with	1	0.04

Phrase	Description	Occurrences	% Training Data
	a coordinating conjunction (main category being singular common noun).		
NP&	Compound singular proper noun	1	0.04
NP+	Second or subsequent conjoin of a compound singular proper noun	1	0.04
NPL/NN&	NPL without a NN&	1	0.04
PN=	Idiom tags for nominal pronoun	1	0.04
Po-	A conjoin when it does not begin with a coordinating conjunction (main category being prepositional phrase beginning with 'of').	1	0.04
Po&	Compound prepositional phrase beginning with 'of'.	1	0.04
Pq	A prepositional phrase with a wh-word	1	0.04
R/P&	R without P&	1	0.04
RB+	Second or subsequent conjoin of a compound adverb (general)	1	0.04
RN/RB&	RN without a RB&	1	0.04
VBN&	Compound past participle of lexical verb	1	0.04
VBN+	Second or subsequent conjoin of a compound past participle of lexical verb	1	0.04
X	Negative word 'not' when acting as an independent element of a clause.	1	0.04

Composition of The Constrained Test Sample

The constrained test sample consisted of 687 LPC sentences and contained 2,765 unique syntactic phrases. These are listed below.

Phrase	Description	Occurrences	% Test Data
N	Noun phrase.	872	31.54
V	Finite verb phrase.	467	16.89
P	Prepositional phrase.	246	8.90
S	Sentence.	200	7.23
R	Adverb phrase	126	4.56
J	Adjective phrase.	81	2.93
Na	Noun phrase marked as subject of the verb.	76	2.75
Po	Prepositional phrase beginning with 'of'.	75	2.71
Vi	Non-finite verb phrase of Ti clause.	74	2.68
Ti	to-infinitive clause.	68	2.46
Fn	Finite nominal clause	47	1.70
Fr	Relative clause.	35	1.27
Nq	Wh- noun phrase.	34	1.23
Vg	Non-finite verb phrase of Tg clause.	32	1.16
Tg	-ing clause.	31	1.12
S+	Second or subsequent conjoin of a compound sentence.	27	0.98
N&	Compound Noun phrase	19	0.69
Rq	Adverb phrase beginning with a wh-word	17	0.61
S&	Compound sentence.	17	0.61
Fa	Finite adverbial clause	16	0.58
N+	Second or subsequent conjoin of a compound Noun phrase.	15	0.54
Si	Interpolated sentence.	13	0.47
Vn	Non-finite verb phrase of Tn clause	13	0.47
NN&	Compound singular common noun	12	0.43
NN+	Second or subsequent conjoin of a compound singular common noun	12	0.43
Vr	Used when a verb phrase is split into two parts by subj-aux inversion.r=remaindr	11	0.40
U	tag used for an exclamatory word such as `Oh','yes', or 'no'.	9	0.33
G	Genitive phrase - phrase with two or more words acting as the genitive in a noun.	8	0.29
Tn	Past participle clause.	8	0.29

Phrase	Description	Occurrences	% Test Data
Vo	Used when a verb phrase is split into two parts by subj-aux inversion.o=operator	8	0.29
N-	A conjoin when it does not begin with a coordinating conjunction (main category being noun phrase).	7	0.25
RB=	Idiom tags for adverb (general)	7	0.25
IN=	Idiom tags for preposition (general)	6	0.22
JJ&	Compound adjective (general)	5	0.18
S-	A conjoin when it does not begin with a coordinating conjunction (main category being sentence).	5	0.18
Sq	piece of direct quotation.	5	0.18
Tb	`bare infinitive clause'.	5	0.18
E	Existential `there'.	4	0.14
JJ+	Second or subsequent conjoin of a compound adjective (general)	4	0.14
Ti&	Compound to-infinitive clause.	4	0.14
Fc	Comparative clause, normally beginning with `than' or `as'.	3	0.11
L	Verbless clause that is not introduced by with or by a subordinating conjunction	3	0.11
NNS+	Second or subsequent conjoin of a compound plural common noun	3	0.11
NN-	A conjoin when it does not begin with a coordinating conjunction (main category being singular common noun).	2	0.07
NNS&	Compound plural common noun	2	0.07
NP&	Compound singular proper noun	2	0.07
NP+	Second or subsequent conjoin of a compound singular proper noun	2	0.07
RP&	Compound adverbial particle	2	0.07
RP+	Second or subsequent conjoin of a compound adverbial particle	2	0.07
Tb&	Compound bare infinitive clause.	2	0.07
Tb+	Second or subsequent conjoin of a compound bare infinitive clause.	2	0.07
Ti-	A conjoin when it does not begin with a coordinating conjunction (main category being to-infinitive clause).	2	0.07
Ti+	Second or subsequent conjoin of a compound to-infinitive clause.	2	0.07
AP=	Idiom tags for post-determiner	1	0.04
CS=	Idiom tag for subordinating conjunctions	1	0.04
Dq	Determiner phrase beginning with wh-word.	1	0.04
JJ-	A conjoin when it does not begin with a coordinating conjunction (main category being adjective).	1	0.04
Jq	An adjective phrase beginning with	1	0.04

Phrase	Description	Occurrences	% Test Data
	wh-word		
L-	A conjoin when it does not begin with a coordinating conjunction (main category being verbless clause that is not introduced by with or by a subordinating conjunction).	1	0.04
L&	Compound verbless clause that is not introduced by with or by a subordinating conjunction.	1	0.04
NN/NNS&	NN without NNS&	1	0.04
P+	Second or subsequent conjoin of a compound prepositional phrase	1	0.04
Po&	Compound prepositional phrase beginning with 'of'.	1	0.04
Po+	Second or subsequent conjoin of a prep phrase beginning with 'of'.	1	0.04
Pq	A prepositional phrase with a wh-word	1	0.04
R&	Compound adverb phrase	1	0.04
RB-	A conjoin when it does not begin with a coordinating conjunction (main category being adverb).	1	0.04
RB&	Compound adverb	1	0.04

Appendix H

Composition of The Full Test Sample

The full unconstrained test sample consisted of 1,478 LPC sentences and contained 11,281 unique syntactic phrases. These are listed below.

Phrase	Description	Occurrences	% Test Data
N	Noun phrase.	3,559	31.55
V	Finite verb phrase.	1,725	15.29
P	Prepositional phrase.	1,120	9.93
R	Adverb phrase	591	5.24
S	Sentence.	512	4.54
Po	Prepositional phrase beginning with 'of'.	369	3.27
Na	Noun phrase marked as subject of the verb.	345	3.06
J	Adjective phrase.	267	2.37
Vi	Non-finite verb phrase of Ti clause.	237	2.10
Ti	to-infinitive clause.	222	1.97
Fn	Finite nominal clause	184	1.63
Fa	Finite adverbial clause	174	1.54
S+	Second or subsequent conjoin of a compound sentence.	171	1.52
Vg	Non-finite verb phrase of Tg clause.	143	1.27
Tg	-ing clause.	140	1.24
S&	Compound sentence.	128	1.13
Fr	Relative clause.	125	1.11
Nq	Wh- noun phrase.	124	1.10
N&	Compound Noun phrase	72	0.64
Vn	Non-finite verb phrase of Tn clause	71	0.63
Rq	Adverb phrase beginning with a wh-word	69	0.61
N+	Second or subsequent conjoin of a compound Noun phrase.	65	0.58
Tn	Past participle clause.	61	0.54
S-	A conjoin when it does not begin with a coordinating conjunction (main category being sentence).	58	0.51
Vr	Used when a verb phrase is split into two parts by subj-aux inversion.r=remaindr	45	0.40
G	Genitive phrase - phrase with two or more words acting as the genitive in a noun.	41	0.36
NN+	Second or subsequent conjoin of a compound singular common noun	41	0.36
NN&	Compound singular common noun	40	0.35

Phrase	Description	Occurrences	% Test Data
Sq	piece of direct quotation.	38	0.34
Vo	Used when a verb phrase is split into two parts by subj-aux inversion.o=operator	38	0.34
Si	Interpolated sentence.	33	0.29
JJ&	Compound adjective (general)	29	0.26
N-	A conjoin when it does not begin with a coordinating conjunction (main category being noun phrase).	25	0.22
E	Existential `there'.	23	0.20
RB=	Idiom tags for adverb (general)	22	0.20
JJ+	Second or subsequent conjoin of a compound adjective (general)	21	0.19
Fc	Comparative clause, normally beginning with `than' or `as'.	19	0.17
IN=	Idiom tags for preposition (general)	19	0.17
NNS+	Second or subsequent conjoin of a compound plural common noun	17	0.15
CS=	Idiom tag for subordinating conjunctions	16	0.14
U	tag used for an exclamatory word such as `Oh','yes', or 'no'.	15	0.13
NNS&	Compound plural common noun	14	0.12
NP&	Compound singular proper noun	13	0.12
NP+	Second or subsequent conjoin of a compound singular proper noun	13	0.12
JJ-	A conjoin when it does not begin with a coordinating conjunction (main category being adjective).	11	0.10
Tb	`bare infinitive clause'.	11	0.10
L	Verbless clause that is not introduced by with or by a subordinating conjunction	10	0.09
AP=	Idiom tags for post-determiner	9	0.08
Fn&	Compound finite nominal clause	9	0.08
Ti&	Compound to-infinitive clause.	9	0.08
Fn+	Second or subsequent conjoin of a compound finite nominal clause	7	0.06
W	Non-finite or verbless clause that is introduced by 'with'.	7	0.06
NN-	A conjoin when it does not begin with a coordinating conjunction (main category being singular common noun).	6	0.05
Ti+	Second or subsequent conjoin of a compound to-infinitive clause.	6	0.05
X	Negative word 'not' when acting as an independent element of a clause.	6	0.05
CD+	Second or subsequent conjoin of a compound cardinal number	4	0.04
Fa&	Compound finite adverbial clause.	5	0.04
P&	Compound prep phrase	4	0.04
Pq	A prepositional phrase with a wh-	4	0.04

Phrase	Description	Occurrences	% Test Data
	word		
Tf	Subject of infinitive which is introduced by 'for'	4	0.04
VB&	Compound base form of lexical verb.	5	0.04
VB+	Second or subsequent conjoin of a compound base form of lexical verb.	5	0.04
Fa+	Second or subsequent conjoin of a compound finite adverbial clause.	3	0.03
J+	Second or subsequent conjoin of an adjective phrase.	3	0.03
L&	Compound verbless clause that is not introduced by with or by a subordinating conjunction.	3	0.03
NNU=	Idiom tags for singular unit of measurement.	3	0.03
NR&	Compound singular adverbial noun.	3	0.03
NR+	Second or subsequent conjoin of a compound singular adverbial noun.	3	0.03
P-	A conjoin when it does not begin with a coordinating conjunction.	3	0.03
P+	Second or subsequent conjoin of a compound prepositional phrase	3	0.03
RP&	Compound adverbial particle	3	0.03
RP+	Second or subsequent conjoin of a compound adverbial particle	3	0.03
Tb&	Compound bare infinitive clause.	3	0.03
Tb+	Second or subsequent conjoin of a compound bare infinitive clause.	3	0.03
Ti-	A conjoin when it does not begin with a coordinating conjunction (main category being to-infinitive clause).	3	0.03
CD-	A conjoin when it does not begin with a coordinating conjunction (main category being a cardinal number).	2	0.02
CD&	Compound cardinal number.	2	0.02
CD1/CD&	CD1 without a CD&	2	0.02
Fa-	A conjoin when it does not begin with a coordinating conjunction (main category being a finite adverbial clause).	2	0.02
Fn-	A conjoin when it does not begin with a coordinating conjunction (main category being finite nominal clause)	2	0.02
IN&	Compound preposition.	2	0.02
IN+	Second or subsequent conjoin of a compound preposition.	2	0.02
J&	Compound adjective phrase	2	0.02
Jq	An adjective phrase beginning with wh-word	2	0.02
L-	A conjoin when it does not begin with a coordinating conjunction (main category being verbless clause that is not introduced by with or by a	2	0.02

Phrase	Description	Occurrences	% Test Data
	subordinating conjunction).		
NN/NNS&	NN without NNS&	2	0.02
NP-	A conjoin when it does not begin with a coordinating conjunction (main category being singular proper noun)	2	0.02
R&	Compound adverb phrase	2	0.02
RB&	Compound adverb.	2	0.02
TO=	Idiom tags for infinitival TO	2	0.02
VBN&	Compound past participle of lexical verb.	2	0.02
VBN+	Second or subsequent conjoin of a compound past participle of lexical verb.	2	0.02
AP+	Second or subsequent conjoin of a compound post-determiner.	1	0.01
D	Determiner phrase.	1	0.01
Dq	Determiner phrase beginning with wh-word.	1	0.01
Fr&	Compound relative clause.	1	0.01
Fr+	Second or subsequent conjoin of a compound relative clause.	1	0.01
J-	A conjoin when it does not begin with a coordinating conjunction (main category being adjective phrase).	1	0.01
JJ/AP&	JJ without an AP&	1	0.01
JJ=	Idiom tags for general adjective.	1	0.01
JJB/JJ&	JJB without a JJ&	1	0.01
JNP&	Compound adjective with word-initial cap.	1	0.01
JNP+	Second or subsequent conjoin of a compound adjective with word-initial cap	1	0.01
L+	Second or subsequent conjoin of a compound verbless clause that is not introduced by 'with'.	1	0.01
N/J&	N without a J&	1	0.01
NNS/NN&	NNS without a NN&	1	0.01
PN=	Idiom tags for nominal pronoun.	1	0.01
Po&	Compound prepositional phrase beginning with 'of'.	1	0.01
Po+	Second or subsequent conjoin of a prep phrase beginning with 'of'.	1	0.01
Poq	A prepositional phrase beginning with 'of' with wh-word.	1	0.01
PPLS=	Idiom tags for plural reflexive pronoun.	1	0.01
R+	Second or subsequent conjoin of a compound adverb phrase	1	0.01
RB-	A conjoin when it does not begin with a coordinating conjunction (main category being adverb).	1	0.01
RB+	Second or subsequent conjoin of a compound adverb.	1	0.01

Phrase	Description	Occurrences	% Test Data
Sq&	Compound piece of direct quotation.	1	0.01
Sq+	Second or subsequent conjoin of a compound piece of direct quotation.	1	0.01
Tb-	A conjoin when it does not begin with a coordinating conjunction (main category being 'bare infinitive clause').	1	0.01
Tg&	Compound –ing clause.	1	0.01
Tg/J&	Tg without a J&	1	0.01
Tg+	Second or subsequent conjoin of a compound –ing clause.	1	0.01
VBG&	Compound present participle or gerund of lexical verb.	1	0.01
VBG+	Second or subsequent conjoin of a compound present participle or gerund of lexical verb.	1	0.01
ZZ&	Compound letter of the alphabet.	1	0.01
ZZ+	Second or subsequent conjoin of a compound letter of the alphabet.	1	0.01

Appendix I

Parse Failures Made On The Training Sample

AO5 : 344
Desired parse : [S but_CC [N \0Mr_NPT Healey_NP N][V had_HVD V][N a_AT [JJ& partial_JJ [JJ+ and_CC limited_JJ JJ+]JJ&] success_NN N] ._. S]
Desired bracketing : (S(N)(V)(N(JJ&(JJ+))))

Concords attempt ....

State 1 : [N+ and_CC limited_JJ success_NN N+]
Description : Second or subsequent conjoin of a compound Label for a noun phrase, whether it is a single word or a sequence of words.

State 2 : [Vn& a_AT partial_JJ [N+ and_CC limited_JJ success_NN N+] Vn&]
Description : Compound Label for nonfinite verb phrases(VP). i.e VP's that are VP's of Ti,Tg and Tn.

State 3 : [V had_HVD V]
Description : A finite `verb phrase' i.e. one that excludes objects, complements, etc.

State 4 : [N 0Mr_NPT Healey_NP N]
Description : Label for a noun phrase, whether it is a single word or a sequence of words.

State 5 : [S- but_CC [N 0Mr_NPT Healey_NP N] [V had_HVD V] [Vn& a_AT partial_JJ [N+ and_CC limited_JJ success_NN N+] Vn&] S-]
Description : A conjoin when it does not begin with a coordinating conjunction (main category being sentence.).

C04 : 219
Desired parse : [S[Na she_PP3A Na][V plays_VBZ V][N the_ATI possessive_JJ mother_NN [Po of_INO [N a_AT man_NN [Fr[Nq whose_WP$ [N hobby_NN N]Nq][V revolves_VBZ V][P round_IN [N[G a_AT doll's_NN$ G] house_NN N]P]Fr]N]Po]N] ._. S]
Desired bracketing : (S(Na)(V)(N(Po(N(Fr(Nq(N))(V)(P(N(G))))))))

Concords attempt ....

D09 : 609
Desired parse : [S[Na we_PP1AS Na][V are_BER told_VBN V][Fn[E there_EX E][V is_BEZ V][N a_AT deep_JJ gulf_NN N][P between_IN [N the_ATI two_CD N]P]Fn] ._. S]
Desired bracketing : (S(Na)(V)(Fn(E)(V)(N)(P(N))))

Concords attempt ....

State 1 : [N the_ATI two_CD N]
Description : Label for a noun phrase, whether it is a single word or a sequence of words.

State 2 : [P between_IN [N the_ATI two_CD N] P]
Description : A prepositional phrase, e.g. `in London' or `on arriving at the station'.

State 3 : [N a_AT deep_JJ gulf_NN [P between_IN [N the_ATI two_CD N] P] N]
Description : Label for a noun phrase, whether it is a single word or a sequence of words.

State 4 : [V is_BEZ V]

Description : A finite `verb phrase' i.e. one that excludes objects, complements, etc.

State 5 : [E there_EX E]
Description : label used for existential `there' i.e 'There' is nothing wrong.

State 6 : [Fn [E there_EX E] [V is_BEZ V] [N a_AT deep_JJ gulf_NN [P between_IN [N the_ATI two_CD N] P] N] Fn]
Description : finite nominal clause (subordinate clause functioning in the pos of a Noun PH

State 7 : [V are_BER told_VBN V]
Description : A finite `verb phrase' i.e. one that excludes objects, complements, etc.

State 8 : [Na we_PP1AS Na]
Description : A noun phrase marked as subject of the verb.

E04 : 224
Desired parse : [S[Na they_PP3AS Na][V are_BER V][J& very_QL simple_JJ ,_, [J- cheap_JJ J-] [J+ and_CC easy_JJ [Ti[Vi to_TO make_VB Vi]Ti]J+]J&] ._. S]
Desired bracketing : (S(Na)(V)(J&(J-)(J+(Ti(Vi)))))

Concords attempt ....

State 1 : [Vi to_TO make_VB Vi]
Description : Label for nonfinite verb phrases(VP). i.e VP's that are VP's of Ti,Tg and Tn.

State 2 : [Ti [Vi to_TO make_VB Vi] Ti]
Description : to-infinitive clause.

State 3 : [J+ and_CC easy_JJ [Ti [Vi to_TO make_VB Vi] Ti] J+]
Description : Second or subsequent conjoin of a compound An adjective phrase such as `happy', `very tall', `too happy for words', etc

State 4 : [J& cheap_JJ [J+ and_CC easy_JJ [Ti [Vi to_TO make_VB Vi] Ti] J+] J&]
Description : Compound An adjective phrase such as `happy', `very tall', `too happy for words', etc

State 5 : [Vo simple_JJ Vo]
Description : Used when a verb phrase is split into two parts by subj-aux inversion.o=operator

State 6 : [V very_QL V]
Description : A finite `verb phrase' i.e. one that excludes objects, complements, etc.

State 7 : [V are_BER V]
Description : A finite `verb phrase' i.e. one that excludes objects, complements, etc.

State 8 : [S [V are_BER V] [V very_QL V] [Vo simple_JJ Vo] [J& cheap_JJ [J+ and_CC easy_JJ [Ti [Vi to_TO make_VB Vi] Ti] J+] J&] S]
Description : sentence.

State 9 : [Jq they_PP3AS Jq]
Description : A phrase beginning with a wh-word e.g. `How old'.

L07 : 859
Desired parse : [S[E there_EX E][V 's_BEZ V][N a_AT place_NN [P just_RB below_IN [N Bletcham_NP N]P]N] ,_, [P near_IN [N the_ATI footbridge_NN N]P] ._. S] **'_**'
Desired bracketing : (S(E)(V)(N(P(N)))(P(N)))

Concords attempt ....

State 1 : [N the_ATI footbridge_NN N]
Description : Label for a noun phrase, whether it is a single word or a sequence of words.

State 2 : [P near_IN [N the_ATI footbridge_NN N] P]
Description : A prepositional phrase, e.g. `in London' or `on arriving at the station'.

State 3 : [N Bletcham_NP N]
Description : Label for a noun phrase, whether it is a single word or a sequence of words.

State 4 : [IN= just_RB below_IN IN=]
Description : Idiom tags for preposition (general)

State 5 : [Rq- place_NN IN= Rq-]
Description : A conjoin when it does not begin with a coordinating conjunction (main category being an adverb phrase beginning with a wh-word, e.g. `How do you feel?',or `how long').

State 6 : [V a_AT [Rq- place_NN IN= Rq-] V]
Description : A finite `verb phrase' i.e. one that excludes objects, complements, etc.

State 7 : [T [V a_AT [Rq- place_NN IN= Rq-] V] [N Bletcham_NP N] [P near_IN [N the_ATI footbridge_NN N] P] T]
Description : nonfinite clause.

State 8 : [V s_BEZ V]
Description : A finite `verb phrase' i.e. one that excludes objects, complements, etc.

L08 : 939
Desired parse : [S and_CC ,_, [R[RN/RB& now_RN [RB+ and_CC again_RB RB+]RN/RB&]R] ,_, [N a_AT sudden_JJ feeling_NN [Po of_INO [N tension_NN [P in_IN [N the_ATI air_NN N]P]N]Po]N] ._. S]
Desired bracketing : (S(R(RN/RB&(RB+)))(N(Po(N(P(N))))))

Concords attempt ....

State 1 : [N the_ATI air_NN N]
Description : Label for a noun phrase, whether it is a single word or a sequence of words.

State 2 : [P in_IN [N the_ATI air_NN N] P]
Description : A prepositional phrase, e.g. `in London' or `on arriving at the station'.

State 3 : [N tension_NN [P in_IN [N the_ATI air_NN N] P] N]
Description : Label for a noun phrase, whether it is a single word or a sequence of words.

State 4 : [Po of_INO [N tension_NN [P in_IN [N the_ATI air_NN N] P] N] Po]
Description : A prepositional phrase beginning with the preposition 'of'.

State 5 : [N a_AT sudden_JJ feeling_NN [Po of_INO [N tension_NN [P in_IN [N the_ATI air_NN N] P] N] Po] N]
Description : Label for a noun phrase, whether it is a single word or a sequence of words.

