Simple Synchrony Networks : A New Connectionist Architecture Applied to Natural Language Parsing

Submitted by Peter Lane to the University of Exeter as a thesis for the degree of Doctor of Philosophy in April, 2000.

Abstract:

This thesis develops a new implementation of Temporal Synchrony Variable Binding (TSVB) using standard sigmoid activation units. TSVB is an extension to the standard connectionist framework allowing a network to represent multiple entities by dividing time periods into separate phases. The importance of TSVB is that TSVB units support the use of structured output representations. A range of TSVB networks is created by adding these new TSVB units to the architecture of a Simple Recurrent Network and developing a novel extension of the Backpropagation Through Time training algorithm.

This thesis also develops two ideas for achieving the most effective learning from TSVB networks. The first idea is a restriction to the space of possible TSVB network architectures. Specifically, TSVB networks contain two types of unit, pulsing and non-pulsing. The most effective networks, known as Simple Synchrony Networks (SSNs), do not have links from pulsing to non-pulsing units. The second idea removes a computational inefficiency in the basic definition of TSVB networks by only computing the phases in a bounded queue, known as the short-term memory (STM). Mechanisms for managing the STM retain phases of use to the current problem, and remove those that are irrelevant.

Experiments on two toy grammars test a range of TSVB networks for specific generalisations across entities and constituent structure. Additionally, a number of SSNs (with a STM) are tested on a more demanding application: the SSN is trained to output parse trees in response to samples of natural language taken from newspaper reports. Through its use of appropriate generalisations and a general learning algorithm, the SSN achieves good results in a field previously dominated by specialised algorithms. This makes the SSN the first serious connectionist proposal for an alternative to standard statistical approaches to natural language parsing.

Acknowledgements:

Grateful acknowledgements are due to my supervisor, Jamie Henderson, and my thesis committee, Derek Partridge and Antony Galton, for helping me to complete this thesis. Also thanks to my examiners, Stefan Wermter and Ajit Narayanan, for a number of suggestions which improved its presentation and quality.

Thanks are also due to the Economic and Social Research Council (UK) as sponsor and the University of Sussex as grantholder for providing the SUSANNE corpus used in the experiments on natural language parsing in this thesis.

This research has been funded by the Engineering and Physical Sciences Research Council, UK.

Contents

Chapter 1

Introduction

This thesis describes the development and empirical validation of a new machine learning algorithm. This new algorithm, the Simple Synchrony Network, has been developed by combining two earlier extensions to the basic connectionist network. Although the basic connectionist network has proved an excellent algorithm for learning associations between input and output patterns over a wide range of domains, various extensions have been proposed in order to extend the range of domains to which such networks can be applied. The two extensions used here are the Simple Recurrent Network, which learns about patterns across time, and Temporal Synchrony Variable Binding, which provides a representation for multiple entities and generalisations across them.

The theoretical importance of the Simple Synchrony Network (SSN) lies in its combination of these two earlier extensions, which makes the SSN an inherently powerful learning algorithm. Further, the SSN is a novel implementation of Temporal Synchrony Variable Binding (TSVB) which produces trainable networks but using fully general, distributed representations. These theoretical ideas are tested in a series of experiments training the SSN to parse samples of natural language, culminating in a set of experiments using a sample of real text. These latter experiments are significant in the connectionist learning literature for two reasons: firstly, because they use a corpus of naturally occurring text (a subset of the Brown corpus); and secondly, because the SSN incrementally outputs a parse tree representation for its input sentence, and consequently may be evaluated in the same terms as parsers from the statistical language community.

This introduction chapter begins with a review of some basic questions in machine learning relating to representations and learning procedures, discussed from the perspective of comparing such algorithms. This attempts to show that requiring a good result and requiring a general approach are different criteria, and lead to different algorithms. These ideas are then applied to the problem of learning language, where shortcomings in the ability of connectionist networks are brought out through a comparison with statistical approaches. This discussion places the SSN in context as a machine learning algorithm and motivates some of the ideas behind its development. Next, this chapter summarises the main findings and results of this thesis, and ends with an overview of the following chapters.

1.1 Comparing Machine Learning Algorithms

Machine learning is an area of research falling within Artificial Intelligence (AI). Practitioners may have a number of motivations for pursuing a specific project, but essentially research in AI aims to uncover the general principles upon which any form of intelligent behaviour on given tasks must be based. Winston [105] (Chapter 1) describes two central goals of AI research:

- One central goal of Artificial Intelligence is to make computers more useful.

- Another central goal is to understand the principles that make intelligence possible.