State 6 : [R again_RB R]
Description : An adverb phrase, e.g. `there',`quickly' or a sequence such as `quite often' etc

State 7 : [S and_CC [R again_RB R] [N a_AT sudden_JJ feeling_NN [Po of_INO [N tension_NN [P in_IN [N the_ATI air_NN N] P] N] Po] N] S]
Description : sentence.

State 8 : [RB+ and_CC now_RN RB+]
Description : Second or subsequent conjoin of a compound adverb (general)

State 9 : [R RB+ R]
Description : An adverb phrase, e.g. `there',`quickly' or a sequence such as `quite often' etc

M01 : 87

Desired parse : [S[Na we_PP1AS Na][V must_MD learn_VB V][N all_ABN [Fr[Na we_PP1AS Na][V can_MD V]Fr]N][P about_IN [N them_PP3OS N]P] ._. S]
Desired bracketing : (S(Na)(V)(N(Fr(Na)(V)))(P(N)))

Concords attempt ....

State 1 : [N them_PP3OS N]
Description : Label for a noun phrase, whether it is a single word or a sequence of words.

State 2 : [V can_MD about_IN V]
Description : A finite `verb phrase' i.e. one that excludes objects, complements, etc.

State 3 : [Na we_PP1AS Na]
Description : A noun phrase marked as subject of the verb.

State 4 : [N all_ABN N]
Description : Label for a noun phrase, whether it is a single word or a sequence of words.

State 5 : [Fn [N all_ABN N] [Na we_PP1AS Na] [V can_MD about_IN V] [N them_PP3OS N] Fn]
Description : finite nominal clause (subordinate clause functioning in the pos of a Noun PH

State 6 : [V must_MD learn_VB V]
Description : A finite `verb phrase' i.e. one that excludes objects, complements, etc.

State 7 : [Na we_PP1AS Na]
Description : A noun phrase marked as subject of the verb.

N01 : 46
Desired parse : [S[Na he_PP3A Na][V tried_VBD V][Ti[Vi to_TO imagine_VB Vi][Fn[Nq what_WDT Nq][N the_ATI Italians_NNPS N][V would_MD do_DO V][R next_RB R]Fn]Ti] ._. S]
Desired bracketing : (S(Na)(V)(Ti(Vi)(Fn(Nq)(N)(V)(R))))

Concords attempt ....

State 1 : [R next_RB R]
Description : An adverb phrase, e.g. `there',`quickly' or a sequence such as `quite often' etc

State 2 : [V would_MD do_DO V]
Description : A finite `verb phrase' i.e. one that excludes objects, complements, etc.

State 3 : [N the_ATI Italians_NNPS N]
Description : Label for a noun phrase, whether it is a single word or a sequence of words.

State 4 : [Nq what_WDT Nq]
Description : A wh- noun phrase, such as `who', `which', `which car', `what time' etc.

State 5 : [Fn [Nq what_WDT Nq] [N the_ATI Italians_NNPS N] [V would_MD do_DO V] [R next_RB R] Fn]
Description : finite nominal clause (subordinate clause functioning in the pos of a Noun PH

State 6 : [V tried_VBD to_TO imagine_VB V]
Description : A finite `verb phrase' i.e. one that excludes objects, complements, etc.

State 7 : [Na he_PP3A Na]
Description : A noun phrase marked as subject of the verb.

N04 : 318
Desired parse : [S[R ever_RB R][V known_VBN V][N us_PP1OS N][Tb[V bowl_VB V][N a_AT wide_NN N][P about_IN [N your_PP$ service_NN N]P]Tb] ?_? S]
Desired bracketing : (S(R)(V)(N)(Tb(V)(N)(P(N))))

Concords attempt ....

State 1 : [N your_PP$ service_NN N]
Description : Label for a noun phrase, whether it is a single word or a sequence of words.

State 2 : [P about_IN [N your_PP$ service_NN N] P]
Description : A prepositional phrase, e.g. `in London' or `on arriving at the station'.

State 3 : [N a_AT wide_NN [P about_IN [N your_PP$ service_NN N] P] N]
Description : Label for a noun phrase, whether it is a single word or a sequence of words.

State 4 : [V bowl_VB V]
Description : A finite `verb phrase' i.e. one that excludes objects, complements, etc.

State 5 : [N us_PP1OS N]
Description : Label for a noun phrase, whether it is a single word or a sequence of words.

State 6 : [V known_VBN V]
Description : A finite `verb phrase' i.e. one that excludes objects, complements, etc.

State 7 : [S [V known_VBN V] [N us_PP1OS N] [V bowl_VB V] [N a_AT wide_NN [P about_IN [N your_PP$ service_NN N] P] N] S]
Description : sentence.

State 8 : [R ever_RB R]
Description : An adverb phrase, e.g. `there',`quickly' or a sequence such as `quite often' etc

N05 : 462
Desired parse : [S[Na he_PP3A Na][V was_BEDZ V][J& not_XNOT only_RB puzzled_JJ ,_, [J+ but_CC alarmed_JJ J+]J&] ._. S]
Desired bracketing : (S(Na)(V)(J&(J+)))

Concords attempt ....

State 1 : [J+ but_CC alarmed_JJ J+]
Description : Second or subsequent conjoin of a compound An adjective phrase such as `happy', `very tall', `too happy for words', etc

State 2 : [J only_RB puzzled_JJ J]
Description : An adjective phrase such as `happy', `very tall', `too happy for words', etc

State 3 : [Rq was_BEDZ not_XNOT [J only_RB puzzled_JJ J] [J+ but_CC alarmed_JJ J+] Rq]
Description : an adverb phrase beginning with a wh-word, e.g. `How do you feel?',or `how long'

State 4 : [Vi he_PP3A [Rq was_BEDZ not_XNOT [J only_RB puzzled_JJ J] [J+ but_CC alarmed_JJ J+] Rq] Vi]
Description : Label for nonfinite verb phrases(VP). i.e VP's that are VP's of Ti,Tg and Tn.

State 5 : [Ti [Vi he_PP3A [Rq was_BEDZ not_XNOT [J only_RB puzzled_JJ J] [J+ but_CC alarmed_JJ J+] Rq] Vi] Ti]
Description : to-infinitive clause.

P10 : 1326
Desired parse : [S[Na they_PP3AS Na][V were_BED V][J content_JJ [Ti[R merely_RB R][Vi to_TO be_BE Vi][R together_RB R]Ti]J] ._. S]
Desired bracketing : (S(Na)(V)(J(Ti(R)(Vi)(R))))

Concords attempt ....

State 1 : [V be_BE together_RB V]
Description : A finite `verb phrase' i.e. one that excludes objects, complements, etc.

R09 : 669
Desired parse : [S[N one_CD1 N] ,_, [Si[Na I_PP1A Na][V think_VB V]Si] ,_, [V is_BEZ V][N an_AT expert_NN [P on_IN [N Russia_NP N]P]N] ._. S]
Desired bracketing : (S(N)(Si(Na)(V))(V)(N(P(N))))

Concords attempt ....

State 1 : [N Russia_NP N]
Description : Label for a noun phrase, whether it is a single word or a sequence of words.

State 2 : [P on_IN [N Russia_NP N] P]
Description : A prepositional phrase, e.g. `in London' or `on arriving at the station'.

State 3 : [N an_AT expert_NN [P on_IN [N Russia_NP N] P] N]
Description : Label for a noun phrase, whether it is a single word or a sequence of words.

State 4 : [V is_BEZ V]
Description : A finite `verb phrase' i.e. one that excludes objects, complements, etc.

F03 : 269
Desired parse : [S and_CC [N you_PP2 N][V want_VB V][Ti[Vi to_TO get_VB Vi][N it_PP3 N][P from_IN [N the_ATI word_NN go_NC N]P]Ti] !_! S]
Desired bracketing : (S(N)(V)(Ti(Vi)(N)(P(N))))

Concords attempt ....

State 1 : [N the_ATI word_NN go_NC N]
Description : Label for a noun phrase, whether it is a single word or a sequence of words.

State 2 : [P from_IN [N the_ATI word_NN go_NC N] P]
Description : A prepositional phrase, e.g. `in London' or `on arriving at the station'.

State 3 : [N it_PP3 N]
Description : Label for a noun phrase, whether it is a single word or a sequence of words.

State 4 : [Vi to_TO get_VB Vi]
Description : Label for nonfinite verb phrases(VP). i.e VP's that are VP's of Ti,Tg and Tn.

State 5 : [V want_VB V]
Description : A finite `verb phrase' i.e. one that excludes objects, complements, etc.

F09 : 525
Desired parse : [S[Na I_PP1A Na][V felt_VBD V][Tn[Vn impelled_VBN Vn][Ti&[Vi to_TO read_VB Vi][R on_RP R][Ti+ and_CC [V share_VB V][N his_PP$ experiences_NNS N]Ti+]Ti&]Tn] ._. S]
Desired bracketing : (S(Na)(V)(Tn(Vn)(Ti&(Vi)(R)(Ti+(V)(N)))))

Concords attempt ....

State 1 : [N his_PP$ experiences_NNS N]
Description : Label for a noun phrase, whether it is a single word or a sequence of words.

State 2 : [V share_VB V]
Description : A finite `verb phrase' i.e. one that excludes objects, complements, etc.

State 3 : [Ti+ and_CC [V share_VB V] [N his_PP$ experiences_NNS N] Ti+]
Description : Second or subsequent conjoin of a compound to-infinitive clause.

State 4 : [Rq on_RP [Ti+ and_CC [V share_VB V] [N his_PP$ experiences_NNS N] Ti+] Rq]

Description : an adverb phrase beginning with a wh-word, e.g. `How do you feel?',or `how long'

J03 : 174
Desired parse : [S[Nq what_WDT Nq][V are_BER V][N the_ATI factors_NNS [Fr[Nq which_WDT Nq][V have_HV operated_VBN V][P in_IN [N the_ATI differentiation_NN [Po of_INO [N these_DTS soils_NNS N]Po]N]P]Fr]N] ?_? S]
Desired bracketing : (S(Nq)(V)(N(Fr(Nq)(V)(P(N(Po(N))))))

Concords attempt ....

State 1 : [N these_DTS soils_NNS N]
Description : Label for a noun phrase, whether it is a single word or a sequence of words.

State 2 : [Po of_INO [N these_DTS soils_NNS N] Po]
Description : A prepositional phrase beginning with the preposition 'of'.

State 3 : [N the_ATI differentiation_NN [Po of_INO [N these_DTS soils_NNS N] Po] N]
Description : Label for a noun phrase, whether it is a single word or a sequence of words.

State 4 : [P in_IN [N the_ATI differentiation_NN [Po of_INO [N these_DTS soils_NNS N] Po] N] P]
Description : A prepositional phrase, e.g. `in London' or `on arriving at the station'.

State 5 : [V have_HV operated_VBN V]
Description : A finite `verb phrase' i.e. one that excludes objects, complements, etc.

State 6 : [Vo factors_NNS which_WDT Vo]
Description : Used when a verb phrase is split into two parts by subj-aux inversion.o=operator

State 7 : [N the_ATI [Vo factors_NNS which_WDT Vo] N]
Description : Label for a noun phrase, whether it is a single word or a sequence of words.

State 8 : [V are_BER V]
Description : A finite `verb phrase' i.e. one that excludes objects, complements, etc.

State 9 : [S [V are_BER V] [N the_ATI [Vo factors_NNS which_WDT Vo] N] [V have_HV operated_VBN V] [P in_IN [N the_ATI differentiation_NN [Po of_INO [N these_DTS soils_NNS N] Po] N] P] S]
Description : sentence.

State 10 : [Nq what_WDT Nq]
Description : A wh- noun phrase, such as `who', `which', `which car', `what time' etc.

K02 : 66
Desired parse : [S[Na I_PP1A Na][V had_HVD not_XNOT known_VBN V][Fn[Na he_PP3A Na][V was_BEDZ V][J so_QL good-natured_JJ J]Fn] ._. S]
Desired bracketing : (S(Na)(V)(Fn(Na)(V)(J)))

Concords attempt ....

State 1 : [J so_QL goodnatured_JJ J]
Description : An adjective phrase such as `happy', `very tall', `too happy for words', etc

State 2 : [R he_PP3A was_BEDZ R]
Description : An adverb phrase, e.g. `there',`quickly' or a sequence such as `quite often' etc

State 3 : [V not_XNOT known_VBN V]
Description : A finite `verb phrase' i.e. one that excludes objects, complements, etc.

K03 : 218
Desired parse : [S[N it_PP3 N][V 's_BEZ V][N a_AT sign_NN [Po of_INO [N life_NN N]Po]N] *-_*- [J very_QL encouraging_JJ J] !_! S]

Desired bracketing : (S(N)(V)(N(Po(N)))(J))

Concords attempt ....

K04 : 250

Desired parse : [S[N you_PP2 N][V would_MD be_BE V][N the_ATI last_AP man_NN [Fr[Na I_PP1A Na][V would_MD ask_VB V]Fr]N] ._. S]

Desired bracketing : (S(N)(V)(N(Fr(Na)(V))))

Concords attempt ....

State 1	: [V would_MD ask_VB V]
Description	: A finite `verb phrase' i.e. one that excludes objects, complements, etc.
State 2	: [Na I_PP1A Na]
Description	: A noun phrase marked as subject of the verb.
State 3	: [N last_AP man_NN N]
Description	: Label for a noun phrase, whether it is a single word or a sequence of words.
State 4	: [R the_ATI [N last_AP man_NN N] R]
Description	: An adverb phrase, e.g. `there',`quickly' or a sequence such as `quite often' etc
State 5	: [V be_BE V]
Description	: A finite `verb phrase' i.e. one that excludes objects, complements, etc.

K08 : 794

Desired parse : [S[Na I_PP1A Na][V think_VB V][Fn[N it_PP3 N][V gave_VBD V][N him_PP3O N][N the_ATI feeling_NN [Po of_INO [Tg[Vg being_BEG Vg][P& in_IN [N his_PP$ office_NN N] *-_*- [P+ and_CC more_RBR at_IN [N home_NR N]P+]P&]Tg]Po]N]Fn] ._. S] **'_**'

Desired bracketing : (S(Na)(V)(Fn(N)(V)(N)(N(Po(Tg(Vg)(P&(N)(P+(N)))))))

Concords attempt ....

State 1	: [N home_NR N]
Description	: Label for a noun phrase, whether it is a single word or a sequence of words.
State 2	: [Rq+ and_CC more_RBR at_IN Rq+]
Description	: Second or subsequent conjoin of a compound an adverb phrase beginning with a wh-word, e.g. `How do you feel?',or `how long'
State 3	: [N& his_PP$ office_NN [Rq+ and_CC more_RBR at_IN Rq+] N&]
Description	: Compound Label for a noun phrase, whether it is a single word or a sequence of words.
State 4	: [V being_BEG in_IN V]
Description	: A finite `verb phrase' i.e. one that excludes objects, complements, etc.

L05 : 571

Desired parse : [S[N you_PP2 N][V know_VB V][R as_QL well_RB [Fc as_CS [Na I_PP1A Na][V do_DO V]Fc]R][Fn[N Rose_NP N][V did_DOD n't_XNOT kill_VB V][N herself_PPL N]Fn] ._. S]

Desired bracketing : (S(N)(V)(R(Fc(Na)(V)))(Fn(N)(V)(N)))

Concords attempt ....

State 1	: [N herself_PPL N]
Description	: Label for a noun phrase, whether it is a single word or a sequence of words.
State 2	: [V did_DOD nt_XNOT kill_VB V]
Description	: A finite `verb phrase' i.e. one that excludes objects, complements, etc.
State 3	: [N Rose_NP N]

Description	: Label for a noun phrase, whether it is a single word or a sequence of words.
State 4	: [V do_DO V]
Description	: A finite `verb phrase' i.e. one that excludes objects, complements, etc.
State 5	: [T [V do_DO V] [N Rose_NP N] T]
Description	: nonfinite clause.
State 6	: [Na I_PP1A Na]
Description	: A noun phrase marked as subject of the verb.

Appendix J

A Sample of Failed Parses for The Constrained Test Sample

N02 : 135
Desired parse : [S and_CC [Na he_PP3A Na][V did_DOD kiss_VB V][N me_PP1O N] ._. S]
Desired bracketing : (S(Na)(V)(N))

Concords attempt ....

State 1	: [N me_PP1O N]
Description	: Label for a noun phrase, whether it is a single word or a sequence of words.

N06 : 591
Desired parse : [S&[Na she_PP3A Na][V came_VBD V][R along_RP R][S+ and_CC [V whickered_VBD V][Fa[Rq when_WRB Rq][Na she_PP3A Na][V saw_VBD V][N me_PP1O N]Fa]S+] ._. S&]
Desired bracketing : (S&(Na)(V)(R)(S+(V)(Fa(Rq)(Na)(V)(N))))

Concords attempt ....

State 1	: [N me_PP1O N]
Description	: Label for a noun phrase, whether it is a single word or a sequence of words.
State 2	: [V saw_VBD V]
Description	: A finite `verb phrase' i.e. one that excludes objects, complements, etc.
State 3	: [Rq she_PP3A Rq]
Description	: an adverb phrase beginning with a wh-word, e.g. `How do you feel?',or `how long'
State 4	: [Na when_WRB [Rq she_PP3A Rq] Na]
Description	: A noun phrase marked as subject of the verb.
State 5	: [V whickered_VBD V]
Description	: A finite `verb phrase' i.e. one that excludes objects, complements, etc.

N07 : 719
Desired parse : [S[V reckon_VB V][Fn[Na we_PP1AS Na][V 'll_MD just_RB have_HV V][Ti[Vi to_TO let_VB Vi][N the_ATI matter_NN N][Tb[V solve_VB V][N itself_PPL N]Tb]Ti]Fn] ._. S] **'_**'
Desired bracketing : (S(V)(Fn(Na)(V)(Ti(Vi)(N)(Tb(V)(N)))))

Concords attempt ....

State 1	: [Vg itself_PPL Vg]
Description	: Label for nonfinite verb phrases(VP). i.e VP's that are VP's of Ti,Tg and Tn.
State 2	: [Tg [Vg itself_PPL Vg] Tg]
Description	: -ing clause.
State 3	: [V solve_VB V]
Description	: A finite `verb phrase' i.e. one that excludes objects, complements, etc.
State 4	: [N the_ATI matter_NN N]
Description	: Label for a noun phrase, whether it is a single word or a sequence of words.

State 5	: [Vn let_VB Vn]
Description	: Label for nonfinite verb phrases(VP). i.e VP's that are VP's of Ti,Tg and Tn.
State 6	: [L [Vn let_VB Vn] [N the_ATI matter_NN N] L]
Description	: verbless clause that is not introduced by with or by a subordinating conjunction.
State 7	: [Vg to_TO Vg]
Description	: Label for nonfinite verb phrases(VP). i.e VP's that are VP's of Ti,Tg and Tn.
State 8	: [Ti [Vg to_TO Vg] [L [Vn let_VB Vn] [N the_ATI matter_NN N] L] Ti]
Description	: to-infinitive clause.
State 9	: [V have_HV V]
Description	: A finite `verb phrase' i.e. one that excludes objects, complements, etc.
State 10	: [Fn [V have_HV V] [Ti [Vg to_TO Vg] [L [Vn let_VB Vn] [N the_ATI matter_NN N] L] Ti] Fn]
Description	: finite nominal clause (subordinate clause functioning in the pos of a Noun PH

N09 : 1095

Desired parse : [S[Na I_PP1A Na][V know_VB V][Fn[N you_PP2 N][V will_MD do_DO V][N it_PP3 N] ..._... [R well_RB R]Fn] ._. S]
Desired bracketing : (S(Na)(V)(Fn(N)(V)(N)(R)))

Concords attempt ....

State 1	: [R well_RB R]
Description	: An adverb phrase, e.g. `there',`quickly' or a sequence such as `quite often' etc
State 2	: [N it_PP3 N]
Description	: Label for a noun phrase, whether it is a single word or a sequence of words.
State 3	: [V will_MD do_DO V]
Description	: A finite `verb phrase' i.e. one that excludes objects, complements, etc.
State 4	: [N you_PP2 N]
Description	: Label for a noun phrase, whether it is a single word or a sequence of words.
State 5	: [Fn [N you_PP2 N] [V will_MD do_DO V] [N it_PP3 N] [R well_RB R] Fn]
Description	: finite nominal clause (subordinate clause functioning in the pos of a Noun PH
State 6	: [V know_VB V]
Description	: A finite `verb phrase' i.e. one that excludes objects, complements, etc.
State 7	: [Na I_PP1A Na]
Description	: A noun phrase marked as subject of the verb.

P03 : 255

Desired parse : [S&[Na she_PP3A Na][V was_BEDZ engaged_VBN V][P to_IN [N Nigel_NP N]P] ,_, [S+[V had_HVD been_BEN V][P for_IN [N two_CD years_NNS N]P]S+] ._. S&]
Desired bracketing : (S&(Na)(V)(P(N))(S+(V)(P(N))))

Concords attempt ....

State 1	: [N two_CD years_NNS N]
Description	: Label for a noun phrase, whether it is a single word or a sequence of words.
State 2	: [P for_IN [N two_CD years_NNS N] P]
Description	: A prepositional phrase, e.g. `in London' or `on arriving at the station'.
State 3	: [V had_HVD been_BEN V]
Description	: A finite `verb phrase' i.e. one that excludes objects, complements, etc.

State 4 : [N Nigel_NP N]
Description : Label for a noun phrase, whether it is a single word or a sequence of words.

State 5 : [P to_IN [N Nigel_NP N] P]
Description : A prepositional phrase, e.g. `in London' or `on arriving at the station'.

State 6 : [Vn engaged_VBN Vn]
Description : Label for nonfinite verb phrases(VP). i.e VP's that are VP's of Ti,Tg and Tn.

P05 : 583
Desired parse : [S[R always_RB R][J glad_JJ [Ti[Vi to_TO hear_VB Vi][Po of_INO [N[G a_AT friend's_NN$ G] good_JJ fortune_NN N]Po]Ti]J] !_! S] **'_**'
Desired bracketing : (S(R)(J(Ti(Vi)(Po(N(G))))))

Concords attempt ....

State 1 : [N a_AT friends_NN$ good_JJ fortune_NN N]
Description : Label for a noun phrase, whether it is a single word or a sequence of words.

State 2 : [Po of_INO [N a_AT friends_NN$ good_JJ fortune_NN N] Po]
Description : A prepositional phrase beginning with the preposition 'of'.

State 3 : [Vi to_TO hear_VB Vi]
Description : Label for nonfinite verb phrases(VP). i.e VP's that are VP's of Ti,Tg and Tn.