The first of these may be termed a *results-based* goal, and the second a *principles-based* goal. Evaluating an algorithm with respect to the former entails applying a suitable measure of 'usefulness'; the algorithm with the highest score will be the one to use. Understanding principles, however, requires a more qualitative evaluation. For example, given a new algorithm which learns to parse samples of natural language, a results-based evaluation would compare the generalisation performance of the new algorithm with that of a previous one, and draw conclusions based on this performance measure. Instead, a principles-based evaluation would consider the class of applications to which the algorithm may be subsequently applied with a similar degree of success; the wider the class, the more general the principles captured by the algorithm. In order to determine which of these evaluations is more important, the *motivation* for the algorithm's design and use must be determined: is it intended to be a specialist model for language learning, or instead a general machine learning algorithm which can be effectively applied to language learning?

The field of machine learning is, to a large extent, concerned with the question of 'performance improvement'; a learning algorithm is given a representative sample of training data from some domain, and its performance is evaluated on some previously unseen test data. From this definition it is apparent that the primary evaluation of a machine learning algorithm is a results-based one. However, it is also possible to ask of a learning algorithm, which has proven successful on a representative sample of tasks of a specific *type*, whether it performs well on further tasks taken from that type. For example, given an algorithm which has learnt to play a board game, e.g. checkers [88], how well does it learn to play other games of that type, e.g. chess? Such an idea is behind the Metagame challenge of Pell [76], where algorithms are tested against one another on novel games from well-defined classes. The performance measure employed in this case is a principles-based evaluation, and is used to separate the high-powered specialist algorithms from those of a more general capability.

It is the interplay of these two evaluation measures which drives much of the variety found in machine learning algorithms. There are three broad classes of such algorithms. The first class is described as *statistical learning*, in which results on specific applications are the sole criteria for evaluating an algorithm. The other two classes strive more for general principles applicable to many domains, and are the algorithms known more widely under the heading of *machine learning* algorithms. These last two classes are distinguished based on the internal representation used for knowledge acquired from the training data: the *symbolic*, or high-level, and the *connectionist*, or low-level, classes of algorithm. (However, this division is fairly arbitrary, as all three classes may be considered to be examples of *machine* learning. Here the division is based on whether the representational space is given to the algorithm or built up by it; this accords with the treatment in Mitchell [73], though Kubat, Bratko and Mitchell [52] treat neural networks and genetic algorithms as non-AI algorithms, and therefore in a class of their own.)

For the class of statistical learning algorithms, the first step in the learning process is to construct a set of *predictor variables*. This set constitutes the *model* of the domain, and is constructed so as to maximise the performance of the learner. For example, consider an algorithm learning to determine whether an instance x is a member of a class C. If the probability that $x \in C$ depends on x^2 then, the statistician would argue, x^2 should be in the predictor set, i.e. the set of variables used to predict the output class.

> "Relying on the system to work out that x^2 should be included is unlikely to lead to as good a rule; after all, the system has to separate this truth from the superfluous random variation arising in the particular [training] set you have given it." (Hand [34], Chapter 11)

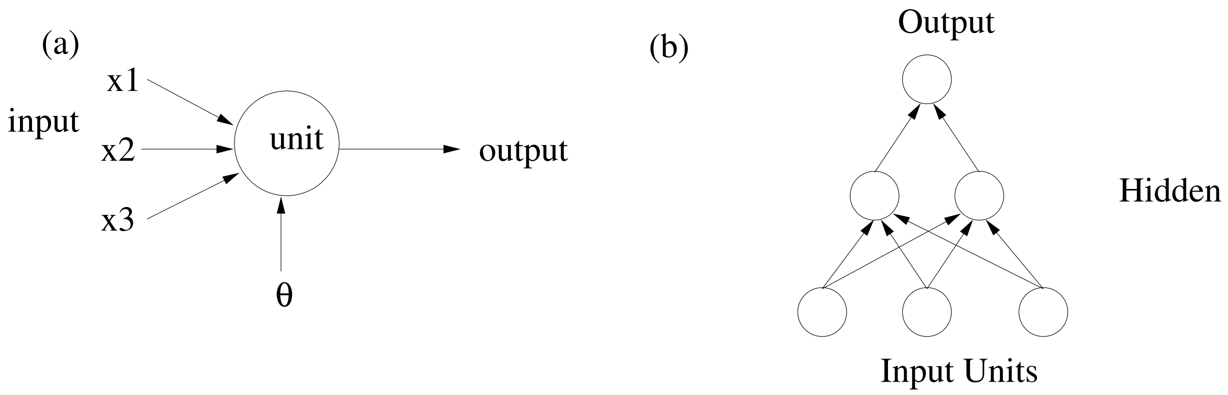


Figure 1.1: The standard connectionist unit, and a feed-forward network.

Having constructed the predictor set, the learning process proceeds by working out the statistics behind each variable's contribution to the class of a given instance. These statistics are gathered directly from the training data, which is therefore assumed to be a representative sample of the data to be modelled. Part of the variety in statistical learning algorithms derives from the different assumptions made about the interactions of the different predictor variables, e.g. the particular probability distribution within the sample followed by the variable.