State 4 : [Ti [Vi to_TO hear_VB Vi] [Po of_INO [N a_AT friends_NN$ good_JJ fortune_NN N] Po] Ti]
Description : to-infinitive clause.

State 5 : [J always_RB glad_JJ J]
Description : An adjective phrase such as `happy', `very tall', `too happy for words', etc

P09 : 1071
Desired parse : *'_*' [S[Vo will_MD Vo][N you_PP2 N][Vr be_BE Vr][R& here_RN [P+ or_CC at_IN [N the_ATI house_NN N][N tomorrow_NR N]P+]R&] ?_? S] **'_**'
Desired bracketing : (S(Vo)(N)(Vr)(R&(P+(N)(N))))

Concords attempt ....

State 1 : [N the_ATI house_NN tomorrow_NR N]
Description : Label for a noun phrase, whether it is a single word or a sequence of words.

State 2 : [P at_IN [N the_ATI house_NN tomorrow_NR N] P]
Description : A prepositional phrase, e.g. `in London' or `on arriving at the station'.

R09 : 694
Desired parse : [S[N you_PP2 N][V 'd_MD never_RB think_VB V][Fn[Na we_PP1AS Na][V were_BED V][N a_AT mile_NN [P from_IN [N the_ATI sea_NN N]P]N] ,_, [Si[V would_MD V][N you_PP2 N] ,_, [N sir_NN N]Si]Fn] ?_? S]
Desired bracketing : (S(N)(V)(Fn(Na)(V)(N(P(N)))(Si(V)(N)(N))))

Concords attempt ....

State 1 : [N you_PP2 sir_NN N]
Description : Label for a noun phrase, whether it is a single word or a sequence of words.

State 2 : [V would_MD V]
Description : A finite `verb phrase' i.e. one that excludes objects, complements, etc.

State 3 : [N the_ATI sea_NN N]
Description : Label for a noun phrase, whether it is a single word or a sequence of words.

State 4 : [P from_IN [N the_ATI sea_NN N] P]
Description : A prepositional phrase, e.g. `in London' or `on arriving at the station'.

State 5 : [Ti [P from_IN [N the_ATI sea_NN N] P] [V would_MD V] [N you_PP2 sir_NN N] Ti]
Description : to-infinitive clause.

State 6 : [N a_AT mile_NN [Ti [P from_IN [N the_ATI sea_NN N] P] [V would_MD V] [N you_PP2 sir_NN N] Ti] N]
Description : Label for a noun phrase, whether it is a single word or a sequence of words.

A05 : 329
Desired parse : [S[N Britain_NP N][V could_MD not_XNOT be_BE V][N a_AT party_NN [P to_IN [N an_AT imposed_JJ division_NN N]P]N] ._. S]
Desired bracketing : (S(N)(V)(N(P(N))))

Concords attempt ....

State 1 : [N an_AT imposed_JJ division_NN N]
Description : Label for a noun phrase, whether it is a single word or a sequence of words.

State 2 : [P to_IN [N an_AT imposed_JJ division_NN N] P]
Description : A prepositional phrase, e.g. `in London' or `on arriving at the station'.

State 3 : [N a_AT party_NN [P to_IN [N an_AT imposed_JJ division_NN N] P] N]
Description : Label for a noun phrase, whether it is a single word or a sequence of words.

State 4 : [V be_BE V]
Description : A finite `verb phrase' i.e. one that excludes objects, complements, etc.

A09 : 601
Desired parse : [S&[Na she_PP3A Na][V has_HVZ bought_VBN V][N a_AT house_NN [P in_IN [N Belgravia_NP N]P]N] ,_, [S+ and_CC [V hopes_VBZ V] [Ti[Vi to_TO move_VB Vi][R in_RP R][P after_IN [N Easter_NP N] P]Ti]S+] ._. S&]
Desired bracketing : (S&(Na)(V)(N(P(N)))(S+(V)(Ti(Vi)(R)(P(N)))))

Concords attempt ....

State 1 : [N Easter_NP N]
Description : Label for a noun phrase, whether it is a single word or a sequence of words.

State 2 : [P after_IN [N Easter_NP N] P]
Description : A prepositional phrase, e.g. `in London' or `on arriving at the station'.

State 3 : [R in_RP R]
Description : An adverb phrase, e.g. `there',`quickly' or a sequence such as `quite often' etc

State 4 : [Vi to_TO move_VB Vi]
Description : Label for nonfinite verb phrases(VP). i.e VP's that are VP's of Ti,Tg and Tn.

State 5 : [Ti [Vi to_TO move_VB Vi] [R in_RP R] [P after_IN [N Easter_NP N] P] Ti]
Description : to-infinitive clause.

State 6 : [RB+ and_CC hopes_VBZ RB+]
Description : Second or subsequent conjoin of a compound adverb (general)

State 7 : [V& Belgravia_NP RB+ V&]
Description : Compound A finite `verb phrase' i.e. one that excludes objects, complements, etc.

State 8 : [P in_IN P]
Description : A prepositional phrase, e.g. `in London' or `on arriving at the station'.

State 9 : [N house_NN [P in_IN P] N]

Description : Label for a noun phrase, whether it is a single word or a sequence of words.

State 10 : [Pq a_AT [N house_NN [P in_IN P] N] Pq]
Description : A prepositional phrase with a wh-word, e.g. `on whose behalf', `in which case'.

State 11 : [Fn [Pq a_AT [N house_NN [P in_IN P] N] Pq] [V& Belgravia_NP RB+ V&] [Ti [Vi to_TO move_VB Vi] [R in_RP R] [P after_IN [N Easter_NP N] P] Ti] Fn]
Description : finite nominal clause (subordinate clause functioning in the pos of a Noun PH

State 12 : [V has_HVZ bought_VBN V]
Description : A finite `verb phrase' i.e. one that excludes objects, complements, etc.

State 13 : [Na she_PP3A Na]
Description : A noun phrase marked as subject of the verb.

A10 : 681

Desired parse : [S[N the_ATI royal_JJ family_NN [Tg[Vg driving_VBG Vg][R up_RP R][N the_ATI course_NN N]Tg]N] ..._... ._. S]
Desired bracketing : (S(N(Tg(Vg)(R)(N))))

Concords attempt ....

State 1 : [N the_ATI course_NN N]
Description : Label for a noun phrase, whether it is a single word or a sequence of words.

State 2 : [R up_RP R]
Description : An adverb phrase, e.g. `there',`quickly' or a sequence such as `quite often' etc

State 3 : [V driving_VBG V]
Description : A finite `verb phrase' i.e. one that excludes objects, complements, etc.

State 4 : [N family_NN N]
Description : Label for a noun phrase, whether it is a single word or a sequence of words.

B03 : 193

Desired parse : [S[N the_ATI purpose_NN N][V is_BEZ V][Ti[Vi to_TO give_VB Vi][N fahrenheit_NN N][N the_ATI knock_NN out_RP N]Ti] ._. S]
Desired bracketing : (S(N)(V)(Ti(Vi)(N)(N)))

Concords attempt ....

State 1 : [Jq out_RP Jq]
Description : A phrase beginning with a wh-word e.g. `How old'.

State 2 : [N fahrenheit_NN the_ATI knock_NN N]
Description : Label for a noun phrase, whether it is a single word or a sequence of words.

State 3 : [Vi to_TO give_VB Vi]
Description : Label for nonfinite verb phrases(VP). i.e VP's that are VP's of Ti,Tg and Tn.

State 4 : [Ti [Vi to_TO give_VB Vi] [N fahrenheit_NN the_ATI knock_NN N] Ti]
Description : to-infinitive clause.

State 5 : [V is_BEZ V]
Description : A finite `verb phrase' i.e. one that excludes objects, complements, etc.

State 6 : [N purpose_NN N]
Description : Label for a noun phrase, whether it is a single word or a sequence of words.

B04 : 249

Desired parse : [S[Na he_PP3A Na][V says_VBZ V][Fn that_CS [N the_ATI Scots_NNPS N][V are_BER V][N foreigners_NNS [Fr[Nq who_WP Nq][V have_HV V][N no_ATI business_NN N][Ti[Vi to_TO be_BE Vi][P in_IN [N England_NP N]P]Ti]Fr]N]Fn] ._. S]
Desired bracketing : (S(Na)(V)(Fn(N)(V)(N(Fr(Nq)(V)(N)(Ti(Vi)(P(N)))))))

Concords attempt ....

State 1 : [N England_NP N]
Description : Label for a noun phrase, whether it is a single word or a sequence of words.

State 2 : [P in_IN [N England_NP N] P]
Description : A prepositional phrase, e.g. `in London' or `on arriving at the station'.

State 3 : [Vi to_TO be_BE Vi]
Description : Label for nonfinite verb phrases(VP). i.e VP's that are VP's of Ti,Tg and Tn.

State 4 : [N no_ATI business_NN [Vi to_TO be_BE Vi] [P in_IN [N England_NP N] P] N]
Description : Label for a noun phrase, whether it is a single word or a sequence of words.

State 5 : [V have_HV V]
Description : A finite `verb phrase' i.e. one that excludes objects, complements, etc.

State 6 : [Na who_WP Na]
Description : A noun phrase marked as subject of the verb.

State 7 : [N foreigners_NNS N]
Description : Label for a noun phrase, whether it is a single word or a sequence of words.

State 8 : [Fn [N foreigners_NNS N] [Na who_WP Na] [V have_HV V] [N no_ATI business_NN [Vi to_TO be_BE Vi] [P in_IN [N England_NP N] P] N] Fn]
Description : finite nominal clause (subordinate clause functioning in the pos of a Noun PH

State 9 : [V are_BER V]
Description : A finite `verb phrase' i.e. one that excludes objects, complements, etc.

State 10 : [N the_ATI Scots_NNPS N]
Description : Label for a noun phrase, whether it is a single word or a sequence of words.

State 11 : [Fn that_CS [N the_ATI Scots_NNPS N] [V are_BER V] [Fn [N foreigners_NNS N] [Na who_WP Na] [V have_HV V] [N no_ATI business_NN [Vi to_TO be_BE Vi] [P in_IN [N England_NP N] P] N] Fn] Fn]
Description : finite nominal clause (subordinate clause functioning in the pos of a Noun PH

State 12 : [V says_VBZ V]
Description : A finite `verb phrase' i.e. one that excludes objects, complements, etc.

State 13 : [Na he_PP3A Na]
Description : A noun phrase marked as subject of the verb.

B07 : 545
Desired parse : [S[E there_EX E][V are_BER V][N two_CD basic_JJ points_NNS [Fr[Nq which_WDT Nq][V seem_VB V][Ti[Vi to_TO be_BE Vi][N a_AT necessary_JJ preface_NN [P to_IN [N any_DTI sensible_JJ discussion_NN [Po of_INO [N taxation_NN reform_NN N]Po]N]P]N]Ti]Fr]N] ._. S]
Desired bracketing : (S(E)(V)(N(Fr(Nq)(V)(Ti(Vi)(N(P(N(Po(N))))))))

Concords attempt ....

State 1 : [N taxation_NN reform_NN N]
Description : Label for a noun phrase, whether it is a single word or a sequence of words.

State 2 : [Po of_INO [N taxation_NN reform_NN N] Po]
Description : A prepositional phrase beginning with the preposition 'of'.

State 3 : [Vn any_DTI sensible_JJ discussion_NN [Po of_INO [N taxation_NN reform_NN N] Po] Vn]
Description : Label for nonfinite verb phrases(VP). i.e VP's that are VP's of Ti,Tg and Tn.

State 4 : [P to_IN [Vn any_DTI sensible_JJ discussion_NN [Po of_INO [N taxation_NN reform_NN N] Po] Vn] P]
Description : A prepositional phrase, e.g. `in London' or `on arriving at the station'.

State 5 : [N a_AT necessary_JJ preface_NN [P to_IN [Vn any_DTI sensible_JJ discussion_NN [Po of_INO [N taxation_NN reform_NN N] Po] Vn] P] N]
Description : Label for a noun phrase, whether it is a single word or a sequence of words.

State 6 : [Rq be_BE Rq]
Description : an adverb phrase beginning with a wh-word, e.g. `How do you feel?',or `how long'

State 7 : [Vi to_TO [Rq be_BE Rq] Vi]
Description : Label for nonfinite verb phrases(VP). i.e VP's that are VP's of Ti,Tg and Tn.

State 8 : [Ti [Vi to_TO [Rq be_BE Rq] Vi] [N a_AT necessary_JJ preface_NN [P to_IN [Vn any_DTI sensible_JJ discussion_NN [Po of_INO [N taxation_NN reform_NN N] Po] Vn] P] N] Ti]
Description : to-infinitive clause.

State 9 : [Nq which_WDT seem_VB Nq]
Description : A wh- noun phrase, such as `who', `which', `which car', `what time' etc.

State 10 : [N basic_JJ points_NNS N]
Description : Label for a noun phrase, whether it is a single word or a sequence of words.

State 11 : [Na two_CD [N basic_JJ points_NNS N] Na]
Description : A noun phrase marked as subject of the verb.

State 12 : [V are_BER V]
Description : A finite `verb phrase' i.e. one that excludes objects, complements, etc.

C04 : 220
Desired parse : [S[Na they_PP3AS Na][V will_MD be_BE asked_VBN V][Ti[Vi to_TO comment_VB Vi][P on_IN [N the_ATI design_NN [Po of_INO [N everyday_JJB articles_NNS [P[IN= such_IN21 as_IN22 IN=][N& a_AT chair_NN [N+ and_CC a_AT motor-car_NN N+]N&]P]N]Po]N]P]Ti] ._. S]
Desired bracketing : (S(Na)(V)(Ti(Vi)(P(N(Po(N(P(IN=)(N&(N+)))))))))

Concords attempt ....

State 1 : [N+ and_CC a_AT motorcar_NN N+]
Description : Second or subsequent conjoin of a compound Label for a noun phrase, whether it is a single word or a sequence of words.

State 2 : [N a_AT chair_NN [N+ and_CC a_AT motorcar_NN N+] N]
Description : Label for a noun phrase, whether it is a single word or a sequence of words.

State 3 : [P as_IN22 [N a_AT chair_NN [N+ and_CC a_AT motorcar_NN N+] N] P]
Description : A prepositional phrase, e.g. `in London' or `on arriving at the station'.

State 4 : [Po such_IN21 [P as_IN22 [N a_AT chair_NN [N+ and_CC a_AT motorcar_NN N+] N] P] Po]
Description : A prepositional phrase beginning with the preposition 'of'.

State 5 : [N everyday_JJB articles_NNS [Po such_IN21 [P as_IN22 [N a_AT chair_NN [N+ and_CC a_AT motorcar_NN N+] N] P] Po] N]
Description : Label for a noun phrase, whether it is a single word or a sequence of words.

State 6 : [Po of_INO [N everyday_JJB articles_NNS [Po such_IN21 [P as_IN22 [N a_AT chair_NN [N+ and_CC a_AT motorcar_NN N+] N] P] Po] N] Po]

Description : A prepositional phrase beginning with the preposition 'of'.

State 7 : [N the_ATI design_NN [Po of_INO [N everyday_JJB articles_NNS [Po such_IN21 [P as_IN22 [N a_AT chair_NN [N+ and_CC a_AT motorcar_NN N+] N] P] Po] N] Po] N]
Description : Label for a noun phrase, whether it is a single word or a sequence of words.

State 8 : [P on_IN [N the_ATI design_NN [Po of_INO [N everyday_JJB articles_NNS [Po such_IN21 [P as_IN22 [N a_AT chair_NN [N+ and_CC a_AT motorcar_NN N+] N] P] Po] N] Po] N] P]
Description : A prepositional phrase, e.g. `in London' or `on arriving at the station'.

State 9 : [Vi to_TO comment_VB Vi]
Description : Label for nonfinite verb phrases(VP). i.e VP's that are VP's of Ti,Tg and Tn.

State 10 : [V be_BE asked_VBN V]
Description : A finite `verb phrase' i.e. one that excludes objects, complements, etc.

State 11 : [V will_MD V]
Description : A finite `verb phrase' i.e. one that excludes objects, complements, etc.

State 12 : [S [V will_MD V] [V be_BE asked_VBN V] [Vi to_TO comment_VB Vi] [P on_IN [N the_ATI design_NN [Po of_INO [N everyday_JJB articles_NNS [Po such_IN21 [P as_IN22 [N a_AT chair_NN [N+ and_CC a_AT motorcar_NN N+] N] P] Po] N] Po] N] P] S]
Description : sentence.

State 13 : [Jq they_PP3AS Jq]
Description : A phrase beginning with a wh-word e.g. `How old'.

C06 : 316

Desired parse : [S[Nq what_WDT Nq][V happened_VBD V][P to_IN [N the_ATI new_JJ book_NN [Tn[R partially_RB R][Vn tape-recorded_VBN Vn][P by_IN [N his_PP$ publishers_NNS N]P][P in_IN [N March_NR [Po of_INO [N last_AP year_NN N]Po]N]P]Tn]N]P] ?_? S]
Desired bracketing : (S(Nq)(V)(P(N(Tn(R)(Vn)(P(N))(P(N(Po(N)))))))

Concords attempt ....

State 1 : [N last_AP year_NN N]
Description : Label for a noun phrase, whether it is a single word or a sequence of words.

State 2 : [Po of_INO [N last_AP year_NN N] Po]
Description : A prepositional phrase beginning with the preposition 'of'.

State 3 : [N March_NR [Po of_INO [N last_AP year_NN N] Po] N]
Description : Label for a noun phrase, whether it is a single word or a sequence of words.

State 4 : [P in_IN [N March_NR [Po of_INO [N last_AP year_NN N] Po] N] P]
Description : A prepositional phrase, e.g. `in London' or `on arriving at the station'.

State 5 : [N his_PP$ publishers_NNS [P in_IN [N March_NR [Po of_INO [N last_AP year_NN N] Po] N] P] N]
Description : Label for a noun phrase, whether it is a single word or a sequence of words.

State 6 : [P by_IN [N his_PP$ publishers_NNS [P in_IN [N March_NR [Po of_INO [N last_AP year_NN N] Po] N] P] N] P]
Description : A prepositional phrase, e.g. `in London' or `on arriving at the station'.

State 7 : [Vn taperecorded_VBN Vn]
Description : Label for nonfinite verb phrases(VP). i.e VP's that are VP's of Ti,Tg and Tn.

State 8 : [R partially_RB R]
Description : An adverb phrase, e.g. `there',`quickly' or a sequence such as `quite often' etc

State 9 : [N book_NN N]

Description : Label for a noun phrase, whether it is a single word or a sequence of words.

C09 : 540

Desired parse : [S[L&[N bad_JJ characters_NNS N][J good_JJ J] ,_, [L-[N good_JJ characters_NNS N][J bad_JJ J]L-]L&] ._. S]

Desired bracketing : (S(L&(N)(J)(L-(N)(J))))

Concords attempt ....

D03 : 186

Desired parse : [S[Ti[Vi to_TO cheer_VB Vi][N my_PP$ body_NN N][P with_IN [N wine_NN N]P]Ti] ._. S]

Desired bracketing : (S(Ti(Vi)(N)(P(N))))

Concords attempt ....

State 1 : [N wine_NN N]

Description : Label for a noun phrase, whether it is a single word or a sequence of words.

State 2 : [P with_IN [N wine_NN N] P]

Description : A prepositional phrase, e.g. `in London' or `on arriving at the station'.

State 3 : [N my_PP$ body_NN N]

Description : Label for a noun phrase, whether it is a single word or a sequence of words.

State 4 : [Vi to_TO cheer_VB Vi]

Description : Label for nonfinite verb phrases(VP). i.e VP's that are VP's of Ti,Tg and Tn.

State 5 : [Ti [Vi to_TO cheer_VB Vi] [N my_PP$ body_NN N] [P with_IN [N wine_NN N] P] Ti]

Description : to-infinitive clause.

E09 : 489

Desired parse : [S[N Dog_NP River_NPL *-_*- [N Nahr_&FW el-Kelb_&FW N]N] !_! S]

Desired bracketing : (S(N(N)))

Concords attempt ....

F01 : 70

Desired parse : [S but_CC [Na they_PP3AS Na][V could_MD [N both_ABX N] see_VB V][N the_ATI cloud_NN N] ._. S]

Desired bracketing : (S(Na)(V(N))(N))

Concords attempt ....

State 1 : [N the_ATI cloud_NN N]

Description : Label for a noun phrase, whether it is a single word or a sequence of words.

State 2 : [V see_VB V]

Description : A finite `verb phrase' i.e. one that excludes objects, complements, etc.

State 3 : [V could_MD both_ABX V]

Description : A finite `verb phrase' i.e. one that excludes objects, complements, etc.

State 4 : [Na they_PP3AS Na]

Description : A noun phrase marked as subject of the verb.

State 5 : [S [Na they_PP3AS Na] [V could_MD both_ABX V] [V see_VB V] [N the_ATI cloud_NN N] S]

Description : sentence.

State 6 : [S+ but_CC [S [Na they_PP3AS Na] [V could_MD both_ABX V] [V see_VB V] [N the_ATI cloud_NN N] S] S+]

Description : Second or subsequent conjoin of a compound sentence.

Appendix K

A Sample of Matching Parses for The Constrained Test Sample

liquid_JJ glass_NN is_BEZ in_IN the_ATI form_NN of_INO small_JJ crystals_NNS
Desired parse : [S[N liquid_JJ glass_NN N][V is_BEZ V][P in_IN [N the_ATI form_NN [Po of_INO [N small_JJ crystals_NNS N]Po]N]P] ._. S]
Actual parse : [S [N liquid_JJ glass_NN N] [V is_BEZ V] [P in_IN [N the_ATI form_NN [Po of_INO [N small_JJ crystals_NNS N] Po] N] P] S]
Desired bracketing : (S(N)(V)(P(N(Po(N)))))
Actual bracketing : (S(N)(V)(P(N(Po(N)))))

where_WRB was_BEDZ it_PP3
Desired parse : [S[Rq where_WRB Rq][V was_BEDZ V][N it_PP3 N] ?_? S]
Actual parse : [S [Rq where_WRB Rq] [V was_BEDZ V] [N it_PP3 N] S]
Desired bracketing : (S(Rq)(V)(N))
Actual bracketing : (S(Rq)(V)(N))

Frank_NP stopped_VBD the_ATI machine_NN and_CC stood_VBD taut_JJ
Desired parse : [S&[N Frank_NP N][V stopped_VBD V][N the_ATI machine_NN N][S+ and_CC [V stood_VBD V][J taut_JJ J]S+] ._. S&]
Actual parse : [S& [N Frank_NP N] [V stopped_VBD V] [N the_ATI machine_NN N] [S+ and_CC [V stood_VBD V] [J taut_JJ J] S+] S&]
Desired bracketing : (S&(N)(V)(N)(S+(V)(J)))
Actual bracketing : (S&(N)(V)(N)(S+(V)(J)))

Percy_NP tried_VBD to_TO hold_VB him_PP3O back_RP
Desired parse : [S[N Percy_NP N][V tried_VBD V][Ti[Vi to_TO hold_VB Vi][N him_PP3O N][R back_RP R]Ti] ._. S]
Actual parse : [S [N Percy_NP N] [V tried_VBD V] [Ti [Vi to_TO hold_VB Vi] [N him_PP3O N] [R back_RP R] Ti] S]
Desired bracketing : (S(N)(V)(Ti(Vi)(N)(R)))
Actual bracketing : (S(N)(V)(Ti(Vi)(N)(R)))

I_PP1A just_RB thought_VBD
Desired parse : [S[Na I_PP1A Na][R just_RB R][V thought_VBD V] ._. S]
Actual parse : [S [Na I_PP1A Na] [R just_RB R] [V thought_VBD V] S]
Desired bracketing : (S(Na)(R)(V))
Actual bracketing : (S(Na)(R)(V))

say_VB something_PN to_IN me_PP1O
Desired parse : [S[V say_VB V][N something_PN N][P to_IN [N me_PP1O N]P] ._. S] **'_**'
Actual parse : [S [V say_VB V] [N something_PN N] [P to_IN [N me_PP1O N] P] S]
Desired bracketing : (S(V)(N)(P(N)))
Actual bracketing : (S(V)(N)(P(N)))

the_ATI verger_NN was_BEDZ simple_JJ in_IN his_PP$ nature_NN
Desired parse : [S[N the_ATI verger_NN N][V was_BEDZ V][J simple_JJ [P in_IN [N his_PP$ nature_NN N]P]J] ._. S]
Actual parse : [S [N the_ATI verger_NN N] [V was_BEDZ V] [J simple_JJ [P in_IN [N his_PP$ nature_NN N] P] J] S]
Desired bracketing : (S(N)(V)(J(P(N))))
Actual bracketing : (S(N)(V)(J(P(N))))

I_PP1A will_MD nt_XNOT drink_VB it_PP3
Desired parse : *'_*' [S[Na I_PP1A Na][V will_MD n't_XNOT drink_VB V][N it_PP3 N] ._. S] **'_**'
Actual parse : [S [Na I_PP1A Na] [V will_MD nt_XNOT drink_VB V] [N it_PP3 N] S]
Desired bracketing : (S(Na)(V)(N))

Actual bracketing : (S(Na)(V)(N))

he_PP3A gave_VBD a_AT low_JJ laugh_NN
Desired parse : [S[Na he_PP3A Na][V gave_VBD V][N a_AT low_JJ laugh_NN N] ._. S]
Actual parse : [S [Na he_PP3A Na] [V gave_VBD V] [N a_AT low_JJ laugh_NN N] S]
Desired bracketing : (S(Na)(V)(N))
Actual bracketing : (S(Na)(V)(N))

Grant_NP was_BEDZ driving_VBG the_ATI car_NN carefully_RB
Desired parse : [S[N Grant_NP N][V was_BEDZ driving_VBG V][N the_ATI car_NN N][R carefully_RB R] ._. S]
Actual parse : [S [N Grant_NP N] [V was_BEDZ driving_VBG V] [N the_ATI car_NN N] [R carefully_RB R] S]
Desired bracketing : (S(N)(V)(N)(R))
Actual bracketing : (S(N)(V)(N)(R))

he_PP3A came_VBD round_RP and_CC opened_VBD the_ATI back_JJB door_NN of_INO the_ATI car_NN
Desired parse : [S&[Na he_PP3A Na][V came_VBD V][R round_RP R][S+ and_CC [V opened_VBD V][N the_ATI back_JJB door_NN [Po of_INO [N the_ATI car_NN N]Po]N]S+] ._. S&]
Actual parse : [S& [Na he_PP3A Na] [V came_VBD V] [R round_RP R] [S+ and_CC [V opened_VBD V] [N the_ATI back_JJB door_NN [Po of_INO [N the_ATI car_NN N] Po] N] S+] S&]
Desired bracketing : (S&(Na)(V)(R)(S+(V)(N(Po(N)))))
Actual bracketing : (S&(Na)(V)(R)(S+(V)(N(Po(N)))))

I_PP1A have_HV just_RB heard_VBN a_AT good_JJ example_NN of_INO their_PP$ officious_JJ efficiency_NN
Desired parse : [S[Na I_PP1A Na][V have_HV just_RB heard_VBN V][N a_AT good_JJ example_NN [Po of_INO [N their_PP$ officious_JJ efficiency_NN N]Po]N] ._. S]
Actual parse : [S [Na I_PP1A Na] [V have_HV just_RB heard_VBN V] [N a_AT good_JJ example_NN [Po of_INO [N their_PP$ officious_JJ efficiency_NN N] Po] N] S]
Desired bracketing : (S(Na)(V)(N(Po(N))))
Actual bracketing : (S(Na)(V)(N(Po(N))))

the_ATI earth_NN is_BEZ crammed_VBN with_IN teeming_JJ multiplying_JJ humanity_NN
Desired parse : [S[N the_ATI earth_NN N][V is_BEZ crammed_VBN V][P with_IN [N teeming_JJ ,_, multiplying_JJ humanity_NN N]P] ._. S]
Actual parse : [S [N the_ATI earth_NN N] [V is_BEZ crammed_VBN V] [P with_IN [N teeming_JJ multiplying_JJ humanity_NN N] P] S]
Desired bracketing : (S(N)(V)(P(N)))
Actual bracketing : (S(N)(V)(P(N)))

once_RB he_PP3A was_BEDZ a_AT nonstarter_NN
Desired parse : [S[R once_RB R][Na he_PP3A Na][V was_BEDZ V][N a_AT non-starter_NN N] ._. S]
Actual parse : [S [R once_RB R] [Na he_PP3A Na] [V was_BEDZ V] [N a_AT nonstarter_NN N] S]
Desired bracketing : (S(R)(Na)(V)(N))
Actual bracketing : (S(R)(Na)(V)(N))

maybe_RB a_AT scheme_NN is_BEZ coming_VBG to_IN fruition_NN
Desired parse : [S[R maybe_RB R][N a_AT scheme_NN N][V is_BEZ coming_VBG V][P to_IN [N fruition_NN N]P] ._. S]
Actual parse : [S [R maybe_RB R] [N a_AT scheme_NN N] [V is_BEZ coming_VBG V] [P to_IN [N fruition_NN N] P] S]
Desired bracketing : (S(R)(N)(V)(P(N)))
Actual bracketing : (S(R)(N)(V)(P(N)))

there_EX s_BEZ another_DT idea_NN for_IN the_ATI wastepaper_JJB basket_NN
Desired parse : [S[E there_EX E][V 's_BEZ V][N another_DT idea_NN [P for_IN [N the_ATI waste-paper_JJB basket_NN N]P]N] ._. S]
Actual parse : [S [E there_EX E] [V s_BEZ V] [N another_DT idea_NN [P for_IN [N the_ATI wastepaper_JJB basket_NN N] P] N] S]
Desired bracketing : (S(E)(V)(N(P(N))))
Actual bracketing : (S(E)(V)(N(P(N))))

it_PP3 is_BEZ something_PN to_TO do_DO with_IN that_DT oil_NN vapour_NN
Desired parse : [S[N it_PP3 N][V is_BEZ V][N something_PN [Ti[Vi to_TO do_DO Vi][P with_IN [N that_DT oil_NN vapour_NN N]P]Ti]N] ._. S]

Actual parse : [S [N it_PP3 N] [V is_BEZ V] [N something_PN [Ti [Vi to_TO do_DO Vi] [P with_IN [N that_DT oil_NN vapour_NN N] P] Ti] N] S]
Desired bracketing : (S(N)(V)(N(Ti(Vi)(P(N)))))
Actual bracketing : (S(N)(V)(N(Ti(Vi)(P(N)))))

it_PP3 was_BEDZ a_AT cosmopolitan_JJ gathering_NN of_INO stars_NNS
Desired parse : [S[N it_PP3 N][V was_BEDZ V][N a_AT cosmopolitan_JJ gathering_NN [Po of_INO [N stars_NNS N]Po]N] ._. S]
Actual parse : [S [N it_PP3 N] [V was_BEDZ V] [N a_AT cosmopolitan_JJ gathering_NN [Po of_INO [N stars_NNS N] Po] N] S]
Desired bracketing : (S(N)(V)(N(Po(N))))
Actual bracketing : (S(N)(V)(N(Po(N))))

the_ATI relationship_NN began_VBD in_IN the_ATI primary_JJ School_NPL
Desired parse : [S[N the_ATI relationship_NN N][V began_VBD V][P in_IN [N the_ATI primary_JJ School_NPL N]P] ._. S]
Actual parse : [S [N the_ATI relationship_NN N] [V began_VBD V] [P in_IN [N the_ATI primary_JJ School_NPL N] P] S]
Desired bracketing : (S(N)(V)(P(N)))
Actual bracketing : (S(N)(V)(P(N)))

we_PP1AS plan_VB to_TO attack_VB the_ATI heavy_JJ world_NN
Desired parse : [S[Na we_PP1AS Na][V plan_VB V][Ti[Vi to_TO attack_VB Vi][N the_ATI heavy_JJ world_NN N]Ti] ._. S]
Actual parse : [S [Na we_PP1AS Na] [V plan_VB V] [Ti [Vi to_TO attack_VB Vi] [N the_ATI heavy_JJ world_NN N] Ti] S]
Desired bracketing : (S(Na)(V)(Ti(Vi)(N)))
Actual bracketing : (S(Na)(V)(Ti(Vi)(N)))

it_PP3 was_BEDZ done_VBN and_CC nothing_PN could_MD ever_RB change_VB it_PP3
Desired parse : [S&[N it_PP3 N][V was_BEDZ done_VBN V] ,_, [S+ and_CC [N nothing_PN N][V could_MD ever_RB change_VB V][N it_PP3 N]S+] ._. S&]
Actual parse : [S& [N it_PP3 N] [V was_BEDZ done_VBN V] [S+ and_CC [N nothing_PN N] [V could_MD ever_RB change_VB V] [N it_PP3 N] S+] S&]
Desired bracketing : (S&(N)(V)(S+(N)(V)(N)))
Actual bracketing : (S&(N)(V)(S+(N)(V)(N)))

the_ATI saying_NN of_INO R_NPT Joshua_NP b_&FW Levi_NP
Desired parse : [S[N the_ATI saying_NN [Po of_INO [N \0R_NPT Joshua_NP b_&FW Levi_NP N]Po]N] ._. S]
Actual parse : [S [N the_ATI saying_NN [Po of_INO [N R_NPT Joshua_NP b_&FW Levi_NP N] Po] N] S]
Desired bracketing : (S(N(Po(N))))
Actual bracketing : (S(N(Po(N))))

I_PP1A ve_HV enjoyed_VBN Pepita_NP so_QL much_RB
Desired parse : *'_*' [S[Na I_PP1A Na][V 've_HV enjoyed_VBN V][N Pepita_NP N][R so_QL much_RB R] ._. S] **'_**'
Actual parse : [S [Na I_PP1A Na] [V ve_HV enjoyed_VBN V] [N Pepita_NP so_QL N] [R much_RB R] S]
Desired bracketing : (S(Na)(V)(N)(R))
Actual bracketing : (S(Na)(V)(N)(R))

I_PP1A can_MD nt_XNOT bear_VB to_TO see_VB her_PP3O like_IN this_DT
Desired parse : [S[Na I_PP1A Na][V can_MD n't_XNOT bear_VB V][Ti[Vi to_TO see_VB Vi][N her_PP3O N][P like_IN [N this_DT N]P]Ti] ._. S]
Actual parse : [S [Na I_PP1A Na] [V can_MD nt_XNOT bear_VB V] [Ti [Vi to_TO see_VB Vi] [N her_PP3O N] [P like_IN [N this_DT N] P] Ti] S]
Desired bracketing : (S(Na)(V)(Ti(Vi)(N)(P(N))))
Actual bracketing : (S(Na)(V)(Ti(Vi)(N)(P(N))))

this_DT is_BEZ the_ATI pardus_&FW
Desired parse : [S (_([N this_DT N][V is_BEZ V][N the_ATI pardus_&FW N] ._.)_) S]
Actual parse : [S [N this_DT N] [V is_BEZ V] [N the_ATI pardus_&FW N] S]
Desired bracketing : (S(N)(V)(N))
Actual bracketing : (S(N)(V)(N))

0Mr_NPT Brown_NP went_VBD on_RP
Desired parse : [S[N \0Mr_NPT Brown_NP N][V went_VBD V][R on_RP R] :_: S]
Actual parse : [S [N 0Mr_NPT Brown_NP N] [V went_VBD V] [R on_RP R] S]
Desired bracketing : (S(N)(V)(R))
Actual bracketing : (S(N)(V)(R))

by_IN John_NP Dickie_NP
Desired parse : [S[P by_IN [N John_NP Dickie_NP N]P] ._. S]
Actual parse : [S [P by_IN [N John_NP Dickie_NP N] P] S]
Desired bracketing : (S(P(N)))
Actual bracketing : (S(P(N)))

Sir_NPT Roy_NP attacks_VBZ Kaundas_NP$ vicious_JJ monster_NN
Desired parse : [S[N Sir_NPT Roy_NP N][V attacks_VBZ V][N Kaunda's_NP$ *'_*' vicious_JJ monster_NN N] **'_**' ._. S]
Actual parse : [S [N Sir_NPT Roy_NP N] [V attacks_VBZ V] [N Kaundas_NP$ vicious_JJ monster_NN N] S]
Desired bracketing : (S(N)(V)(N))
Actual bracketing : (S(N)(V)(N))

recruiting_NN confidence_NN
Desired parse : [S[N recruiting_NN confidence_NN N] ._. S]
Actual parse : [S [N recruiting_NN confidence_NN N] S]
Desired bracketing : (S(N))
Actual bracketing : (S(N))

but_CC I_PP1A am_BEM not_XNOT going_VBG to_TO waste_VB time_NN
Desired parse : *'_*' [S but_CC [Na I_PP1A Na][V am_BEM not_XNOT going_VBG V][Ti[Vi to_TO waste_VB Vi][N time_NN N]Ti] ._. S]
Actual parse : [S but_CC [Na I_PP1A Na] [V am_BEM not_XNOT going_VBG V] [Ti [Vi to_TO waste_VB Vi] [N time_NN N] Ti] S]
Desired bracketing : (S(Na)(V)(Ti(Vi)(N)))
Actual bracketing : (S(Na)(V)(Ti(Vi)(N)))

Bagenal_NP Harvey_NP is_BEZ the_ATI name_NN
Desired parse : [S[N Bagenal_NP Harvey_NP N][V is_BEZ V][N the_ATI name_NN N] ._. S]
Actual parse : [S [N Bagenal_NP Harvey_NP N] [V is_BEZ V] [N the_ATI name_NN N] S]
Desired bracketing : (S(N)(V)(N))
Actual bracketing : (S(N)(V)(N))

jockey_NN judge_NN will_MD ride_VB on_IN circuit_NN
Desired parse : [S[N jockey_NN judge_NN N][V will_MD ride_VB V][P on_IN [N circuit_NN N]P] ._. S]
Actual parse : [S [N jockey_NN judge_NN N] [V will_MD ride_VB V] [P on_IN [N circuit_NN N] P] S]
Desired bracketing : (S(N)(V)(P(N)))
Actual bracketing : (S(N)(V)(P(N)))

he_PP3A will_MD be_BE racing_VBG against_IN five_CD barristers_NNS
Desired parse : [S[Na he_PP3A Na][V will_MD be_BE racing_VBG V][P against_IN [N five_CD barristers_NNS N]P] ._. S]
Actual parse : [S [Na he_PP3A Na] [V will_MD be_BE racing_VBG V] [P against_IN [N five_CD barristers_NNS N] P] S]
Desired bracketing : (S(Na)(V)(P(N)))
Actual bracketing : (S(Na)(V)(P(N)))

and_CC I_PP1A like_VB a_AT challenge_NN
Desired parse : [S and_CC [Na I_PP1A Na][V like_VB V][N a_AT challenge_NN N] ._. S]
Actual parse : [S and_CC [Na I_PP1A Na] [V like_VB V] [N a_AT challenge_NN N] S]
Desired bracketing : (S(Na)(V)(N))
Actual bracketing : (S(Na)(V)(N))

nevertheless_RB the_ATI tour_NN has_HVZ been_BEN an_AT immense_JJ success_NN
Desired parse : [S[R nevertheless_RB R] ,_, [N the_ATI tour_NN N][V has_HVZ been_BEN V][N an_AT immense_JJ success_NN N] ._. S]

Actual parse : [S [R nevertheless_RB R] [N the_ATI tour_NN N] [V has_HVZ been_BEN V] [N an_AT immense_JJ success_NN N] S]
Desired bracketing : (S(R)(N)(V)(N))
Actual bracketing : (S(R)(N)(V)(N))

certainly_RB the_ATI rise_NN is_BEZ very_QL small_JJ
Desired parse : [S[R certainly_RB R] ,_, [N the_ATI rise_NN N][V is_BEZ V][J very_QL small_JJ J] ._. S]
Actual parse : [S [R certainly_RB R] [N the_ATI rise_NN N] [V is_BEZ V] [J very_QL small_JJ J] S]
Desired bracketing : (S(R)(N)(V)(J))
Actual bracketing : (S(R)(N)(V)(J))

it_PP3 has_HVZ its_PP$ complications_NNS
Desired parse : [S[N it_PP3 N][V has_HVZ V][N its_PP$ complications_NNS N] ._. S]
Actual parse : [S [N it_PP3 N] [V has_HVZ V] [N its_PP$ complications_NNS N] S]
Desired bracketing : (S(N)(V)(N))
Actual bracketing : (S(N)(V)(N))

they_PP3AS are_BER backed_VBN by_IN a_AT resurgent_JJ film_NN industry_NN
Desired parse : [S[Na they_PP3AS Na][V are_BER backed_VBN V][P by_IN [N a_AT resurgent_JJ film_NN industry_NN N]P] ._. S]
Actual parse : [S [Na they_PP3AS Na] [V are_BER backed_VBN V] [P by_IN [N a_AT resurgent_JJ film_NN industry_NN N] P] S]
Desired bracketing : (S(Na)(V)(P(N)))
Actual bracketing : (S(Na)(V)(P(N)))

here_RN are_BER the_ATI new_JJ pioneers_NNS
Desired parse : [S[R here_RN R][V are_BER V][N the_ATI new_JJ pioneers_NNS N] ._. S]
Actual parse : [S [R here_RN R] [V are_BER V] [N the_ATI new_JJ pioneers_NNS N] S]
Desired bracketing : (S(R)(V)(N))
Actual bracketing : (S(R)(V)(N))

afterwards_RB they_PP3AS said_VBD
Desired parse : [S[R afterwards_RB R][Na they_PP3AS Na][V said_VBD V] :_: S]
Actual parse : [S [R afterwards_RB R] [Na they_PP3AS Na] [V said_VBD V] S]
Desired bracketing : (S(R)(Na)(V))
Actual bracketing : (S(R)(Na)(V))

he_PP3A means_VBZ it_PP3
Desired parse : [S[Na he_PP3A Na][V means_VBZ V][N it_PP3 N] ._. S]
Actual parse : [S [Na he_PP3A Na] [V means_VBZ V] [N it_PP3 N] S]
Desired bracketing : (S(Na)(V)(N))
Actual bracketing : (S(Na)(V)(N))

polarisation_NN of_INO the_ATI labour_NN party_NN
Desired parse : [S[N polarisation_NN [Po of_INO [N the_ATI labour_NN party_NN N]Po]N] ._. S]
Actual parse : [S [N polarisation_NN [Po of_INO [N the_ATI labour_NN party_NN N] Po] N] S]
Desired bracketing : (S(N(Po(N))))
Actual bracketing : (S(N(Po(N))))

it_PP3 was_BEDZ not_XNOT enough_DTI
Desired parse : [S[N it_PP3 N][V was_BEDZ not_XNOT V][N enough_DTI N] ._. S]
Actual parse : [S [N it_PP3 N] [V was_BEDZ not_XNOT V] [N enough_DTI N] S]
Desired bracketing : (S(N)(V)(N))
Actual bracketing : (S(N)(V)(N))

age_NN of_INO the_ATI coloured_JJ aquatint_NN
Desired parse : [S[N age_NN [Po of_INO [N the_ATI coloured_JJ aquatint_NN N]Po]N] ._. S]
Actual parse : [S [N age_NN [Po of_INO [N the_ATI coloured_JJ aquatint_NN N] Po] N] S]
Desired bracketing : (S(N(Po(N))))
Actual bracketing : (S(N(Po(N))))

do_DO nt_XNOT tell_VB me_PP1O
Desired parse : [S[V do_DO n't_XNOT tell_VB V][N me_PP1O N] ._. S]
Actual parse : [S [V do_DO nt_XNOT tell_VB V] [N me_PP1O N] S]
Desired bracketing : (S(V)(N))
Actual bracketing : (S(V)(N))

Rodins_NP$ ghost_NN will_MD not_XNOT be_BE laid_VBN
Desired parse : [S[N Rodin's_NP$ ghost_NN N][V will_MD not_XNOT be_BE laid_VBN V] ._. S]
Actual parse : [S [N Rodins_NP$ ghost_NN N] [V will_MD not_XNOT be_BE laid_VBN V] S]
Desired bracketing : (S(N)(V))
Actual bracketing : (S(N)(V))

aplomb_NN is_BEZ a_AT cooling_JJ quality_NN
Desired parse : [S[N aplomb_NN N][V is_BEZ V][N a_AT cooling_JJ quality_NN N] ._. S]
Actual parse : [S [N aplomb_NN N] [V is_BEZ V] [N a_AT cooling_JJ quality_NN N] S]
Desired bracketing : (S(N)(V)(N))
Actual bracketing : (S(N)(V)(N))

the_ATI reasons_NNS are_BER quite_RB simple_JJ
Desired parse : [S[N the_ATI reasons_NNS N][V are_BER V][J quite_RB simple_JJ J] ._. S]
Actual parse : [S [N the_ATI reasons_NNS N] [V are_BER V] [J quite_RB simple_JJ J] S]
Desired bracketing : (S(N)(V)(J))
Actual bracketing : (S(N)(V)(J))

this_DT is_BEZ a_AT key_NN for_IN human_JJ sin_NN
Desired parse : [S[N this_DT N][V is_BEZ V][N a_AT key_NN [P for_IN [N human_JJ sin_NN N]P]N] ._. S]
Actual parse : [S [N this_DT N] [V is_BEZ V] [N a_AT key_NN [P for_IN [N human_JJ sin_NN N] P] N] S]
Desired bracketing : (S(N)(V)(N(P(N))))
Actual bracketing : (S(N)(V)(N(P(N))))

there_EX is_BEZ no_ATI necessary_JJ suggestion_NN of_INO evil_JJ desire_NN
Desired parse : [S[E there_EX E][V is_BEZ V][N no_ATI necessary_JJ suggestion_NN [Po of_INO [N evil_JJ desire_NN N]Po]N] ._. S]
Actual parse : [S [E there_EX E] [V is_BEZ V] [N no_ATI necessary_JJ suggestion_NN [Po of_INO [N evil_JJ desire_NN N] Po] N] S]
Desired bracketing : (S(E)(V)(N(Po(N))))
Actual bracketing : (S(E)(V)(N(Po(N))))

they_PP3AS will_MD not_XNOT work_VB in_IN union_NN
Desired parse : [S[Na they_PP3AS Na][V will_MD not_XNOT work_VB V][P in_IN [N union_NN N]P] ._. S]
Actual parse : [S [Na they_PP3AS Na] [V will_MD not_XNOT work_VB V] [P in_IN [N union_NN N] P] S]
Desired bracketing : (S(Na)(V)(P(N)))
Actual bracketing : (S(Na)(V)(P(N)))