The second and third classes of machine learning algorithm differ from the statistical approach in assuming that a common *representation* for knowledge will be applicable to a wide range of possible applications. The algorithm will then use its specific learning *procedure* to construct, within that representation, knowledge of how to predict the output class of a given instance. The emphasis is therefore on a general procedure for constructing the predictor variables and statistics used in the statistical approach. The intention is that a powerful enough procedure will indeed manage to separate the correct knowledge from the training data over a broad class of domains. The two classes of algorithm differ because they use a different level of representation in which to represent the knowledge gained from the training data. Essentially, the second class uses a propositional or symbolic level of representation, and the third class uses connectionist (or neural) networks. (Other forms of representation exist, some closer to connectionist representations, some closer to propositional, but these two are used for expository purposes, as they are at opposite ends of the representational spectrum.)

The high-level, symbolic representation of knowledge is characterised by the use of propositional logic. For example, the proposition

$$\exists x.\exists y.\text{red}(x) \wedge \text{blue}(y) \wedge \text{square}(x) \wedge \text{circle}(y)$$

can be used to represent a scene with two objects or entities – a red square and a blue circle. The features present in the scene are named, and the entity to which they relate are indicated by variables. The presence of the variables 'x' and 'y' enables the representation to unambiguously combine the descriptions of two or more separate entities.

In direct contrast to this is the low-level, connectionist representation of knowledge, in which it is assumed that learned information can be described in terms of numerical values assigned to labelled units and their interconnections. Figure 1.1 illustrates (a) an individual connectionist unit, and (b) a network of such units, where the input units are used to store input patterns, hidden units hold intermediate representations, and the output units hold the computed output. Each unit in the network can be affected by information stored on other units by receiving their activation values via weighted links. Each unit can further affect other units via its output links. With a standard connectionist network, each unit will apply a non-linear threshold function to the weighted sum of its inputs.

Each of these two representations, propositional logic and connectionist networks, offers a flexible and expressive language in which to represent functions mapping certain input data to a desired output. However, because the two representations differ, they also differ on the generalisations from learned information which each make possible for a learning system confronted with a novel item of input data. For instance, in the symbolic system, it is quite straightforward to imagine that a system which can represent 'a red square and a blue circle' will also be able to represent 'a blue square and a red circle', because the only change to be made is which entity the 'red' or 'blue' variable is predicated on. This ability, which Fodor and Pylyshyn [24] termed *systematicity*, is the ability to work on the form of a representation, as well as its actual values. That is, because the system is currently attributing entity 'x' with the colour value 'red', the system is also able to attribute entity 'x' with any other colour value, such as 'blue'. Therefore, the generalisations which the symbolic system can make to novel items of input data can be based on the structural form of the input and learned data.

In contrast to this, the inherent generalisations afforded by the connectionist representation begin at a much lower-level. Because each unit only computes with numbers, the information input to a connectionist network can be represented in a multi-dimensional space of real numbers. This space can be used to plot the outputs produced by the network. The generalisation behaviour is seen if a test point falls between two input points in this multi-dimensional space. In such cases the network tends to compute its output for the test point as an interpolation of the outputs at the two earlier input points (e.g. see Huang & Lippmann [46]).

Once the representational space for these two classes of algorithm has been defined, next comes the problem of exploring that space, during learning, in order to find an appropriate representation of the training data. The particular procedure used by the algorithm is conditioned, to a large extent, by the particular representation it is working on. However, Langley [59] and Mitchell [73] both argue for similarities across the symbolic and connectionist schemes in this respect. Specifically, both schemes rely on a procedure which explores their representational space for an appropriate mapping between the input and output data used for training. A common method for guiding this process is to measure the error made by the algorithm, which is used to alter the internal representation so as to reduce that error. Thus, for symbolic systems, this hill-climbing approach is applied to the learning of discrimination trees in ID3 [79], and for connectionist systems, the hill-climbing approach is known as gradient descent, as used in the popular backpropagation training algorithm [85].

The discussion so far has focused on the dimension of specialisation–generality, although there are almost as many ways to compare two algorithms as there are domains. For some domains, such as medical expertise, the ability of the algorithm to explain its results to a doctor is important, which makes a higher-level representational space mandatory. In other domains, such as real-time control of instrumentation, the speed and robustness of operation would be more important. However, in all cases where changes are made to an algorithm to enhance performance, it must be remembered that: "small improvements under 'laboratory conditions' may not transfer to real application environments" (Hand [34], Chapter 11). And so any changes proposed to a given algorithm in order for it to achieve good performance in a given domain must be shown to be as general as possible.