Appendix L

A Sample of Parse Mismatches for The Constrained Test Sample

obtainable_JJ in_IN the_ATI following_JJ shades_NNS X_&FW
Desired parse : [S[J obtainable_JJ J][P in_IN [N the_ATI following_JJ shades_NNS N]P] :_: [N X_&FW N] ._. S]
Actual parse : [S [J obtainable_JJ [P in_IN [N the_ATI following_JJ shades_NNS X_&FW N] P] J] S]
Desired bracketing : (S(J)(P(N))(N))
Actual bracketing : (S(J(P(N))))

repeat_VB from_IN X_&FW all_RB round_RP
Desired parse : [S[V repeat_VB V][P from_IN [N X_&FW N]P][R all_RB round_RP R] ._. S]
Actual parse : [S [V repeat_VB V] [P from_IN [Tg [Vg X_&FW all_RB round_RP Vg] Tg] P] S]
Desired bracketing : (S(V)(P(N))(R))
Actual bracketing : (S(V)(P(Tg(Vg))))

he_PP3A did_DOD not_XNOT mention_VB personal_JJ talks_NNS with_IN 0Dr_NPT Adenauer_NP the_ATI West_NP German_JNP Chancellor_NPT
Desired parse : [S[Na he_PP3A Na][V did_DOD not_XNOT mention_VB V][N personal_JJ talks_NNS [P with_IN [N \0Dr_NPT Adenauer_NP ,_, [N the_ATI West_NP German_JNP Chancellor_NPT N]N]P]N] ._. S]
Actual parse : [S [Na he_PP3A Na] [V did_DOD not_XNOT mention_VB V] [N personal_JJ talks_NNS [Vi with_IN [Vn 0Dr_NPT Adenauer_NP Vn] [N the_ATI West_NP German_JNP Chancellor_NPT N] Vi] N] S]
Desired bracketing : (S(Na)(V)(N(P(N(N)))))
Actual bracketing : (S(Na)(V)(N(Vi(Vn)(N))))

the_ATI most_QL extraordinary_JJ things_NNS are_BER happening_VBG
Desired parse : *'_*' [S[N the_ATI [J most_QL extraordinary_JJ J] things_NNS N][V are_BER happening_VBG V] ._. S]
Actual parse : [S [N the_ATI most_QL N] [N extraordinary_JJ things_NNS N] [V are_BER happening_VBG V] S]
Desired bracketing : (S(N(J))(V))
Actual bracketing : (S(N)(N)(V))

the_ATI question_NN their_PP$ status_NN in_IN an_AT independent_JJ Uganda_NP
Desired parse : [S[N the_ATI question_NN N] :_: [N their_PP$ status_NN [P in_IN [N an_AT independent_JJ Uganda_NP N]P]N] ._. S]
Actual parse : [S [N the_ATI [N question_NN their_PP$ status_NN [P in_IN [N an_AT independent_JJ Uganda_NP N] P] N] N] S]
Desired bracketing : (S(N)(N(P(N))))
Actual bracketing : (S(N(N(P(N)))))

all_ABN four_CD were_BED conservative_JJ strongholds_NNS
Desired parse : [S[N all_ABN four_CD N][V were_BED V][N conservative_JJ strongholds_NNS N] ._. S]
Actual parse : [S [N all_ABN four_CD were_BED N] [N conservative_JJ strongholds_NNS N] S]
Desired bracketing : (S(N)(V)(N))
Actual bracketing : (S(N)(N))

Hastings_NP Monday_NR
Desired parse : [S[N Hastings_NP N] ,_, [N Monday_NR N] ._. S]
Actual parse : [S [N Hastings_NP Monday_NR N] S]
Desired bracketing : (S(N)(N))
Actual bracketing : (S(N))

it_PP3 should_MD be_BE sited_VBN in_IN a_AT more_QL remote_JJ area_NN
Desired parse : [S[N it_PP3 N][V should_MD be_BE sited_VBN V][P in_IN [N a_AT [J more_QL remote_JJ J] area_NN N]P] ._. S]

Actual parse : [S [N it_PP3 N] [V should_MD be_BE sited_VBN V] [P in_IN [N a_AT more_QL remote_JJ area_NN N] P] S]
Desired bracketing : (S(N)(V)(P(N(J))))
Actual bracketing : (S(N)(V)(P(N)))

0Mr_NPT Cooper_NP suggested_VBD that_CS the_ATI distortion_NN arose_VBD from_IN enhanced_JJ payments_NNS agreed_VBN at_IN local_JJ level_NN
Desired parse : [S[N \0Mr_NPT Cooper_NP N][V suggested_VBD V][Fn that_CS [N the_ATI distortion_NN N][V arose_VBD V][P from_IN [N enhanced_JJ payments_NNS [Tn[Vn agreed_VBN Vn][P at_IN [N local_JJ level_NN N]P]Tn]N]P]Fn] ._. S]
Actual parse : [S [N 0Mr_NPT Cooper_NP N] [V suggested_VBD V] [Fn that_CS [N the_ATI distortion_NN N] [V arose_VBD V] [P from_IN [N enhanced_JJ payments_NNS [Vn agreed_VBN Vn] [P at_IN [N local_JJ level_NN N] P] N] P] Fn] S]
Desired bracketing : (S(N)(V)(Fn(N)(V)(P(N(Tn(Vn)(P(N)))))))
Actual bracketing : (S(N)(V)(Fn(N)(V)(P(N(Vn)(P(N))))))

daily_JJ telegraph_NN political_JJ correspondent_NN
Desired parse : [S[N daily_JJ telegraph_NN [N political_JJ correspondent_NN N]N] ._. S]
Actual parse : [S [N daily_JJ telegraph_NN political_JJ correspondent_NN N] S]
Desired bracketing : (S(N(N)))
Actual bracketing : (S(N))

Westminster_NP Wednesday_NR
Desired parse : [S[N Westminster_NP N] ,_, [N Wednesday_NR N] ._. S]
Actual parse : [S [N Westminster_NP Wednesday_NR N] S]
Desired bracketing : (S(N)(N))
Actual bracketing : (S(N))

he_PP3A returned_VBD five_CD minutes_NNS later_RBR to_TO stand_VB just_RB inside_IN the_ATI doorway_NN looking_VBG more_QL composed_JJ
Desired parse : [S[Na he_PP3A Na][V returned_VBD V][R[N five_CD minutes_NNS N] later_RBR R][Ti[Vi to_TO stand_VB Vi][P just_RB inside_IN [N the_ATI doorway_NN N]P][Tg[Vg looking_VBG Vg][J more_QL composed_JJ J]Tg]Ti] ._. S]
Actual parse : [S [Na he_PP3A Na] [Fn [V returned_VBD V] [R five_CD [N minutes_NNS N] R] [V later_RBR to_TO V] [Ti [Vg stand_VB Vg] [R just_RB R] [P inside_IN [N the_ATI doorway_NN [Vg looking_VBG Vg] N] P] [R more_QL composed_JJ R] Ti] Fn] S]
Desired bracketing : (S(Na)(V)(R(N))(Ti(Vi)(P(N))(Tg(Vg)(J))))
Actual bracketing : (S(Na)(Fn(V)(R(N))(V)(Ti(Vg)(R)(P(N(Vg)))(R))))

Team_NP Spirit_NP nap_NN to_TO repeat_VB his_PP$ Mildmay_NP win_NN
Desired parse : [S[N Team_NP Spirit_NP nap_NN N][Ti[Vi to_TO repeat_VB Vi][N his_PP$ Mildmay_NP win_NN N]Ti] ._. S]
Actual parse : [S [N Team_NP N] [N Spirit_NP nap_NN [Vi to_TO repeat_VB Vi] N] [N his_PP$ Mildmay_NP win_NN N] S]
Desired bracketing : (S(N)(Ti(Vi)(N)))
Actual bracketing : (S(N)(N(Vi))(N))

we_PP1AS wish_VB to_TO negotiate_VB on_IN one_CD1 of_INO your_PP$ ideas_NNS an_AT eased_JJ form_NN of_INO the_ATI retain_NN and_CC transfer_NN system_NN
Desired parse : *'_*' [S[Na we_PP1AS Na][V wish_VB V][Ti[Vi to_TO negotiate_VB Vi][P on_IN [N one_CD1 [Po of_INO [N your_PP$ ideas_NNS *-_*- [N an_AT eased_JJ form_NN [Po of_INO [N the_ATI [NN& retain_NN [NN+ and_CC transfer_NN NN+]NN&] system_NN N]Po]N]N]Po]N]P]Ti] ._. S]
Actual parse : [S [Vn [S& [Rq we_PP1AS Rq] [T [V wish_VB V] [Vi to_TO negotiate_VB Vi] [P on_IN [N one_CD1 [Po of_INO [N your_PP$ ideas_NNS N] Po] [Vr an_AT eased_JJ form_NN Vr] N] P] T] S&] Vn] [Na& of_INO [N- the_ATI retain_NN [N+ and_CC transfer_NN system_NN N+] N-] Na&] S]
Desired bracketing : (S(Na)(V)(Ti(Vi)(P(N(Po(N(N(Po(N(NN&(NN+))))))))))
Actual bracketing : (S(Vn(S&(Rq)(T(V)(Vi)(P(N(Po(N))(Vr))))))(Na&(N-(N+))))

Oxford_NP University_NPL 5_CD
Desired parse : [S[N Oxford_NP University_NPL N] ..._... [N 5_CD N] ._. S]
Actual parse : [S [N Oxford_NP University_NPL 5_CD N] S]

Desired bracketing : (S(N)(N))
Actual bracketing : (S(N))

royal_JJ navy_NN 3_CD
Desired parse : [S[N royal_JJ navy_NN N] ..._... [N 3_CD N] ._. S]
Actual parse : [S [N royal_JJ navy_NN 3_CD N] S]
Desired bracketing : (S(N)(N))
Actual bracketing : (S(N))

are_BER you_PP2 lonesome_JJ tonight_NR
Desired parse : '_*' [S[V are_BER V][N you_PP2 N] [J lonesome_JJ J][N tonight_NR N] ?_? S]
Actual parse : [S [V are_BER V] [N you_PP2 lonesome_JJ tonight_NR N] S]
Desired bracketing : (S(V)(N)(J)(N))
Actual bracketing : (S(V)(N))

it_PP3 will_MD take_VB 18_CD months_NNS to_TO put_VB 0no_NN 1a_CD in_IN habitable_JJ order_NN
Desired parse : [S[N it_PP3 N][V will_MD take_VB V][N 18_CD months_NNS N] [Ti[Vi to_TO put_VB Vi][N \0no_NN 1a_CD N][P in_IN [N habitable_JJ order_NN N]P]Ti] ._. S]
Actual parse : [S [N it_PP3 N] [V will_MD take_VB V] [N 18_CD [Fn [N months_NNS N] [Vg to_TO put_VB Vg] [N 0no_NN 1a_CD N] [P in_IN [N habitable_JJ order_NN N] P] Fn] N] S]
Desired bracketing : (S(N)(V)(N)(Ti(Vi)(N)(P(N))))
Actual bracketing : (S(N)(V)(N(Fn(N)(Vg)(N)(P(N)))))

wanted_VBN a_AT star_NN
Desired parse : [S[Vn wanted_VBN Vn] :_: [N a_AT star_NN N] ._. S]
Actual parse : [S [V wanted_VBN V] [N a_AT star_NN N] S]
Desired bracketing : (S(Vn)(N))
Actual bracketing : (S(V)(N))

then_RN there_EX were_BED wines_NNS to_IN order_NN
Desired parse : [S[R then_RN R][E there_EX E][V were_BED V][N wines_NNS [P to_IN [N order_NN N]P]N] ._. S]
Actual parse : [S [R then_RN R] [E there_EX E] [Vr were_BED Vr] [N wines_NNS [P to_IN [N order_NN N] P] N] S]
Desired bracketing : (S(R)(E)(V)(N(P(N))))
Actual bracketing : (S(R)(E)(Vr)(N(P(N))))

this_DT will_MD be_BE true_JJ however_WRB the_ATI money_NN is_BEZ shared_VBN
Desired parse : [S[N this_DT N][V will_MD be_BE V][J true_JJ J][Fa[Rq however_WRB Rq][N the_ATI money_NN N][V is_BEZ shared_VBN V]Fa] ._. S]
Actual parse : [S [N this_DT N] [V will_MD be_BE V] [Na true_JJ [Rq however_WRB the_ATI money_NN Rq] Na] [V is_BEZ shared_VBN V] S]
Desired bracketing : (S(N)(V)(J)(Fa(Rq)(N)(V)))
Actual bracketing : (S(N)(V)(Na(Rq))(V))

they_PP3AS may_MD have_HV to_TO call_VB up_RP the_ATI reinforcement_NN of_INO the_ATI common_JJ market_NN
Desired parse : [S[Na they_PP3AS Na][V may_MD have_HV V][Ti[Vi to_TO call_VB Vi][R up_RP R][N the_ATI reinforcement_NN [Po of_INO [N the_ATI common_JJ market_NN N]Po]N]Ti] ._. S]
Actual parse : [S [Na they_PP3AS Na] [V may_MD have_HV V] [Ti [Vi to_TO call_VB Vi] [P up_RP P] [N the_ATI reinforcement_NN [Po of_INO [N the_ATI common_JJ market_NN N] Po] N] Ti] S]
Desired bracketing : (S(Na)(V)(Ti(Vi)(R)(N(Po(N)))))
Actual bracketing : (S(Na)(V)(Ti(Vi)(P)(N(Po(N)))))

Belgium_NP is_BEZ accused_VBN without_IN a_AT scrap_NN of_INO evidence_NN of_INO being_BEG implicated_VBN in_IN the_ATI murder_NN of_INO Patrice_NP Lumumba_NP
Desired parse : [S[N Belgium_NP N][V is_BEZ accused_VBN V] *-_*- [P without_IN [N a_AT scrap_NN [Po of_INO [N evidence_NN N]Po]N]P] *-_*- [Po of_INO [Tg[Vg being_BEG implicated_VBN Vg][P in_IN [N the_ATI murder_NN [Po of_INO [N Patrice_NP Lumumba_NP N]Po]N]P]Tg]Po] ._. S]
Actual parse : [S [N Belgium_NP N] [V is_BEZ accused_VBN V] [P without_IN [N a_AT scrap_NN [Po of_INO [N evidence_NN [Po of_INO being_BEG implicated_VBN Po] [P in_IN [N the_ATI murder_NN [Po of_INO [N Patrice_NP Lumumba_NP N] Po] N] P] N] Po] N] P] S]
Desired bracketing : (S(N)(V)(P(N(Po(N))))(Po(Tg(Vg)(P(N(Po(N)))))))

Actual bracketing : (S(N)(V)(P(N(Po(N(Po)(P(N(Po(N))))))))

we_PP1AS have_HV fought_VBN beside_IN Belgium_NP in_IN two_CD world_NN wars_NNS
Desired parse : [S[Na we_PP1AS Na][V have_HV fought_VBN V][P beside_IN [N Belgium_NP N]P][P in_IN [N two_CD world_NN wars_NNS N]P] ._. S]
Actual parse : [S [Na we_PP1AS Na] [V have_HV fought_VBN V] [P beside_IN [N Belgium_NP N] [P in_IN [N two_CD world_NN wars_NNS N] P] P] S]
Desired bracketing : (S(Na)(V)(P(N))(P(N)))
Actual bracketing : (S(Na)(V)(P(N)(P(N))))

lucky_JJ lucky_JJ housewives_NNS
Desired parse : [S[N[JJ& lucky_JJ ,_, [JJ- lucky_JJ JJ-]JJ&] housewives_NNS N] !_! S]
Actual parse : [S [N lucky_JJ lucky_JJ housewives_NNS N] S]
Desired bracketing : (S(N(JJ&(JJ-))))
Actual bracketing : (S(N))

out_RP and_CC about_RP
Desired parse : [S[R[RP& out_RP [RP+ and_CC about_RP RP+]RP&]R] ._. S]
Actual parse : [S [V out_RP and_CC V] [V about_RP V] S]
Desired bracketing : (S(R(RP&(RP+))))
Actual bracketing : (S(V)(V))

how_WRB many_AP serfs_NNS
Desired parse : [S[N[Dq how_WRB many_AP Dq] serfs_NNS N] ?_? S]
Actual parse : [S [Po how_WRB many_AP serfs_NNS Po] S]
Desired bracketing : (S(N(Dq)))
Actual bracketing : (S(Po))

then_RN comes_VBZ engineering_NN followed_VBN by_IN iron_NN and_CC steel_NN
Desired parse : [S[R then_RN R][V comes_VBZ V][N engineering_NN ,_, [Tn[Vn followed_VBN Vn][P by_IN [N[NN& iron_NN [NN+ and_CC steel_NN NN+]NN&]N]P]Tn]N] ._. S]
Actual parse : [S [R then_RN R] [V comes_VBZ V] [N engineering_NN [Vn followed_VBN Vn] [P by_IN [N NN& N] P] N] S]
Desired bracketing : (S(R)(V)(N(Tn(Vn)(P(N(NN&(NN+)))))))
Actual bracketing : (S(R)(V)(N(Vn)(P(N))))

how_WRB long_RB
Desired parse : [S[Rq how_WRB long_RB Rq] ?_? S]
Actual parse : [S [Rq how_WRB Rq] [R long_RB R] S]
Desired bracketing : (S(Rq))
Actual bracketing : (S(Rq)(R))

you_PP2 re_BER an_AT American_NNP eh_UH
Desired parse : *'_*' [S[N you_PP2 N][V 're_BER V][N an_AT American_NNP N] ,_, [U eh_UH U] ?_? S] **'_**'
Actual parse : [S [N you_PP2 N] [S& [V re_BER V] S&] [N an_AT American_NNP N] [M eh_UH M] S]
Desired bracketing : (S(N)(V)(N)(U))
Actual bracketing : (S(N)(S&(V))(N)(M))

dispatched_VBN
Desired parse : [S[Vn dispatched_VBN Vn] ..._... ._. S]
Actual parse : [S [V dispatched_VBN V] S]
Desired bracketing : (S(Vn))
Actual bracketing : (S(V))

national_JJ president_NN of_INO the_ATI association_NN of_INO engineering_NN and_CC shipbuilding_NN draughtsmen_NNS writing_VBG in_IN his_PP$ personal_JJ capacity_NN
Desired parse : [S[N national_JJ president_NN [Po of_INO [N the_ATI association_NN [Po of_INO [N[NN& engineering_NN [NN+ and_CC shipbuilding_NN NN+]NN&] draughtsmen_NNS N]Po]N]Po]N] ,_, [Tg[Vg writing_VBG Vg][P in_IN [N his_PP$ personal_JJ capacity_NN N]P]Tg] :_: S]

Actual parse : [S& [S& [N national_JJ president_NN [Po of_INO [N the_ATI association_NN [Po of_INO [N engineering_NN N] Po] N] Po] N] S&] [S+ and_CC [N shipbuilding_NN draughtsmen_NNS N] [V writing_VBG V] [P in_IN [N his_PP$ personal_JJ capacity_NN N] P] S+] S&].
Desired bracketing : (S(N(Po(N(Po(N(NN&(NN+)))))))(Tg(Vg)(P(N))))
Actual bracketing : (S&(S&(N(Po(N(Po(N))))))(S+(N)(V)(P(N))))

property_NN tycoonery_NN in_IN the_ATI 0GPO_NP what_WDT s_BEZ happening_VBG
Desired parse : [S&[N property_NN tycoonery_NN [P in_IN [N the_ATI \0G.P.O_NP N]P]N] *-_*- [S-[Nq what_WDT Nq][V 's_BEZ happening_VBG V]S-] ?_? S&]
Actual parse : [S [N property_NN tycoonery_NN [P in_IN [N the_ATI 0GPO_NP N] P] N] [V what_WDT s_BEZ happening_VBG V] S]
Desired bracketing : (S&(N(P(N)))(S-(Nq)(V)))
Actual bracketing : (S(N(P(N)))(V))

7_CD Holders_NP Hill_NPL Avenue_NPL 0NW4_NP
Desired parse : [S[N 7_CD ,_, [N Holders_NP Hill_NPL Avenue_NPL N] ,_, [N \0N.W.4_NP N]N] ._. S]
Actual parse : [S [N 7_CD [N Holders_NP Hill_NPL Avenue_NPL 0NW4_NP N] N] S]
Desired bracketing : (S(N(N)(N)))
Actual bracketing : (S(N(N)))

Glyndebourne_NP contemporary_JJ
Desired parse : [S[N Glyndebourne_NP N] *'_*' [J contemporary_JJ J] **'_**' ._. S]
Actual parse : [S [N Glyndebourne_NP contemporary_JJ N] S]
Desired bracketing : (S(N)(J))
Actual bracketing : (S(N))

journal_NN debut_NN at_IN Cheltenham_NP
Desired parse : *'_*' [S[N journal_NN **'_**' debut_NN N][P at_IN [N Cheltenham_NP N]P] ._. S]
Actual parse : [S [N journal_NN debut_NN [P at_IN [N Cheltenham_NP N] P] N] S]
Desired bracketing : (S(N)(P(N)))
Actual bracketing : (S(N(P(N))))