1.2 Requirements for Learning Language

This thesis is about a new machine learning algorithm, the Simple Synchrony Network (SSN). The SSN is a new type of connectionist network, formed by combining two specific extensions to the basic connectionist network. These extensions are the Simple Recurrent Network, for learning about patterns across time, and Temporal Synchrony Variable Binding (TSVB), for representing multiple entities. The justification for forming this particular combination is based, partly, on the theoretical goal of implementing a trainable TSVB connectionist network.

The target domain for much of the experimental work in this thesis will be the task of outputting a parse tree representation for a given input sentence. TSVB networks are ideal for such experiments, when compared with other connectionist architectures, because they enable the network to incrementally output the amount of information necessary to build up this parse tree. In addition, the extensions used in the SSN give the network the appropriate abilities to generalise across this structured information. This is best shown by describing, briefly, Probabilistic Context-Free Grammars, a popular statistical algorithm for learning to parse from samples of natural language, where these abilities are most apparent.

1.2.1 Statistical learning

The Probabilistic Context-Free Grammar is one of the more popular statistical algorithms for learning to parse samples of natural language, especially effective where the input sentence is a string of word-tags [11, 48]. The statistical parser is trained by creating a grammatical representation for its training corpus, which is assumed to be a representative sample of the entire dataset. This representation is created by taking each sentence in the training corpus and forming a grammatical rule for generating that sentence. For example, the sentence 'John loves Mary' could be generated by the rule '$\texttt{S} \rightarrow \texttt{N V N}$', with appropriate rules for instantiating nouns (N) and verbs (V). The rules constitute a context-free grammar for the training data. These rules are then augmented by assigning a probability to each rule, based on the number of occurrences of the rule within the training data, thereby generating a Probabilistic Context-Free Grammar (PCFG) representation for the training corpus. In parsing, the grammar is used to assign all possible parses to the input sentence, from which the most probable is selected as the output parse. The output parse is then evaluated against the target parse by comparing the proportion of correct constituents with the totals in the output and target parses (yielding figures known as *precision* and *recall*, akin to the standard measures of errors of commission and omission). Each constituent is counted as correct if it contains the correct set of words and also has the correct constituent label.

The PCFG is a language-specialised example of a statistical algorithm for grammatical inference. The knowledge acquired by the PCFG can be considered under three headings. First, the grammar acquires the relative probabilities of different words and constituents within the entire corpus, i.e. statistical information is acquired about the elements in the representation. Second, the grammar encodes grammatical rules for sequences of words, i.e. sentences. This means the grammar learns about patterns across time. And finally, the grammar is compositional, meaning that information is automatically generalised across syntactic positions. The grammar therefore refers to multiple entities (words and constituents) and makes automatic generalisations across them; this generalisation property was referred to above in the description of symbolic representations, and is known as systematicity [24].

These three requirements are generic to all applications involving the inference of a grammar, and are captured by the *representation* of the induced grammar as a PCFG. The PCFG's specific superiority in language parsing is also in part due to the procedures used for acquiring the probabilities and grammatical information, as well as the selection of the best output parse tree. The challenge for connectionist language learning is to transfer these representational properties to a connectionist network, and rely on general learning procedures (such as backpropagation) to capture enough information from a set of training data to provide an equivalent level of performance. In order to achieve these representational properties, certain extensions are required to the basic connectionist network.

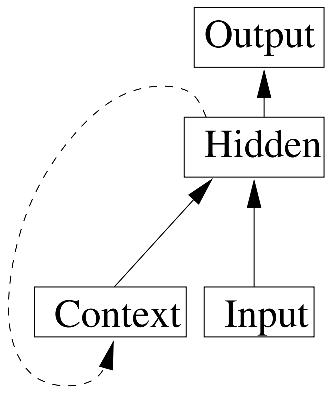


Figure 1.2: The Simple Recurrent Network.

1.2.2 Extending the basic connectionist network

As shown above, the ability to parse natural language requires three abilities: the acquisition of statistics, the ability to learn about patterns across time and the ability to refer to multiple entities and make generalisations across them. The first element, acquiring statistics, is already one of the strengths of the standard connectionist network. Indeed, the connectionist network is a generalised form of regression model, which is a standard statistical tool. The difference between the connectionist and the statistical learning approaches is that in connectionist learning the network is started with small initial weights, which make the hypothesised decision surface simple. As training progresses, some of the weights take on larger values, leading to more complex decision surfaces. This means that the connectionist approach relies on a general procedure (gradient descent) to alter its decision surface in order to learn an appropriate decision surface, whereas the statistical approaches begin with a complex decision surface based on the set of given predictor variables.