0Mr_NPT Richardsons_NP$ skilful_JJ direction_NN
Desired parse : [S[N[G \0Mr_NPT Richardson's_NP$ G] skilful_JJ direction_NN N] ._. S]
Actual parse : [S [N 0Mr_NPT Richardsons_NP$ skilful_JJ direction_NN N] S]
Desired bracketing : (S(N(G)))
Actual bracketing : (S(N))

sing_VB slightly_RB flat_RB
Desired parse : [S[V sing_VB V][R slightly_RB flat_RB R] ._. S]
Actual parse : [S [V sing_VB V] [R slightly_RB R] [R flat_RB R] S]
Desired bracketing : (S(V)(R))
Actual bracketing : (S(V)(R)(R))

now_RN along_RP comes_VBZ his_PP$ solo_JJB disc_NN featuring_VBG two_CD of_INO his_PP$ own_AP compositions_NNS deerstalker_NN and_CC almost_RB grown_VBN up_RP
Desired parse : [S[R now_RN R][R along_RP R][V comes_VBZ V][N his_PP$ solo_JJB disc_NN ,_, [Tg[Vg featuring_VBG Vg][N two_CD [Po of_INO [N his_PP$ own_AP compositions_NNS ,_, *'_*' [N& deerstalker_NN **'_**' [N+ and_CC *'_*' almost_RB grown_VBN up_RP N+]N&]N]Po]N]Tg]N] ._. S]
Actual parse : [S [R now_RN R] [V along_RP comes_VBZ V] [N his_PP$ N] [Ti [D solo_JJB [N disc_NN N] [Vg featuring_VBG Vg] D] [R two_CD [Po of_INO [N his_PP$ N] Po] [N own_AP compositions_NNS deerstalker_NN N] R] [S and_CC [V almost_RB grown_VBN V] [R up_RP R] S] Ti] S]
Desired bracketing : (S(R)(R)(V)(N(Tg(Vg)(N(Po(N(N&(N+)))))))))
Actual bracketing : (S(R)(V)(N)(Ti(D(N)(Vg))(R(Po(N))(N))(S(V)(R))))

but_CC nothing_PN crazy_JJ about_IN his_PP$ pianistics_NN
Desired parse : [S but_CC [N nothing_PN crazy_JJ [P about_IN [N his_PP$ pianistics_NN N]P]N] ._. S]
Actual parse : [S [N PP3AS+ N] S]
Desired bracketing : (S(N(P(N))))
Actual bracketing : (S(N))

the_ATI soldier_NN who_WP was_BEDZ scared_JJ
Desired parse : [S[N the_ATI soldier_NN [Fr[Nq who_WP Nq][V was_BEDZ V][J scared_JJ J]Fr]N] ._. S]
Actual parse : [S [N the_ATI [Fn [N soldier_NN N] [Nq who_WP Nq] [V was_BEDZ V] [J scared_JJ J] Fn] N] S]
Desired bracketing : (S(N(Fr(Nq)(V)(J))))
Actual bracketing : (S(N(Fn(N)(Nq)(V)(J))))

WebberDouglas_NP School_NPL triple_JJ bill_NN
Desired parse : [S[N Webber-Douglas_NP School_NPL N] :_: [N triple_JJ bill_NN N] ._. S]
Actual parse : [S [N WebberDouglas_NP School_NPL triple_JJ bill_NN N] S]
Desired bracketing : (S(N)(N))
Actual bracketing : (S(N))

Behan_NP bestows_VBZ an_AT accolade_NN on_IN Delaney_NP
Desired parse : [S[N Behan_NP N][V bestows_VBZ V][N an_AT accolade_NN N][P on_IN [N Delaney_NP N]P] ._. S]
Actual parse : [S [N Behan_NP N] [V bestows_VBZ V] [N an_AT accolade_NN [P on_IN [N Delaney_NP N] P] N] S]
Desired bracketing : (S(N)(V)(N)(P(N)))
Actual bracketing : (S(N)(V)(N(P(N))))

that_DT s_BEZ as_QL bloody_RB silly_JJ as_CS calling_VBG a_AT RollsRoyce_NP a_AT type_NN of_INO transport_NN
Desired parse : [S[N that_DT N][V 's_BEZ V][J[R as_QL bloody_RB R] silly_JJ [Fc as_CS [Tg[Vg calling_VBG Vg][N a_AT Rolls-Royce_NP N][N a_AT type_NN [Po of_INO [N transport_NN N]Po]N]Tg]Fc]J] ._. S]
Actual parse : [S [N that_DT N] [V s_BEZ V] [R as_QL bloody_RB R] [J silly_JJ [Fc as_CS [V calling_VBG V] [N a_AT [N RollsRoyce_NP a_AT type_NN [Po of_INO [N transport_NN N] Po] N] N] Fc] J] S]
Desired bracketing : (S(N)(V)(J(R)(Fc(Tg(Vg)(N)(N(Po(N)))))))
Actual bracketing : (S(N)(V)(R)(J(Fc(V)(N(N(Po(N)))))))

he_PP3A was_BEDZ in_IN a_AT fight_NN after_IN telling_VBG a_AT Canadian_NNP during_IN a_AT chat_NN about_IN spaceflight_NN
Desired parse : [S[Na he_PP3A Na][V was_BEDZ V][P in_IN [N a_AT fight_NN N]P][P after_IN [Tg[Vg telling_VBG Vg][N a_AT Canadian_NNP N] ,_, [P during_IN [N a_AT chat_NN [P about_IN [N space-flight_NN N]P]N]P]Tg]P] :_: S]
Actual parse : [S [Na he_PP3A Na] [V was_BEDZ V] [P in_IN [N a_AT fight_NN [P after_IN [T [Vg telling_VBG Vg] [N a_AT Canadian_NNP [P during_IN [N a_AT chat_NN [P about_IN [N spaceflight_NN N] P] N] P] N] T] P] N] P] S]
Desired bracketing : (S(Na)(V)(P(N))(P(Tg(Vg)(N)(P(N(P(N)))))))
Actual bracketing : (S(Na)(V)(P(N(P(T(Vg)(N(P(N(P(N))))))))))

books_NNS in_IN brief_JJ
Desired parse : [S[N books_NNS [P in_IN [N brief_JJ N]P]N] ._. S]
Actual parse : [S [N books_NNS [P in_IN [J brief_JJ J] P] N] S]
Desired bracketing : (S(N(P(N))))
Actual bracketing : (S(N(P(J))))

0Mr_NPT Hudson_NP is_BEZ an_AT American_JNP millionaire_NN who_WP spends_VBZ each_DT September_NR in_IN his_PP$ Italian_JNP villa_NN and_CC the_ATI company_NN of_INO Signorina_NPT Lollobrigida_NP
Desired parse : [S[N \0Mr_NPT Hudson_NP N][V is_BEZ V][N an_AT American_JNP millionaire_NN [Fr[Nq who_WP Nq][V spends_VBZ V][N each_DT September_NR N][P in_IN [N& his_PP$ Italian_JNP villa_NN [N+ and_CC the_ATI company_NN [Po of_INO [N Signorina_NPT Lollobrigida_NP N]Po]N+]N&]P]Fr]N] ._. S]
Actual parse : [S [N 0Mr_NPT Hudson_NP N] [V is_BEZ V] [N an_AT [N American_JNP millionaire_NN who_WP N] [Fn [V spends_VBZ V] [N each_DT N] [Vr September_NR Vr] [P in_IN [N& his_PP$ Italian_JNP villa_NN [N+ and_CC the_ATI company_NN [Po of_INO [N Signorina_NPT Lollobrigida_NP N] Po] N+] N&] P] Fn] N] S]
Desired bracketing : (S(N)(V)(N(Fr(Nq)(V)(N)(P(N&(N+(Po(N))))))))
Actual bracketing : (S(N)(V)(N(N)(Fn(V)(N)(Vr)(P(N&(N+(Po(N))))))))

or_CC anything_PN more_QL true_JJ than_IN this_DT
Desired parse : [S or_CC [N anything_PN N][J more_QL true_JJ [P than_IN [N this_DT N]P]J] ?_? S]
Actual parse : [S [N PP3AS+ N] S]
Desired bracketing : (S(N)(J(P(N))))
Actual bracketing : (S(N))

hands_NNS and_CC feet_NNS
Desired parse : [S[N[NNS& hands_NNS [NNS+ and_CC feet_NNS NNS+]NNS&]N] ._. S]
Actual parse : [S [N NNS& N] S]

Desired bracketing : (S(N(NNS&(NNS+))))
Actual bracketing : (S(N))

Appendix M

Published Works

Tepper, J. A., Powell, H., Palmer-Brown, D. (1995). Ambiguity resolution in a connectionist parser. In *Proceedings of the 4th International Conference on the Cognitive Science of Natural Language Processing*. Dublin City University, July 1995.

Ambiguity resolution in a Connectionist Parser

Jonathan A. Tepper, Heather Powell, Dominic Palmer-Brown

Parallel Research Group, Department Of Computing,The Nottingham Trent University, Burton Street,Nottingham NG1 4BU jtc@doc.ntu.ac.uk

Abstract

Connectionism is a relatively new approach to natural language processing and has comparatively few standard methods for syntax analysis and parsing relative to classical symbolic methods. The strengths of connectionist systems have been established empirically : automatic learning, tolerance to noisy input, graceful degradation and generalization. Classical rule-based techniques are well understood but tend to be intolerant of minor variations that fail to adhere to predefined rules. This paper presents a hybrid architecture based on symbolic and subsymbolic processing. It is able to deterministically parse and resolve lexical ambiguity in sentences based on a context-free grammar. The system parses sentences from right-to-left by iteratively processing sentence constituents and reduced input from the previous time steps. Reductions are carried out symbolically. A symbolic stack is used to store intermediate states of the parse. This approach allows sentences of arbitrary length to be parsed and is able to disambiguate sentences based on syntactic context.

1. Introduction

There have been some successful connectionist approaches to parsing natural language sentences, e.g. Fanty [1] applied localist techniques for context-free parsing; Selman [2] utilized ideas based upon the Boltzmann machine for syntactic parsing. Rumelhart & McClelland [3a] provided further insight into connectionist capabilities and presented features such as generalising to the past tense of verbs[3b]. Since 1986, many more CNLP papers have appeared with vast amounts of research regarding various aspects of syntax and parsing (e.g. Hanson & Kegl [4]; Howells [5]; Giles et al [6]; Sun [7]; Zeng, Goodman & Smyth [8]).

The majority of research has maintained a degree of symbolic processing but since the criticisms made by Fodor & Pylyshyn [9] many connectionists have moved towards a purist approach. Connectionist researchers have since developed models to cope with temporal structure [10] and complex recursive data structures [11,12]. These distributed representative models provided a definite contrast to explicitly labelled structures used in classical symbolic methods.

Deterministic parsers are conceptually a plausible model of human sentence processing[13]. Classical deterministic parsers such as PARSIFAL[13] are rule-based and have difficulty in processing ungrammatical and ambiguous sentences.. Ad-hoc extensions (i.e. PARAGRAM[14], ROBIE[15],LPARSIFAL[16]) can provide a limited solution to these and other problems including further language acquisition. This paper presents a hybrid architecture based on symbolic and subsymbolic (*connectionist*) processing. It integrates the advantages of both *classical* symbolic and subsymbolic methods in order to deterministically parse and resolve lexical ambiguity in sentences based on a context-free grammar.

2. Grammatical Framework

The most significant decision to be made when developing an NLP system is the selection of the underlying grammatical framework. Wait-and-see parsers (WASP) are a plausible model of human parsing and reject the backtracking and partial structure approach used by some classical AI parsers such as ATNs, to deal with local ambiguity. This type of parser was first demonstrated by Marcus's deterministic parser PARSIFAL. A deterministic approach is taken with the proposed system to process sentences from a simple context-free grammar.

The subsymbolic component is expected to acquire its primary linguistic data from a set of rules taken from a simple context-free grammar[1]. A grammar to describe a subset of the English language could be defined as a quadruple G = (N,T,S,P) where N is a finite set of non-terminal symbols, T is the finite set of terminal symbols, S is the starting symbol, which must be a member of the set N, and P is the set of productions. The grammar used can be seen in figure 1. Each rule can be broken down into a set of training patterns. These patterns consist of a binary encoding of each syntactic constituent and constituent group (e.g noun phrase) allowed by the grammar. Such patterns form the basis of linguistic data the connectionist network acquires in order to infer the specified grammar.

N	=	S, VP, PP, NP, NP2		
T	=	verb, noun, preposition, determiner, adjective		
S	=	S		
P	=	S	→	NP VP
		S	→	VP
		VP	→	verb I verb NP I VP PP
		PP	→	preposition NP
		NP	→	determiner NP2 I NP2 I NP PP
		NP2	→	noun I adjective NP2

Figure 1 *Simple grammar for a subset of the English language.*

Sharkey[17] made four observations about primary linguistic data : it should consist of a subset of the sentences about which the language user will have reliable intuitions; the data is noisy since children experience ungrammatical sentences during conversations; there is little negative evidence although sets of ungrammatical or incomplete sentences are given to the child; and finally, the data is encountered sequentially. These characteristics are taken into consideration when designing an appropriate training strategy for the connectionist network.

3. Architecture And Parsing Mechanism

The system parses sentences from right-to-left by iteratively processing sentence constituents and reduced input from the previous time steps. The right-to-left strategy has been selected due to the assumption that the complicated phrases of a sentence normally belong to its predicate. In order to deterministically parse and disambiguate natural language sentences the hybrid architecture consists of the following components :

- Subsymbolic element to decide the type of reduction to perform, provide lexical alternatives for ambiguous words, and for further language acquisition.
- Input buffer for sentence constituents.
- Stack to store intermediate constituents during the parsing process.
- Symbolic lexicon to determine the syntactic category of words.
- Reduction table to store reductions and constituents belonging to that reduction. This forms a symbolic representation of the parse tree for the sentence.

The subsymbolic element consists of a three-layered backpropagation network[3a] which consists of 15 input nodes fully interconnected to a minimum of 15 hidden units. The hidden nodes are split into two segments. The first segment is fully interconnected to the first 8 output nodes and the second segment is fully interconnected to the remaining 12 output nodes. This architecture can be seen in figure 2.

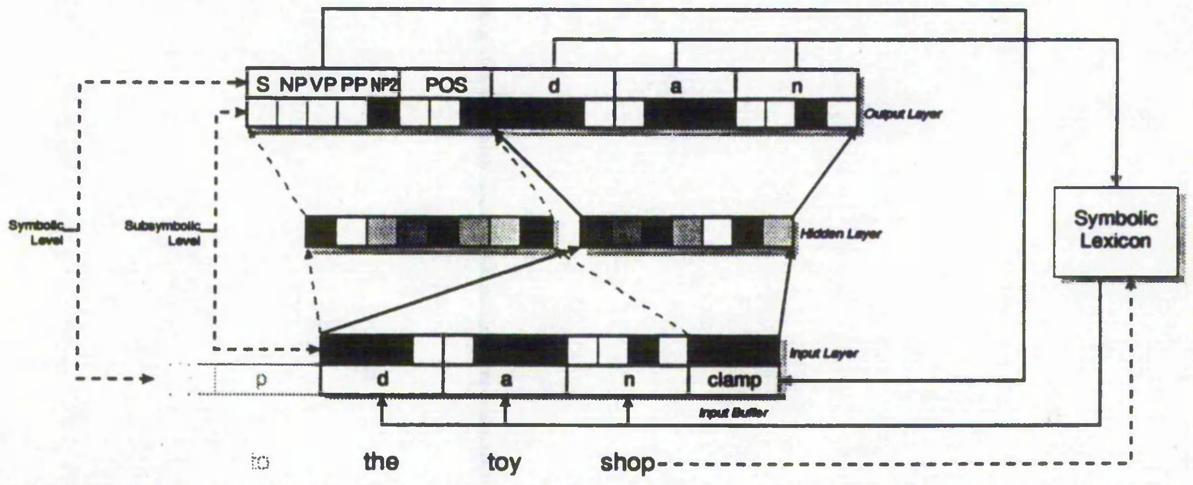


Figure 2 *Hybrid architecture to parse and disambiguate simple natural language sentences.*

Marcus demonstrated that no more than three to four constituents are needed before an appropriate reduction can be performed. Therefore, the input buffer contains at most three constituents (e.g noun, verb) and/or constituent groups (e.g noun phrase group). There are a total of 12 input nodes representing the contents of the input buffer. The constituents initial syntactic category is decided by the symbolic lexicon. Each syntactic category and group is represented by a four bit binary code. The connectionist network takes this binary coded representation of the input buffer as input. If after three constituents the connectionist network cannot disambiguate or decide the correct syntactic grouping, for at least two constituents, an error is indicated and the sentence deemed ungrammatical. But if a reduction can be performed it is done so symbolically. A further *n* sentence constituents will then enter the input buffer, where *n* is 1 - the number of constituents in the reduction.

There are an additional three input nodes that are used to *clamp* constituents contained in the input buffer. These nodes will be termed *clamp nodes*. Each clamp node corresponds to a position in the input buffer. For example, if only clamp node one is activated then the syntactic category of the constituent in position one (leftmost) of the input buffer cannot change. This will affect the alternative input that is produced on the output nodes and is used as an aid to lexical disambiguation.

There are a total of 20 output units which can be broken down into two segments. The first segment consists of 8 nodes and represent actions/reductions to be carried out and positional information to inform the *symbolic processor* of whereabouts in the input to perform the reduction. These nodes have a localist representation. Although there has been much criticisms of localist representations due to their similarity to classical symbolic concepts, this type of representation is used to conveniently interpret how the network is processing each sentence fragment. The five leftmost output nodes represent the type of reduction to perform (e.g NP, VP, PP, NP2, S). The next three consecutive nodes are positional units and correspond to a position in the input buffer. For example, if output node one and positional nodes 2 and 3 are activated then constituents 2 and 3 in the input buffer will be reduced to the start symbol, *S*, and hence a successful parse has resulted. Alternatively, if a reduction cannot be performed then none of these output nodes will be activated. This will cause the *disambiguation nodes* (see figure 4) to be analysed to see if the contents of the input buffer can be modified via referencing the lexicon. This output segment of the connectionist network is easier to visualise independently and can be seen in figure 3.

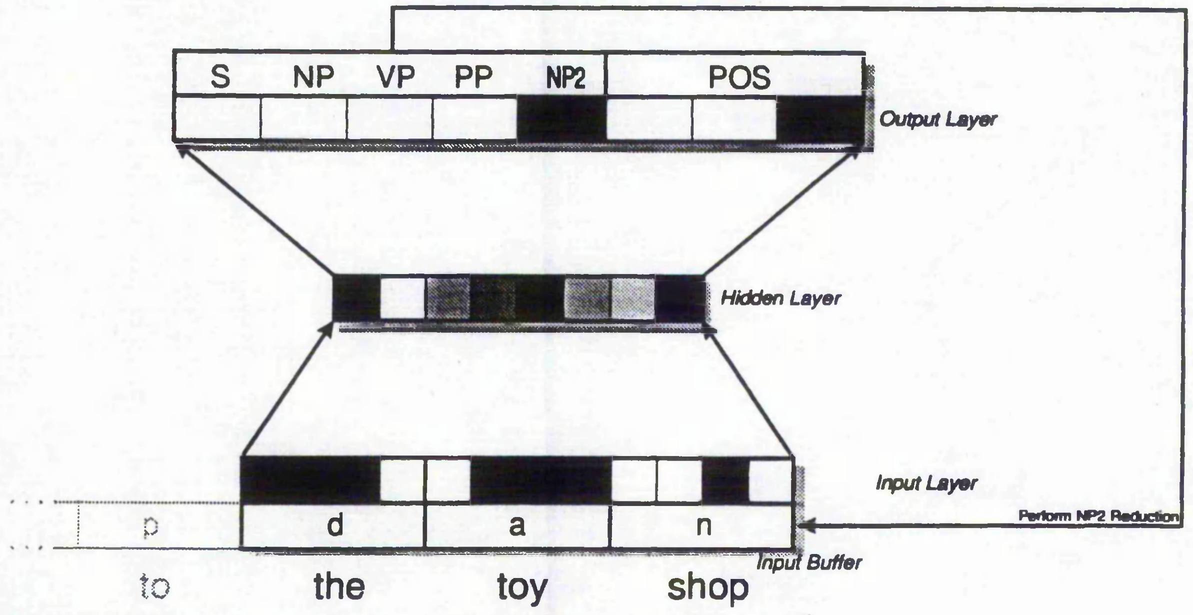


Figure 3 *Constituent reduction mechanism of the hybrid parser.*

The remaining 12 output nodes forms the second output segment of the connectionist network. These nodes are used for lexical disambiguation and are activated depending on the contents of the input buffer and clamp nodes.

3.1 Lexical Disambiguation

The 12 rightmost output nodes are termed *disambiguation nodes* and have a distributed representation. These nodes attempt to represent either the exact contents or a modification of the input buffer depending on the activations of the clamp nodes. The input clamp nodes are initially fully activated forcing the input to be reproduced on the disambiguation nodes.

If in the subphrase *the fast train*, the word *train* is initially assigned a verb category by the lexicon, a reduction is not possible. The lexicon is then checked to see if the word in the right-most buffer position possesses alternative syntactic categories. If it does not then the preceding words, in the input buffer, is processed in the same way. But if none of the current constituents can be assigned alternate syntactic categories, and no reduction is possible, then this input fragment will be rejected and thus the whole sentence interpreted as ungrammatical. In this example, the word *train* can be assigned alternate syntactic categories and this is confirmed by the lexicon. The clamp node relative to the position of *train* can now be deactivated (set to zero) and the input presented to the connectionist network.

The network will now produce a modified input pattern on its disambiguation nodes and the associated reduction on its reduction nodes. The modification being that the syntactic category for *train* is now a noun, which is valid. The input clamp nodes are then fully activated and the modified input pattern presented to the network. A reduction is now possible and parsing resumes. This segment of the network can be seen in figure 4.