In order to learn about patterns across time, the standard extension is to add *context* units. One such network is the Simple Recurrent Network (SRN) [18] illustrated in Figure 1.2. The context units hold a copy of the network's hidden unit activations from the previous time step, therefore including the earlier activation of the network with the current computation. In this way, the output of the network depends both on the current input and on the previous input. These networks can be trained using Backpropagation Through Time [85]. The major benefit of this approach to learning about patterns across time is that the length of the input sequence is not constrained within the architecture or training algorithm. Therefore, information learned about a word in one position of a sentence will automatically be generalised to that same word appearing in any other position; this is a result of the same links being used to process each item in every position of the input sequence.

In Section 3.4 several examples are given showing how these networks have been applied to language learning. However, direct applications of the SRN do not compare well with the statistical parsers; specifically, the output representation of the SRN is not structured, as required to represent a parse tree. The reason for this limitation is that, in each time period, each output unit can only specify one piece of information, i.e. the amount of information output by the network is linear with respect to the number of input words. In order to represent a parse tree, the relationship of each word to each of the preceding words must be shown, which requires a quadratic amount of information with respect to the number of input words.

This shortcoming can be addressed by extending the representational power of standard connectionist units, and this can be done using a technique called Temporal Synchrony Variable Binding (TSVB), introduced by Shastri and Ajjanagadde (S&A) [92]. TSVB enables a connectionist architecture to represent and compute across multiple entities by using the synchrony of activation pulses

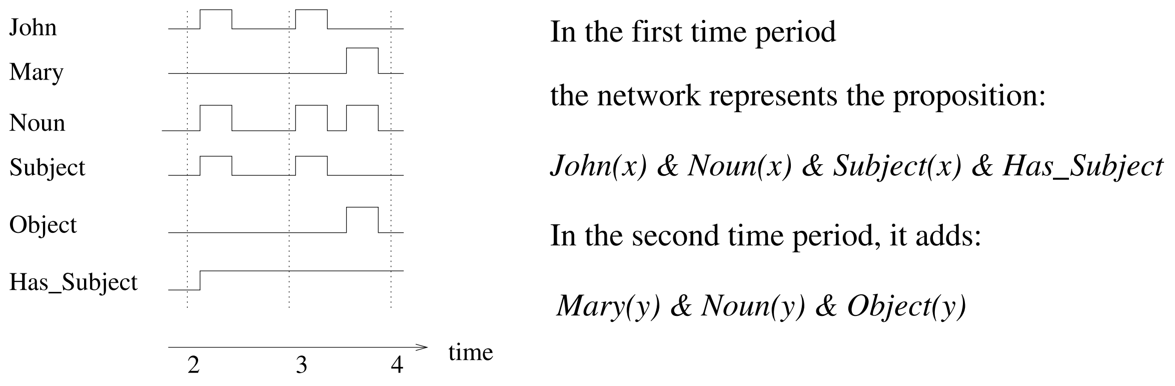

John
Mary
Noun
Subject
Object
Has_Subject
In the first time period
the network represents the proposition:
John(x) & Noun(x) & Subject(x) & Has_Subject
In the second time period, it adds:
Mary(y) & Noun(y) & Object(y)
time
2
3
4

Figure 1.3: An example of variable binding with temporal synchrony.

to represent entities. Within a TSVB network, each time period is divided into a number of phases, and pulsing units represent information about each entity in separate phases. Thus, in each time step the network cycles through the set of entities, computing about each one independently. To communicate information between entities there are also non-pulsing units, which compute across all phases and represent information about the overall context.

Figure 1.3 illustrates the use of TSVB to represent variable bindings. It traces the activation values of some hypothetical units in a network parsing the end of the sentence ‘John loves Mary’. Units that are pulsing synchronously, such as *John* and *Subject*, are representing information about the same entity; this is analogous to the use of identical variables within a logical proposition. The non-pulsing unit *Has_Subject* represents information about the overall sentence, and therefore its activation appears in every phase of the time period.

The use of TSVB in a connectionist network has two important consequences. The first is that the network can output structured information. By representing every constituent in the sentence as a separate phase in each time period, the activation of an output unit can indicate the relationship of the current word with every constituent within the sentence. For example in Figure 1.3, when “loves” is input (the first period shown) the *Subject* unit is activated in the phase previously introduced for “John”, to indicate that “John” is the subject of the verb. The second consequence is that TSVB networks inherently generalise information learned about one entity to other entities. Because different times (i.e. phases) are used to represent different entities, and the same link weights are used at every time, the same learned information is applied to every entity. This argument is used by Henderson [37] to demonstrate that TSVB networks possess inherent systematicity. At the time, this was an ‘in principle’ argument only, as TSVB had only been implemented using binary-threshold units, for which no training algorithm had been devised. Indeed, the nature of binary-threshold units means that their activation function is not differentiable, and therefore standard connectionist training algorithms, such as backpropagation, are not applicable. The representational power of TSVB networks has been demonstrated in SHRUTI, a model of reflexive reasoning [92], and NNEP, a model of syntactic parsing [36]. Each of these models used phases to represent variables and interconnections between nodes to represent logical rules, thereby demonstrating an equivalent representational power to that of a symbolic system using propositional logic.

These considerations lead to a natural prediction: given a *trainable* TSVB network with recurrent links, its representational ability should include that of both standard connectionist networks and the symbolic rules used in SHRUTI and NNEP. Furthermore, the fact that the network is trainable means that two types of generalisation are inherent to its architecture: the first is to generalise across time, due to the recurrent links, and the second is to generalise across multiple entities, as identical weights apply to each phase. Thus, the ability to generalise across structure, typical of symbolic systems, will have been allied to the ability to interpolate between data points, as with standard connectionist networks. It is the aim of this thesis to develop and test such a trainable TSVB network.

1.3 Defining and Testing Simple Synchrony Networks

The contribution of this thesis falls into two parts. First, on the theoretical side, is the integration of the two extensions described above into a trainable connectionist architecture. This integration is achieved by defining Temporal Synchrony Variable Binding (TSVB) units with sigmoid activation functions, and then adding them to the architecture of a Simple Recurrent Network (SRN). The architecture then retains the ability of SRNs to learn about patterns across time, and the TSVB units provide the additional ability to represent multiple entities and generalisations across them.

The second contribution of this thesis is the set of results from experiments on learning to parse samples of real natural language. The main intention here is to test whether the theoretical capabilities of the Simple Synchrony Network really do give it the power to learn in domains such as natural language acquisition which require learning structured representations from real world data.

The major results from this thesis are summarised in [56].

1.3.1 The Simple Synchrony Network

The Simple Synchrony Network (SSN) is defined by reimplementing the central ideas of Temporal Synchrony Variable Binding (TSVB) [92] using standard connectionist units with sigmoid activation functions. The following three principles encapsulate the central ideas of TSVB:

- Each time period is divided into discrete phases, each phase to represent a distinct entity.
- Pulsing units compute within each phase independently of other phases, and so compute information about distinct entities.
- Non-pulsing units compute across all phases, combining information about several entities.

There are a number of possible approaches to implementing these ideas. In this thesis, the basis of the implementation is to introduce temporal synchrony into standard connectionist units. This is achieved by creating two kinds of unit. The first is the *pulsing* unit, which computes in individual phases independent of other phases. Its output activation in each time period is formed from its separate activation in each phase of the specific time period, i.e. the pulsing unit computes activation for every phase in the current time period. The second type of unit is the *non-pulsing* unit, which computes across all phases equally in the current time period; its output activation in a specific time period is constant across every phase in that time period. For each of these units, output activation is computed using the standard sigmoid activation function; for pulsing units, this function is applied to every phase in each time period. Also, in computing the net input to each unit, the same weight is applied to activation passed between units irrespective of phase number; this last fact means that TSVB networks generalise in accordance with systematicity [37].

Training such networks is achieved through a novel extension of Backpropagation Through Time (BPTT) [85]. When training a recurrent network, BPTT begins by unfolding the network over time, making a copy of the network for every time step in the sequence. Extending BPTT to TSVB networks involves making a further copy of the network for every phase in each time period. Both the pulsing and non-pulsing units are copied once per time period and the pulsing units are copied additionally once per phase. As with standard BPTT the unfolded network is a feed-forward

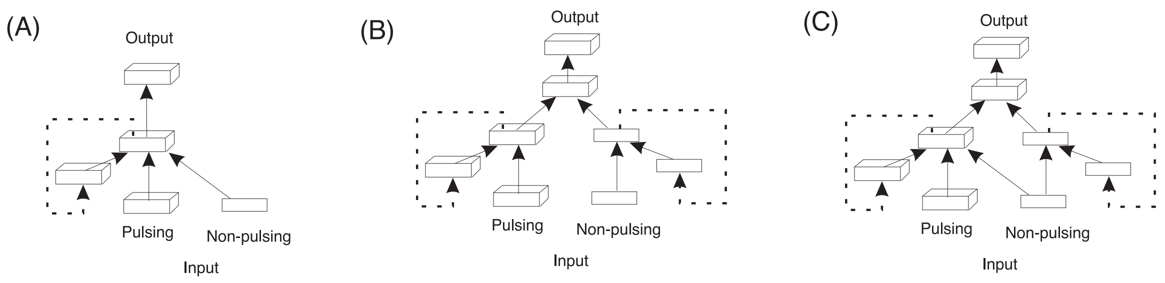


Figure 1.4: Three Simple Synchrony Networks.

network, and can be trained using backpropagation. However, every copy of each link must be updated so as to have the same weight, achieved by summing all the individual changes to each copy of the link.