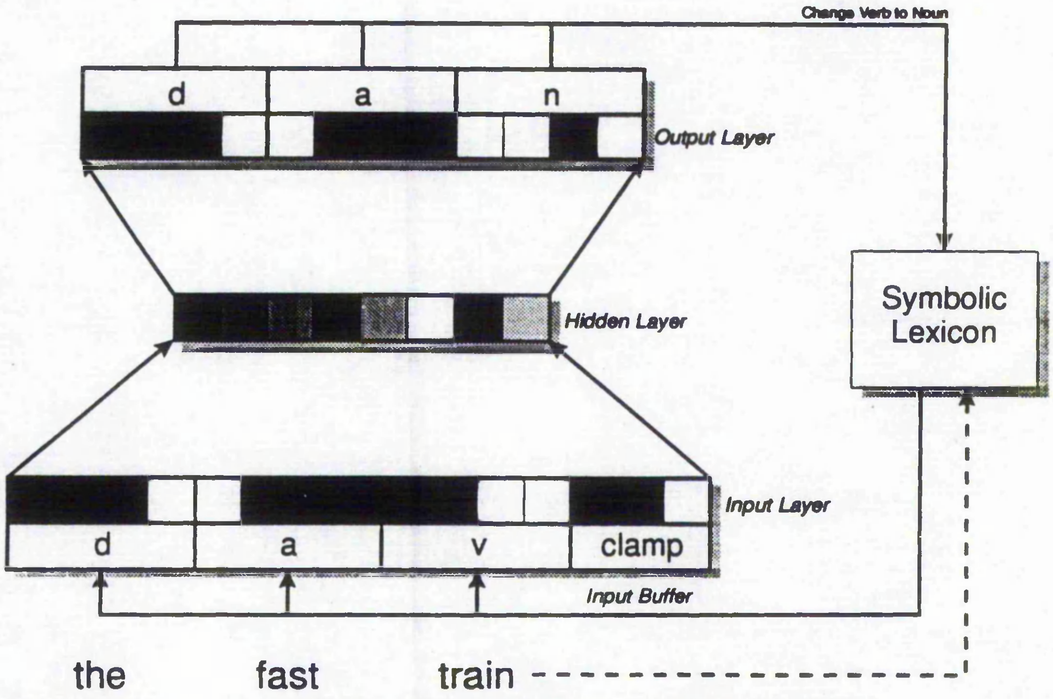


Figure 4 *Constituent disambiguation mechanism of the hybrid parser.*

4. Training And Performance

Syntactic rules derived from the context-free grammar provide a basis for training data so that the network acquires an initial linguistic structure. There are two distinct training sets, one for the reduction segment of the connectionist network and another for the disambiguation segment. Each training set is biased towards grammatical phrases which are replicated throughout the training sets. Ungrammatical phrases and random constituent groupings form the negative examples to ensure that the network is able to reject highly ungrammatical phrases.

There are eleven symbols in total including a null character and a terminating symbol to indicate the end of the input stream. The network can be presented with a maximum of 11^3 constituent combinations of length three. Each symbol has its own 4-bit binary representation. There were 62 phrases manually defined as valid, and 62 mildly invalid phrases manually defined as modifiable for disambiguation purposes. The remaining 1207 phrases were termed invalid.

All training sets consisted of all valid expressions but contained only a percentage of invalid expressions (e.g 1, 10, 20, 30, 40, 50). A number of training experiments were performed to attain the optimum number of invalid phrases, level of valid phrase replication, and number of hidden nodes for the reduction network segment. Each training pair consisted of : the binary input representation of the input buffer as input (excluding the clamp nodes), and the relevant reduction and position information. The optimum training set consisted of all valid phrases replicated by a factor of 20, and 50% of invalid phrases. All training patterns had been learnt after 10000 epochs with an RMS error of 0.033322 using 15 hidden nodes. The network was able to generalise to 92.3% of the remaining 635 invalid cases.

There are eight training patterns per phrase (i.e 10,648 total patterns) for the disambiguation network segment due to the eight possible clamp settings. Each training pair consisted of the binary input representation of the input buffer and clamping information, as input, and the relevant output response for each clamp node combination. Invalid phrases that cannot be modified have no activation on the output nodes regardless of the clamping information. The training set consists of the 496 valid training patterns and 50% of the 496 mildly ungrammatical phrases. The network was able to learn all training patterns with 15 hidden nodes.

5. Conclusions

A hybrid architecture that is able to parse sentences from a simple context-free grammar has been proposed. It is able to resolve lexical ambiguity and parse sentences of arbitrary length. The connectionist network is able to acquire linguistic knowledge from a small set of valid and a relatively large set of invalid phrases all of length three. It is able to generalise in order to reject highly ungrammatical phrases that were not contained within its training set. Also, the network is able to offer alternatives to lexically ambiguous or ungrammatical input.

Further work is under way with a large grammar[18] to assess the technique for a realistic subset of natural language.

References

[1] Fanty, M. A. (1986). Context-free parsing with connectionist networks. In *Proc. of AIP Conf. Neural Networks for Computers*. PP 140-145.

[2] Selman, B. (1985). A rule-based connectionist parsing system. In *Proc. of 7th Annual Conf. of Cognitive Science*. PP 212-221.

[3a] Rumelhart, D.E., Hinton, G.E., Williams R. J. (1986). Learning internal representations by error propagation. In *Parallel Distributed Processing, Explorations in the Microstructure of Cognition, Vol 1: Foundations*. PP 318-362. MIT Press.

[3b] Rumelhart, D., McClelland J. (1986). On learning the past tenses of English verbs.In *Parallel Distributed Processing, Explorations in the Microstructure of Cognition*, Vol 2. PP 216-271. MIT Press.

[4] Hanson, S. J., Kegl, J. (1987). PARSNIP: A connectionist network that learns natural language grammar from exposure to natural language sentences. In *Proc. of 9th Annual Conf. of Cognitive Science*. PP 106-119.

[5] Howells, T. (1988). VITAL: A connectionist parser. In *Proc. of 10th Annual Conf. of Cognitive Science*. PP 18-25.

[6] Giles, C.L., Sun, G.Z., Chen, H.H., Lee, Y.C., & Chen, D. (1990). Higher order recurrent networks and grammatical inference. In D.S. Touretzky (Ed.), *Advances in Neural Information Systems 2* . P 380.

[7] Sun, G. Z. (1993). Learning context-free grammar with enhanced pushdown automaton. In *Proc. of IEEE Conf. of Connectionist Natural Language Processing*. Part 6. PP 1-12.

[8] Zheng, Z., Goodman, R., Smyth, P. (1994). Discrete recurrent neural networks for grammatical inference. In *IEEE Transactions on Neural Networks*. Vol 5. No 2. PP 320-330.

[9] Fodor J.A., Pylyshyn Z.W. (1988). Connectionism and Cognitive Architecture: A critical analysis. In *Cognition*, Vol 28. PP 3-71.

[10] Elman J.L. (1990). Finding structure in time. In *Cognitive Science*, 14, PP 179-211.

[11] Pollack J.B. (1990) Recursive Distributed Representations. In *Artificial Intelligence*. Vol 46. PP 77-105.

[12] Smolensky P. (1990). Tensor Product Variable Binding and the Representation of Symbolic Structures in Connectionist Systems. In *Artificial Intelligence*. Vol 46. PP 159-216.

[13] Marcus, M. P. (1980). *A Theory of Syntactic Recognition for Natural Language.* Cambridge: MIT Press.

[14] Charniak, E. (1983). A parser with something for everyone. In *Parsing Natural Language*. New York: Academic Press.

[15] Milne, R. (1986). Resolving lexical ambiguity in a deterministic parser. In *Computational Linguistics*, 12, PP 1-12.

[16] Berwick, R. C. (1985). *The Acquisition of Syntactic Knowledge*. Cambridge: MIT Press.

[17] Sharkey, A.J.C., Sharkey, N. E. (1993). Connectionism and natural language. In *Proc. of IEEE Conf. of Connectionist Natural Language Processing*. Part 20. PP 1-10.

[18] Noble, H. M. (1988). *Natural Language Processing*. Oxford : Blackwell

Tepper, J. A., Powell, H., Palmer-Brown, D. (1995). Integrating symbolic and subsymbolic architectures for parsing arithmetic expressions and natural language sentences. In *Proceedings of the 3rd SNN Neural Network Symposium.* Nijmegen University, September 1995.

Integrating Symbolic And Subsymbolic Architectures For Parsing Arithmetic Expressions And Natural Language Sentences

Jonathan A. Tepper, Heather Powell, Dominic Palmer-Brown

Parallel Research Group, Department Of Computing,
The Nottingham Trent University,
Burton Street,Nottingham NG1 4BU
jte@doc.ntu.ac.uk

Extended Abstract

Connectionism is a relatively new approach to language processing and has comparatively few standard methods for syntax analysis and parsing relative to classical symbolic methods. The interest in connectionism has arisen due to its learning capability, tolerance to noisy input, and ability to generalize from previous examples. Classical rule-based techniques are well understood but tend to be intolerant of minor variations that do not strictly adhere to predefined rules.

There have been some successful connectionist approaches to parsing natural language sentences; Selman [1] utilized ideas based upon the Boltzmann machine for syntactic parsing, and Fanty [2] employed localist techniques for context-free parsing. Features such as generalising to the past tense of verbs have also been demonstrated by Rumelhart & McClelland [3]. The majority of research has maintained a degree of symbolic processing but since the criticisms of Fodor & Pylyshyn [4] many connectionists have moved towards a purist approach(e.g Elman[5], Pollack[6], and Smolensky[7]).

This paper presents a novel hybrid AI architecture which is able to parse arithmetic expressions both symbolically and subsymbolically. A method of deterministically parsing simple natural language sentences based on this architecture is also presented. A symbolic element is required for communication and interpretation. A subsymbolic component allows the representation of opaque conceptual information, temporal dynamic structures and complex non-linear structures.

A novel method for adapting the learning rate is also presented. This is suited to training sets with an imbalance between the number of examples of different pattern classes.

Parsing Arithmetic Expressions
The connectionist element of the hybrid system uses a three-layered backpropagation network[8] which consists of 12 input nodes fully interconnected to 6 hidden units. The hidden nodes are fully interconnected to one output unit, which signifies the acceptance or rejection of an expression.

There are eight unique symbols (e.g +, -,*, /,(,),operand,terminator) and the network can be presented with a maximum of 8^4 expressions of length four. Each symbol has its own 3-bit binary representation. A total of 17 expressions are defined as valid and the remaining 4079 expressions are deemed invalid. A number of training set configurations were used to investigate the best training strategy. All training sets consisted of all valid expressions but contained only a percentage of invalid expressions (e.g 1, 10, 20, 30, 40, 50). Due to the high magnitude of invalid expressions relative to valid expressions a method for pattern dependant learning rate adaption was developed.

The network parses expressions from left-to-right by iteratively reducing mathematical terms. Reductions are carried out symbolically. Symbolic structures are used to store intermediate states of the parse and reductions. This approach allows expressions of arbitrary length to be parsed. The input presented to the network at any time step is fixed at four expression constituents and/or symbols from the symbolic structure. A reduction will only be made if the first three symbols are reducible (signified by the connectionist network) and the fourth symbol is a closed parenthesis, operator with lower precedence than that in the expression, or a terminating symbol. A successful parse is encountered if a single symbol taken from the symbolic structure is the only symbol remaining in the input buffer, the terminating symbol has been encountered and the stack is empty.

An unsuccessful parse is reported if there are one or more symbols remaining within the input buffer or stack which are irreducible.

The performance of the network with the various training sets was measured by its generalisation capability against the percentage of valid and invalid expressions contained in the training set. In one case 95% of remaining cases are correctly characterised from 1% of the invalid cases.

The standard backpropagation method does not learn all of the valid cases as they constitute such a small percentage of the overall number in the training set. Two successful experiments to overcome this have been performed. One involved replicating each valid expression in the training set so that valid expressions constitute a substantial proportion of the total training examples. This increased the training set size and the level of success was dependant on the degree of replication. All the network configurations performed well, generalising to greater than 90%. When 50% of the training set consisted of invalid expressions a generalisation rate of 100% was achieved.

The second method involved adapting the learning rate for each pattern and its error. The learning rate for each pattern is calculated as follows

$$\eta = \frac{\sum_{i=0}^{p-1} |e_i|}{p} * a + b$$

Where e_i is the error for unit i, p is the total number of hidden or output units and a and b are constants which allow adjustment of the range of η. $a = 0.6$ and $b = 0.2$ were chosen to give $0.2 \leq \eta \leq 0.8$. η is therefore linearly dependent on the average absolute output error within this range.

The training set consisted of all 17 positive examples (no replication) and 50% of the remaining 4079 negative examples as before. All training patterns were learnt after 3000 epochs. The network generalises very well with an accuracy of 97.3%. Further variations of the training set have revealed similarly good results.

Parsing Natural Language Sentences

Classical deterministic parsers such as PARSIFAL[9] are rule-based. They have difficulty in processing ungrammatical and ambiguous sentences and acquiring further language. Ad-hoc extensions (i.e. PARAGRAM[10], ROBIE[11],LPARSIFAL[12]) can provide a limited solution to these problems.

The hybrid architecture proposed provides a framework for deterministic parsing, lexical disambiguation and language acquisition. The connectionist model is based on that used for the arithmetic expression parser. The input tokens consist of sentence constituents and symbols denoting a type of reduction. The output units represent actions to be carried out and positional information to inform the symbolic processor of whereabouts in the input to perform the reduction. There are also association nodes on the output that provide lexical alternatives for ambiguous input. Syntactic rules, derived from a context-free grammar[2], provide a basis for training data so that the network acquires an initial linguistic knowledge structure. Correct input fragments are autoassociated on the output nodes as learning takes place. If an incorrect fragment is present the network may produce a possible alternative. This is used as a means of lexical disambiguation. For example, in the sentence *She wanted to shop,* the word *shop* may initially be incorrectly classified as a noun. The network will produce a verb pattern on the output nodes relative to the position of *shop*.

The system parses sentences from right-to-left by iteratively processing sentence constituents and reduced input from the previous time steps. Reductions are carried out symbolically. A symbolic stack is used to store intermediate states of the parse. Unlike many connectionist parsers this approach allows sentences of arbitrary length to be parsed.

The hybrid architecture is able to parse sentences from a simple grammar and disambiguate sentences based on syntactic context. Further work is under way with a large grammar[13] to assess the technique for a realistic subset of natural language.

References

[1] Selman, B. (1985). A rule-based connectionist parsing system. In *Proc. of 7th Annual Conf. of Cognitive Science*. PP 212-221.

[2] Fanty, M. A. (1986). Context-free parsing with connectionist networks. In *Proc. of AIP Conf. Neural Networks for Computers*. PP 140-145.

[3] Rumelhart, D., McClelland J. (1986). On learning the past tenses of English verbs.In *Parallel Distributed Processing, Explorations in the Microstructure of Cognition*, Vol 2. PP 216-271. MIT Press.

[4] Fodor J.A., Pylyshyn Z.W. (1988). Connectionism and Cognitive Architecture: A critical analysis. In *Cognition*, Vol 28. PP 3-71.

[5] Elman J.L. (1990). Finding structure in time. In *Cognitive Science*, 14, PP 179-211.

[6] Pollack J.B. (1990) Recursive Distributed Representations. In *Artificial Intelligence*. Vol 46. PP 77-105.

[7] Smolensky P. (1990). Tensor Product Variable Binding and the Representation of Symbolic Structures in Connectionist Systems. In *Artificial Intelligence*. Vol 46. PP 159-216.

[8] Rumelhart, D.E., Hinton, G.E., Williams R. J. (1986). Learning internal representations by error propagation. In *Parallel Distributed Processing, Explorations in the Microstructure of Cognition, Vol 1: Foundations.* PP 318-362. MIT Press.

[9] Marcus, M. P. (1980). *A Theory of Syntactic Recognition for Natural Language*. Cambridge: MIT Press.

[10] Charniak, E. (1983). A parser with something for everyone. In *Parsing Natural Language*. New York: Academic Press.

[11] Milne, R. (1986). Resolving lexical ambiguity in a deterministic parser. In *Computational Linguistics*, 12, PP 1-12.

[12] Berwick, R. C. (1985). *The Acquisition of Syntactic Knowledge*. Cambridge: MIT Press.

[13] Noble, H. M. (1988). *Natural Language Processing*. Oxford : Blackwell Scientific.

References

[1] Rosenblatt, F. (1958). The perceptron : a probabilistic model for information storage and organisation in the brain. In *Psychological Review.* Part 64. PP 386-408.

[2a] Rumelhart, D.E., Hinton, G.E., Williams R. J. (1986). Learning internal representations by error propagation. In *Parallel Distributed Processing, Explorations in the Microstructure of Cognition, Vol 1: Foundations.* PP 318-362. MIT Press.

[2b] Rumelhart, D., McClelland J. (1986). On learning the past tenses of English verbs. In *Parallel Distributed Processing, Explorations in the Microstructure of Cognition*, Vol 2. PP 216-271. MIT Press.

[3] Kohonen, T. (1990). The self-organising map. In *Proceedings of the IEEE.* Part 78(9). PP 1464-1480.

[4] McClelland, J., Kawamoto, A. (1986). Mechanisms of sentence processing : assigning roles to constituents. In *Parallel Distributed Processing, Explorations in the Microstructure of Cognition*, Vol 2. MIT Press.

[5] Feldman, J. A., Ballard, D. H. (1982). Connectionist models and their properties. In *Cognitive Science 6.* PP 205-254.

[6] Chomsky, N. (1957). *Syntactic Structures.* The Hague:Mouton.

[7] Henry, A. (1995). Why natural language processing needs GB syntax (and vice versa). In *Proceedings of the 4th International Conference on the Cognitive Science of Natural Language Processing.* PP 1-6.

[8] Woods, W. A. (1970). Transition network grammars for natural language analysis. In *Communications of the ACM.* Volume 13. Number 10. PP 591-606.

[9] Fanty, M. A. (1986). Context-free parsing with connectionist networks. In *Proc. of AIP Conf. Neural Networks for Computers.* PP 140-145.

[10] Selman, B., Hirst, G. (1985). A rule-based connectionist parsing system. In *Proc. of 7th Annual Conf. of Cognitive Science.* PP 212-221.

[11] Hanson, S. J., Kegl, J. (1987). PARSNIP: A connectionist network that learns natural language grammar from exposure to natural language sentences. In *Proc. of 9th Annual Conf. of Cognitive Science.* PP 106-119.

[12] Howells, T. (1988). VITAL: A connectionist parser. In *Proc. of 10th Annual Conf. of Cognitive Science.* PP 18-25.

[13] Giles, C.L., Sun, G.Z., Chen, H.H., Lee, Y.C., & Chen, D. (1990). Higher order recurrent networks and grammatical inference. In D.S. Touretzky (Ed.), *Advances in Neural Information Systems 2* . P 380.

[14] Henderson, J. (1994). *Description Based Parsing in a Connectionist Network.* Unpublished PhD Thesis. University of Pennsylvania.

[15] Fodor J.A., Pylyshyn Z.W. (1988). Connectionism and Cognitive Architecture: A critical analysis. In *Cognition*, Vol 28. PP 3-71.

[16] Minsky, M. L., Papert, S. A. (1969). *Perceptrons.* Cambridge, M.A: MIT Press.

[17] Fodor, J. A., McLaughlin, B. (1990). Connectionism and the problems of systematicity: why Smolensky's solution doesn't work. In *Cognition.* Volume 35. PP 183-204.

[18] Van Gelder, T. (1990). Compositionality: a connectionist variation on a classical theme. In *Cognitive Science*, 14, PP 355-384.

[19] Elman J.L. (1990). Finding structure in time. In *Cognitive Science*, 14, PP 179-211.

[20] Pollack J.B. (1990) Recursive Distributed Representations. In *Artificial Intelligence.* Vol 46. PP 77-105.

[21] Smolensky, P. (1990). Connectionism and the foundations of AI. In *Foundations of Artificial Intelligence.* Vol 11. PP 1-74.

[22] Touretzky, D. S., Hinton, G. E. (1988). A distributed connectionist production system. In *Cognitive Science.* Vol 12. PP 423-466.

[23] Miikkulainen, R. (1993). *Subsymbolic Natural Language Processing.* MIT Press.

[24] Cottrel, G. W. (1985). Connectionist parsing. In *Proceedings of the 7th Annual Conference of the Cognitive Science Society.*

[25] Sharkey, N. (1991). Connectionist representation techniques. In *Artificial Intelligence Review.* Volume 5. PP 143-167.

[26] Sharkey, N. E. (1989). The lexical distance model and word priming. In *Proceedings of the Eleventh Cognitive Science Society.*

[27] Minsky, M. L., Papert, S. A. (1988). *Perceptrons, Expanded Edition.* Cambridge, M.A: MIT Press. Original edition, 1969.

[28] Jordan, M. I. (1986). Attractor dynamics and parallelism in a connectionist sequential machine. In *Proceedings of the 8th Annual Conference of the Cognitive Science Society.* PP 531-545.

[29] Williams, R. J., Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. In *Neural Computation 1(2).* PP 270-280.

[30] Das, S., Mozer, M. C. (1992). Connectionist symbol manipulator that induces rewrite rules. In *Connectionist Natural Language Processing.* (Ed. N. Sharkey) Part 5. Intellect. PP 1-7.

[31] Sun, G. Z. (1992). Learning context-free grammar with enhanced pushdown automaton. In *Connectionist Natural Language Processing.* (Ed. N. Sharkey). Part 6. Intellect. PP 1-12.

[32] Pinker, S., Prince, A. (1988). On language and connectionism : Analysis of a parallel distributed processing model of language acquisition. In *Cognition 28.* PP 73-193.

[33] Sun, R. (1992). On variable binding in connectionist networks. In *Connection Science.* Vol 4. No 2. PP 93-124.

[34] Lange, T., Dyer, M. (1989). Dynamic, non-local role bindings and inferencing in a localist network for natural language understanding. In *Advances in Neural Information Processing Systems 1.* Morgan-Kauffman. PP 545-552.

[35] Touretzky, D. S. (1991). BoltzCONS : Dynamic symbol structures in a connectionist network. In *Connectionist Symbol Processing.* PP 5-46. MIT Press.

[36] Kwansy, S. C., Kalman, B. L. (1995). Tail-recursive distributed representations and simple recurrent networks. In *Connection Science.* Vol 7. No. 1. PP 61-80.

[37] Plate, T. (1995). Holographic reduced representations. In *IEEE Transactions on Neural Networks*. Vol 6. No. 3. PP 623-641.

[38] Small, S. (1981). *Word Expert Parsing : A Theory of Distributed Word-Based Natural Language Understanding*. Unpublished Ph.D Thesis, University of Maryland.

[39] Hinton, G. E. (1991). Mapping part-whole hierarchies into connectionist networks. In *Connectionist Symbol Processing*. PP 47-75. MIT Press.

[40] Wermter, S. (1995). *Hybrid Connectionist Natural Language Processing*. Chapman And Hall, International Thomson Computer Press, London.

[41] Smolensky P. (1990). Tensor Product Variable Binding and the Representation of Symbolic Structures in Connectionist Systems. In *Artificial Intelligence*. Vol 46. PP 159-216.

[42] Sharkey, N. E. (1989). Fast connectionist learning: words and case. In *Artificial Intelligence Review*. Vol 3. PP 33-47.

[43] Shastri, L., Ajjanagadde, V. (1993). From simple associations to systematic reasoning: A connectionist representation of rules, variables and dynamic bindings using temporal synchrony. In *Behavioral and Brain Sciences*. Vol 16.

[44] Chalmers, D. J. (1990). Syntactic transformations on distributed representations. In *Connection Science*. Vol 2.

[45] Chomsky, N. (1965). *Aspects of the Theory of Syntax*. MIT Press. Cambridge. Massachusetts.

[46] Grishman, R. (1986). *Computational Linguistics: an introduction.* Cambridge University Press.

[47] Gazdar, G. (1983). Phrase structure grammars and natural languages. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence*. PP 556-565.