The SSN itself is formed by restricting the available architectures so that pulsing units can never pass activation to non-pulsing units. This restriction, which was first suggested by experimental evidence, is required because the output of units is never precisely 0 or 1; when a non-pulsing unit sums up its inputs over every phase in the time period, a near-zero input on every phase can accumulate into a significant input and produce an erroneous response. This restriction, however, seems to remove the possibility of an interaction of information computed about separate entities. To overcome this problem, a separate set of input units is required, one set being pulsing input units for inputting information about separate entities, and the second set being non-pulsing input units for inputting information about all the entities. Such networks are illustrated in Figure 1.4; the rectangular layers are composed of non-pulsing units and the block-shaped layers pulsing units.

With such networks, the input can be divided into two parts: one part providing information about separate entities, and one part dealing with the entire situation. In the context of experiments in language learning, however, the only input is that of individual words (or word-tags). For all the experiments in this thesis, the input to the network is arranged as follows. The separate pulsing and non-pulsing sets of input each contain a separate input unit for each word in the training corpus, i.e. both the pulsing and non-pulsing inputs will use a localist representation, with only one input unit being active at any given point in time. Each word in the sentence will introduce a new entity to the network, i.e. it is introduced on the pulsing inputs in a previously unused phase. Simultaneously, the word is also introduced on the non-pulsing inputs. The non-pulsing components of the network do not use phase numbers to separate the entities and therefore compute information about the sentence as a whole, based on every word in the sentence. The pulsing components of the network compute entity information about the individual words in the sentence. These two pieces of information are then combined so that the network can output information about every entity in the network. The network is trained to output information about the individual constituents in the target parse tree and their structural relationships, based on the information on the words input to the network.

Throughout this thesis, the SSN referred to will be the TSVB networks described above. SSNs embody all the generalisations and representational strengths argued for with respect to the more general class of TSVB networks (i.e. those where links from pulsing to non-pulsing units are allowed). However, there is a major source of inefficiency present in the definition of the SSN, which is the requirement that every phase introduced to the network by the input be retained throughout the lifetime of the current input sequence; this means the SSN requires a quadratic number of computation cycles with respect to the size of input sequence. For many applications, such as natural language, not all phases will be required for later computation as separate phases. For example, limitations in human cognitive abilities mean that certain sentence constructions, though theoretically possible from the grammar, simply never arise in any sample of naturally occurring text. This is due to limitations on human short-term memory [16, 70]. Thus, in many applications, it is possible for the SSN to only retain a limited number of 'relevant' phases. The problem is in determining the relevance of a phase.

In order to achieve this in a SSN, one possible extension is to use a fixed-bound queue (known as the short-term memory) of current phases, performing computation only for phases appearing in the queue. Whenever a new phase is introduced, it is placed at the head of the queue. Whenever a phase is referred to during the output, it is also placed at the head of the queue. Phases not referred to in the output will, sooner-or-later, be pushed off the end of the queue, and so forgotten. Because the queue is a fixed size, this means the SSN now only requires a number of computation cycles directly proportional to the size of the input sequence, i.e. resource requirements are now linear instead of quadratic. Throughout this thesis, the theoretical properties of the basic SSN will be of greatest interest, as the main focus is on determining the theoretical learning capability of trainable TSVB networks. However, for specific applications, where efficiency is of greater interest or cognitive plausibility important, the SSN with a short-term memory should be employed. This version of the SSN is used in the experiments in learning to parse in Chapter 5 of this thesis.

1.3.2 Experiments in learning to parse

The trainable TSVB networks are first tested on two toy grammars. These grammars are chosen to highlight specific generalisations beyond those normally expected of a connectionist network, and require the TSVB network to output structured information and generalise across multiple entities. Later experiments test the most efficient TSVB networks, the SSNs, on learning to parse samples of real natural language.

Two toy grammars

The first ability of the TSVB network to be tested is its ability to handle, and generalise across, increasing numbers of entities. Accordingly, a simplified version of the task of prepositional-phrase attachment is used, in which the network must indicate which of a preceding sequence of nouns, e.g. '$n_1 n_2 n_2$', a following prepositional-phrase, 'p', modifies. This problem requires the network to refer to a noun based on its type or position in the sequence. Moreover, by testing the network on longer sentences than it has encountered in the training data, the network must generalise learned information to increasing numbers of entities.

The second ability of the TSVB network is related, and that is its ability to output recursively structured information and generalise learned information across this structure. This ability is tested using a simple recursive grammar, which generates sentences such as 'Mary loves John', or with optional relative clauses, 'Mary loves John who likes Jane'. In this case, the network is trained on sentences with limited depths of recursion and nouns occupying restricted syntactic positions, e.g. some nouns may only appear in the subject position of a sentence. The generalisation ability of the networks is tested by using the trained network to parse sentences with deeper levels of recursion and nouns in novel syntactic positions.