[48] Woods, W. A. (1973). Progress in natural language understanding - An application to lunar geology. In *Proceedings of the AFIPS Conference 42*. PP 441-450.

[49] Winograd, T. (1983). *Language as a Cognitive Process*. Addison-Wesley, Reading, MA.

[50] Bates, M. (1978). The theory and practice of augmented transition network grammars. In *Natural Language Communication with Computers*. Ed. L. Bolc. Berlin:Springer-Verlag. PP 191-260.

[51] Marcus, M. P. (1980). *A Theory of Syntactic Recognition for Natural Language*. Cambridge: MIT Press.

[52] Charniak, E. (1983). A parser with something for everyone. In *Parsing Natural Language*. New York: Academic Press.

[53] Milne, R. (1986). Resolving lexical ambiguity in a deterministic parser. In *Computational Linguistics*, 12, PP 1-12.

[54] Berwick, R. C. (1985). *The Acquisition of Syntactic Knowledge*. Cambridge: MIT Press.

[55] Kwansy, S. C., Faisal, K. (1993). Connectionism and determinism in a syntactic parser. In *Proceedings of IEEE Conference on Connectionist Natural Language Processing.*

[56] Wirth, N. (1971). The programming language Pascal. In *Acta Informatica*. Vol 1(1). PP 35-63.

[57] Kernighan, B. W., Richie, D. M. (1978). *The C Programming Language*. Prentice-Hall: Englewood Cliffs, NJ.

[58] Eaton, A., Olivier, T. (1992). Learning coefficient dependence on training set size. In *Neural Networks*. Vol. 5. PP 283-288.

[59] Minai, A. A., Williams, R. D. (1990). Acceleration of backpropagation through learning rate and momentum adaption. In *Int. Joint Conf. on Neural Networks*. PP 676-679.

[60] Sharkey, A.J.C., Sharkey, N. E. (1992). Connectionism and natural language. In *Connectionist Natural Language Processing*. (Ed. N. Sharkey). Part 20. Intellect. PP 1-10.

[61] Lawrence, S., Giles, C. L. (1995). *On the Applicability of Neural Network and Machine Learning Methodoligies to Natural Language Processing*. Technical Report UMIACS-TR-95-64 and CS-TR-3479. NEC Research Institute, Princeton, NJ.

[62] Reilly, R. (1992). Connectionist technique for on-line parsing. In *Neural Networks 3*. PP 37-46.

[63] Berg, G. (1992). A connectionist parser with recursive sentence structure and lexical disambiguation. In *Learning:Constructive and Linguistic*. PP 32-37.

[64] Quirk, R., Greenbaum, S., Leech, G., Svartvik, J. (1985). *A Comprehensive Grammar of the English Language*. Longman.

[65] Noble, H. M. (1988). *Natural Language Processing*. Oxford : Blackwell.

[66] Giles, C. L., Omlin, C. W. (1993). Extraction, insertion, and refinement of symbolic rules in dynamically driven recurrent networks. In *Connection Science*. Vol 5. Part 1 and 4. PP 307-328.

[67] Giles, C. L., Miller, C. B., Chen, D., Chen, H., Sun, G. Z., Lee, Y. C. (1992). Learning and extracting finite state automata with second-order recurrent neural networks. In *Neural Computation*. Vol 4. PP 393-405.

[68] Rosenblatt, F. (1959). Two theorems of statistical seperability in the Perceptron. In *Mechanization of Thought Processes: Proceedings of a Symposium Held at the National Physical Laboratory, November 1958*. London: HM Stationary Office. PP 421-456.

[69] Rosenblatt, F. (1962). *Principles of Neurodynamics*. New York: Spartan.

[70] Widrow, B., Hoff, M. E. (1960). Adaptive switching circuits. In *1960 IRE WESCON Convention Record*. New York. PP 96-104.

[71] Hinton, G. E. (1981). Implementing semantic networks in parallel hardware. In *Parallel Models of Associative Memory*. Ed. G. Hinton & J. Anderson. Hillsdale, N.J. : Lawrence Erlbaum.

[72] Watrous, R. L., Kuhn, G. M. (1991). Induction of finite-state languages using second-order recurrent networks. In *Neural Computation 4*. PP 406-414.

[72] Everitt, B. (1980). *Cluster Analysis*. Heinemann Educational Books. Halstead Press.

[73] Masters, T. (1993). *Practical Neural Network Recipes in C++*. Academic Press, Inc. Harcourt Brace Jovanovich Publishers.

[74] Elman, J. L. (1991). Distributed representations, simple recurrent networks, and grammatical structure. In *Machine Learning 7*. PP 195-225.

[75] Aho, A. V. (1968). Indexed grammars : an extension to context-free grammars. In *Journal of the Association for Computing Machinery*. 15(4). PP 647-671.

[76] Aho, A. V. (1969). Nested Stack Automata. In *Journal of the Association for Computing Machinery*. 16(3). PP 383-406.

[77] Baum, L. E. (1972). An inequality and associated maximisation technique in statistical estimation of probabilistic functions of Markov processes. In *Inequalities*. Vol 3. PP 1-8.

[78] Lee, K. (1989). Hidden Markov models : past, present and future. In *Proc. Of Eurospeech 89*. PP 148-155.

[80] Chomsky, N. (1982). *Some Concepts and Consequences of the Theory of Government and Binding*. Cambridge, Mass. : MIT Press.

[81] Chomsky, N. (1986). *Barriers*. Cambridge, Mass. : MIT Press.

[82] Pollard, C., Sag, I. (1994). *Head-Driven Phrase Structure Grammar*. CSLI Series, Stanford. The University of Chicago Press.

[83] Sells, P. (1985). *Lectures on Contemporary Syntactic Theories*. CSLI Series, Stanford.

[84] Bresnan, J. (1978). A realistic transformational grammar. In *Linguistic Theory and Psychological Reality*. PP 1-59. Cambridge, Mass. : MIT Press.

[85] Bresnan, J. (1982). *The Mental Representation of Grammatical Relations*. Cambridge, Mass. : MIT Press.

[86] Mel'cuk, I. (1988). *Dependency Syntax: Theory and Practice*. State University of New York Press, Albany, New York.

[87] Schabes, Y. (1990). *Mathematical and Computational Aspects of Lexicalised Grammars*. Unpublished PhD thesis, University of Pennsylvania, Department of Computer Science.

[88] Berwick, R., Fong, S. (1995). A quarter century of computation with transformational grammar. In *Linguistics and Computation*. CLSI Lecture Notes No. 52. CLSI Publications. PP 103-143.

[89] Seneff, S. (1992). TINA : A natural language system for spoken language applications. In *Computational Linguistics*. PP 61-86.

[90] Wood, M. M. (1993). *Categorial Grammar*. Routledge.

[91] Jacobs, P., Rau, L. (1993). Innovations in text interpretation. In *Artificial Intelligence*. 63(1-2). PP 143-191.

[92] Fillmore, C. R. (1968). The case for case. In *Universals in Linguistic Theory*. Bach, E., Harms, R. (Eds). Holt, Rinehart and Winston.

[93] Jackson, E., Appelt, D., Bear, J., Moore, R., Podlozny, A. (1991). A template matcher for robust natural-language interpretation. In *Proceedings of the Fourth DARPA Speech and Natural Language Workshop*. Pacific Grove, California. Morgan Kaufmann. PP 190-194.

[94] Sampson, G., Haigh, R., Atwell, E. (1989). Natural language analysis by stochastic optimisation : a progress report on project APRIL. In *Journal of Experimental and Theoretical Artificial Intelligence*. PP 127-287.

[95] Keenan, F. (1994). *Large Vocabulary Syntactic Analysis for Text Recognition*. Unpublished Ph.D Thesis. The Nottingham Trent University, Department of Computing.

[96] Briscoe, E., Carroll, J. (1993). Generalised probabilistic LR Parsing for unification-based grammars. In *Computational Linguistics*. PP 25-60.

[97] Hindle, D. (1993). A parser for text corpora. In *Computational Approaches to the Lexicon*. Atkins, B., Zampolli, A. (Eds). Oxford University Press.

[98] Magerman, D. M., Marcus, M. P. (1991). Pearl : A probabilistic chart parser. In *Proceedings of the Fourth DARPA 9Speech and Natural Language Workshop*. Pacific Grove, California. Morgan Kaufmann.

[99] Hindle, D., Rooth, M. (1991). Structural ambiguity and lexical relations. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*. Berkeley, California. PP 229-236.

[100] Ades, A., Steedman, M. (1982). On the order of words. In *Linguistics and Philosophy*. PP 517-558.

[101] Fong, S. (1992). The computational implementation of principle-based parsers. In *Principle-based Parsing: Computation and Psycholinguistics*. Berwick, R., Abney, S., Tenny, C. (Eds). Kluwer, Dordrecht, The Netherlands.

[102] Jones, G., Lloyd-Thomas, H., Wright, J. H. (1993). Adaptive statistical and grammar models of language for application to speech recognition. In *IEE Colloquium on 'Grammatical Inference : Theory, Applications and Alternatives'*. Bristol University, Centre for Commun. Res. PP 25/1-8.

[103] Sampson, G. (1992). Probabilistic parsing. In *Directions in Corpus Linguistics*. Mouton de Gruyter. PP 419-447.

[104] Wright, J. H. (1990). LR Parsing of probabilistic grammars with input uncertainty for speech recognition. In *Computer Speech and Language*. PP 297-323.

[105] Shieber, S. M. (1983). Sentence disambiguation by a shift-reduce parsing technique. In *Computer Speech and Language*. PP 297-323.

[106] Shieber, S. M. (1992). *Constraint-Based Grammar Formalisms*. MIT Press, Cambridge, Massachusetts.

[107] Slobin, D. (1966). Grammatical transformations and sentence comprehension in childhood and adulthood. In *Journal of Verbal Learning and Verbal Behaviour*. PP 219-227.

[108] Wilks, Y. (1975). A preferential pattern-seeking semantics for natural language inference. In *Artificial Intelligence (6)*. PP 53-74.

[109] Wang, J. (1992). Syntactic preferences for robust parsing with semantic preferences. In *Proceedings of COLING-92*. PP 239-245.

[110] Jones, B. (1995). Is there a role for syntax/parsing in NLP. In *4th International Conference on the Cognitive Science of Natural Language Processing*. Dublin City University. PP 219-227.

[111] Stevenson, S. (1994). *A Competitive Attachment Model for Resolving Syntactic Ambiguities in Natural Language Parsing*. Unpublished Ph.D thesis, The University of Maryland.

[112] Kimball, J. (1973). Seven principles of surface structure parsing in natural language. In *Cognition*. (2:1). PP 15-47.

[113] Fodor, J., Bever, T., Garrett, M. (1974). *The Psychology of Language*. New York : McGraw-Hill.

[114] Frazier, L., Fodor, J. (1978). The sausage machine : A new two-stage parsing model. In *Cognition 6*. PP 291-325.

[115] Frazier, L., Clifton, C., Randall, J. (1983). Filling gaps : Decision principles and structure in sentence comprehension. In *Cognition 13*. PP 187-222.

[116] Frazier, L. (1987). Sentence processing : Evidence from Dutch. In *Natural Language and Linguistic Theory 5*. PP 519-559.

[117] Gorrell, P. (1987). *Studies of Human Syntactic Processing : Ranked-Parallel versus Serial Models*. Unpublished Ph.D Thesis, University of Connecticut.

[118] McRoy, S., Hurst, G. (1990). Race-based parsing and syntactic disambiguation. In *Cognitive Science*. (14). PP 313-353.

[119] Ford, M., Bresnan, J., Kaplan, R. (1982). A competence-based theory of syntactic closure. In *The Mental Representation of Grammatical Relations*. Cambridge, MIT Press.

[120] Fodor, J., Inoue, A. (1994). The diagnosis and cure of garden paths. In *Journal of Psycholinguistic Research*. (23:5). PP 407-434.

[121] Kaplan, R. (1972). Augmented transition networks as psychological models of sentence comprehension. In *Artificial Intelligence 3*. PP 77-100.

[122] Hindle, D. (1983). Deterministic parsing of syntactic non-fluencies. In *Proc. of the 21st Annual Meeting of the ACL*. Cambridge, MA.

[123] Marcus, M., Hindle, D., Fleck, M. (1983). D-theory : Talking about talking about trees. In *Proc. of the 21st Annual Meeting of the ACL*. Cambridge, MA.

[124] Gorrell, P. (1994). *Syntax and Parsing*. Cambridge University Press, Carmbridge.

[125] Gibson, E. (1991). *A Computational Theory of Human Linguistic Processing: Memory Limitations and Processing Breakdown*. Unpublished Ph.D Thesis, Carnegie-Mellon University.

[126] Sharkey, N. E., Sharkey, A. J. C. (1992). A modular design for connectionist parsing. In *Twente Workshop on Language Technology 3 : Connectionism and Natural Language Processing*. University of Twente, Enschede, The Netherlands. PP 87-96.

[127] Miikkulainen, R. (1995). Subsymbolic parsing of embedded structures. In *Computational Architectures Integrating Neural and Symbolic Processes*. (Ed. R. Sun). Kluwer Academic Publishers, Boston.

[128] Waltz, D., Pollack, J. (1985). Massively parallel parsing : A strongly interactive model of natural language interpretation. In *Cognitive Science*. Vol 9. PP 51-74.

[129] Reggia, J. (1987). Properties of competition-based activation mechanism in neuromimetic network models. In *Proc. of the First International Conference on Neural Networks*. San Diego. PP 131-138.

[130] Apolloni, B. (1992). Learning to solve PP-attachment ambiguities in natural language processing through neural networks. In *IEEE Transactions on Neural Networks*. PP 199-205.

[131] Archambault, D., Bassano, J. (1994). A neural network for supervised learning of natural language grammar. In *IEEE Transactions on Neural Networks*. PP 267-273.

[132] Ghahramani, Z., Allen, R. (1991). Temporal processing with connectionist networks. In *IEEE Transactions on Neural Networks*. PP 541-546.

[133] Frasconi, P., Gori, M., Maggini, M., Soda, G. (1995). Unified integration of explicit rules and learning by example in recurrent networks. In *IEEE Transactions on Knowledge and Data Engineering*. Vol 7. No. 2. PP 340-346.

[133] Lin, T., Horne, B. G., Tine, P., Giles, C. L. (1996). Learning long-term dependencies is not as difficult with narx recurrent neural networks. In *Advances in Neural Information Processing Systems 8*. MIT Press, Cambrdige.

[134] Mozer, M. (1994). Neural net architectures for temporal sequence processing. In *Time Series Prediction*. (Ed. A. S. Weigend and N. A. Gershenfeld). Addison-Wesley. PP 243-264.

[135] Tsoi, A. C., Back, A. (1994). Locally recurrent globally feedforward networks, a critical review of architectures. In *IEEE Transactions on Neural Networks.* Vol 5. No 2. PP 229-239.

[136] Narendra, K. S., Parthasarathy, K. (1990). Identification and control of dynamical systems using neural networks. In *IEEE Transactions on Neural Networks.* Vol 1. No 4.

[137] Cleeremans, A., Servan-Schreiber, D., McClelland, J. (1989). Finite state automata and simple recurrent networks. In *Neural Computation.* Vol 1. No 3. PP 372-381.

[138] Moisl, H. (1992). Connectionist finite state language processing. In *Connection Science.* Vol 4. No 2. PP 67-91.

[139] Miikkulainen, R. (1990). A PDP architecture for processing sentences with relative clauses. In *Proc. of the 13th International Conference on Computational Linguistics.* PP 201-206.

[140] Cook, W. A. (1989). *Case Grammar Theory.* Georgetown University Press, Washington, DC.

[141] Sampson, G. (1987). Evidence against the 'grammatical/ungrammatical' distinction. In *Corpus Linguistics and Beyond.* Amsterdam : Rodopi. PP 219-226.

[142] Garside, R. G., Leech, G. N., Sampson, G. (1987). *The Computational Analysis of English : A corpus-based approach.* London : Longman.

[143] Atwell, E. S. (1987). Constituent-likelihood Grammar. In *The Computational Analysis of English : A corpus-based approach.* (Ed. R. G. Garside, G. N. Leech, G. Sampson).London : Longman.

[144] Francis, W. N., Ku era, H. (1979). *Manual of Information to Accompany a Standard Corpus of Present-day Edited American English.* Brown University Linguistics Department.

[145] Marcus, M., Santorini, B., Marcinkiewicz, M. (1998). *Building a Large Annotated Corpus of English : The Penn Treebank.* Technical report, University of Pennsylvania.

[146] Johansson, S., Leech, G., Goodluck, H. (1978). *Manual of Information to Accompany the Lancaster-Oslo/Bergen Corpus of British English.* Department of English, University of Oslo. Also see ICAME Journal 16, PP 124.

[147] Garside, R., Leech, G., Varadi, T. (1987). *Manual of Information to Accompany the Lancaster Parsed Corpus.* Department of English, University of Oslo. Also see [142].

[148] Garside, R. (1987). The CLAWS word-tagging system. In *The Computational Analysis of English : A corpus-based approach.* (Ed. R. G. Garside, G. N. Leech, G. Sampson).London : Longman.

[149] Atwell, E. S., Pocock, A. (1993). HMM Vs ANN Corpus-based Parsing. In Souter, C & Atwell E (editors), *Corpus-based computational linguistics: proceedings of the 12th conference of the International Computer Archive of Modern English,* Amsterdam, Rodopi, 1993.

[150] Schabes, Y., Roth, M., Osborne, R. (1993). Parsing the Wall Street Journal with the inside-outside algorithm. In *Proceedings of the 6th Conference of the European Chapter of the Association for Computational Linguistics.* Utrecht University, The Netherlands.

[151] Brill, E., Magerman, D., Santorini, B. (1990). Deducing linguistic structure from the statistics of large corpora. In *Proceedings of the DARPA Speech and Natural Language Workshop.* Morgan-Kaufmann.

[152] Elman, J. (1995). Language as a dynamical system. In *Mind as Motion : Explorations in the Dynamics of Cognition.* (Ed., R. F. Port., T. van Gelder). Cambridge, MA : MIT Press. PP 195-223.

[153] Tepper, J. A., Powell, H., Palmer-Brown, D. (1995). Ambiguity resolution in a connectionist parser. In *Proceedings of the 4th International Conference on the Cognitive Science of Natural Language Processing.* Dublin City University, July 1995.

[154] Tepper, J. A., Powell, H., Palmer-Brown, D. (1995). Integrating symbolic and subsymbolic architectures for parsing arithmetic expressions and natural language sentences. In *Proceedings of the 3rd SNN Neural Network Symposium.* Nijmegen University, September 1995.

[155] Pereira, F. (1992). Inside-outside reestimation from partially bracketed corpora. In *Proceedings of the ACL '92.*

[156] Fausett, L. (1994). *Fundamentals of Neural Networks.* Prentice-Hall International Editions. ISBN 0-13-042250-9.

[157] Thornton, C. (1992). *Techniques In Computational Learning.* Chapman & Hall Computing. ISBN 0-412-40430-3.

[158] Dyer, M. (1995). Connectionist Natural Language Processing : A Status Report. In *Computational Architectures Integrating Neural and Symbolic Processes.* (Ed. R. Sun). Kluwer Academic Publishers, Boston.

[159] Callan, R., Palmer-Brown, D. (1997). (S)RAAM : An analytical technique for fast and reliable derivation of connectionist symbol structure representations. In *Connection Science.* Vol 9. PP 139-159.

[160] Hertz, J., Krogh, A., Palmer, R. G. (1991). *Introduction To The Theory of Neural Computation.* Redwood City, CA: Addison-Wesley.

[161] Lane, P., Henderson, J. (1998). Simple Synchrony Networks : Learning to parse natural language with temporal synchrony variable binding. In *Proceedings of ICANN 1998.* Skovde, Sweden.

[162] Henderson, J., Lane, P. (1998). A connectionist architecture for learning to parse. In *Proceedings of the Association of Computational Linguistics.*

[163] Wermter, S., Weber, V. (1994). Learning fault-tolerant speech parsing with SCREEN. In *Proceedings of the Twelfth National Conference on Artificial* Intelligence, Seattle.

[164] Charniak, E. (1997). Statistical parsing with a context-free grammar and word statistics. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence.* AAAI Press/MIT Press.

[165] Charniak, E. (1998). Statistical techniques for natural language parsing. In *AI Magazine.*

[166] Jain, A., Waibel, A. (1990). Robust connectionist parser of spoken language. In *Proceedings of IEEE Conference on Acoustics, Speech and Signal Processing.* School of Computer Science, Carnegie Mellon University, Pitt. PA, USA.

[167] Jain, A. (1991). *A Connectionist Learning Architecture for Parsing Spoken Language.* Unpublished Ph.D Thesis, Carnegie Mellon University.

[168] Jain, A. (1992). Generalization performance in Parsec - a structured connectionist parsing architecture. In *Advances in Neural Information Processing Systems 4.* Morgan Kaufmann.

[169] Buo, F. D., Polzin, T. S., Waibel, A. (1994). Learning complex output representations in connectionist parsing of spoken language. In *Proceedings of ICASSP 94.*

[170] Buo, F. D., Waibel, A. (1996). Search in a learnable spoken language parser. In *The Twelfth European Conference on Artificial Intelligence.* John Wiley & Sons.

[171] Buo, F. D., Waibel, A. (1996). FeasPar - a feature structure parser learning to parse spoken language. In *Proceedings of the COLING 96*. Kopenhagen.

[172] Waibel, A., Jain, A., McNair, A., Tebelskis, J., Osterholtz, L., Saito, H., Schmidbauer, O., Sloboda, T., Woszczyna, M. (1992). JANUS: Speech-to-speech translation using connectionist and non-connectionist techniques. In Moody, J. E., Hanson, S. J., Lipmann, R. P., eds., *Advances in Neural Information Processing Systems 4*. San Mateo, CA: Morgan Kaufmann. PP 188-192.

[173] Brill, E. (1993). *A Corpus-Based Approach to Language Learning*. Unpublished Ph.D Thesis, University of Pennsylvania.

[174] Brill, E., Resnik, P. (1994). A rule-based approach to prepositional phrase attachment disambiguation. In *Proceedings of COLING '94*.

[175] Brill, E., Florian, R., Henderson, J. C., Mangu, L. (1998). Beyond N-Grams : can linguistic sophistication improve language modeling? In *Proceedings of ACL '98*.

[176] de Marcken, C. G. (1990). Parsing the LOB corpus. In *Proceedings of the ACL '90*.

[177] Wyard, P., Nightingale, C. (1992). Grammar recognition by a single layer higher order neural net. In *Journal of BT Technology*. Vol 10. No. 3.

[178] BNC Consortium. (1995). *British National Corpus*. Oxford University Press. February 1995.