In general, the various TSVB network architectures do learn the above tasks and produce good generalisation performance. However, one set of networks tends to learn faster and generalise better. These are the Simple Synchrony Networks, which share a common architectural feature – the lack of links from pulsing to non-pulsing units.

Parsing real natural language

The two toy grammars demonstrate that trainable TSVB networks learn and generalise in the manner claimed. The natural progression is therefore to test the networks on a more ambitious task, learning

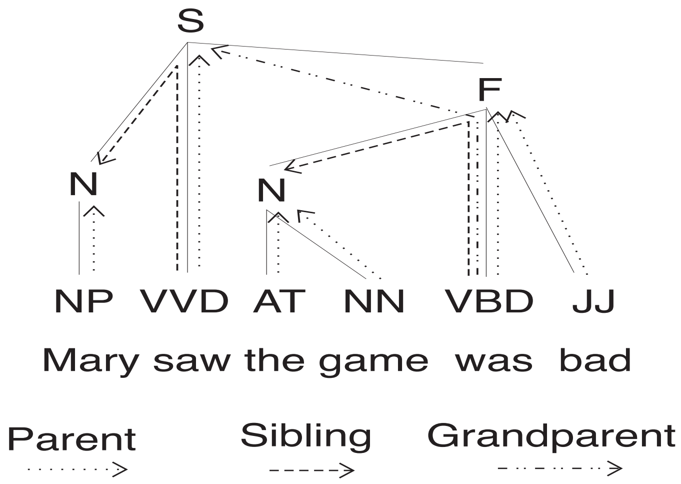


Figure 1.5: A sample parse tree. The solid lines indicate the parse tree itself, the dotted and dashed lines the relationships between the words and nodes.

to parse samples of real natural language. For these experiments only those TSVB networks most successful with the toy grammars are used, i.e. the SSNs.

The SSNs are trained to output a parse tree when a sentence, a sequence of words, is presented on its input units. The parse tree is built up incrementally as the network processes the sentence. For each input word the network should output the set of parent-child relationships which define that word's position in the parse tree. Each word is introduced to the network on a new, previously unused phase, and so introduces a new entity to the network. These entities are used in the output to represent the individual nodes of the parse tree. Figure 1.5 illustrates an example sentence, 'Mary saw the game was bad', with its parse tree shown by solid lines. The relationships output by the network for each word are indicated by the dotted and dashed lines in the figure. Three kinds of relationship are used, corresponding to the three kinds of relationship which may occur within a parse tree. The first is the 'parent' relationship, which identifies the immediate non-terminal node which each word attaches to. The parent node of these non-terminal nodes are indicated with 'grandparent' and 'sibling' relationships. This representation for parse trees is referred to as the GPS representation.

The experiments were performed using a subset of the Brown corpus, the SUSANNE corpus [87], as a source of preparsed sentences. Two changes were made to the corpus before using it in the experiments with the SSNs. Firstly, because the SUSANNE classification scheme employs semantic and meta-sentence information, some preprocessing of the corpus data was necessary, removing all information not related to the syntactic parse tree. This simplification does not affect the boundaries between constituents. Secondly, the parse tree thus obtained must be modified to conform to the requirement of the SSN and its GPS representation for parse trees that each non-terminal node be introduced by at least one word. This means that certain pairs of nodes within the SUSANNE parse tree are collapsed into one. The most important of these is the use of nodes for verbs as well as for clauses. For example, in the sentence in Figure 1.5, the SUSANNE scheme would introduce an extra node for 'saw' and 'was' to indicate that they are verbs. This would leave the 'S' and 'F' nodes without an immediate head word, and therefore, in the experiments presented here, the convention is to remove the extra nodes. Although this is a simplification, the amount of change is relatively small, with less than 1% of the non-verb constituents being affected, and the verb constituents could, in principle, be replaced on output from the SSN. In addition, this change is not linguistically unmotivated, as the result is similar to a dependency grammar [67]. The main reason for making this small change to the corpus is to simplify the SSN's representation of parse trees, and so test its ability to learn appropriately from naturally occurring text. Later work may then address the specific forms of parse tree representation used by different corpora, and whether the SSN can use them directly. Further discussion on this issue can be found in Johnson [48], which discusses how the needs for a parse tree representation differ between the corpus and the parser.

To reiterate, the primary importance of these experiments for connectionist language learning lies in their use of a corpus of naturally occurring text. Not only is the structured output representation close to the parsing scheme used in the corpus, but also the corpus is based on 'real world' sentences. The use of a parse tree representation enables standard statistical measures for evaluating the performance of a parser to be applied to this connectionist parser. The measure used compares the number of correctly output constituents with the number in the target or output parse trees. In the experiments here, the best SSN produced a performance of 80% in testing. What is notable is that this figure was not worse than that produced in training (in fact, it is better), which demonstrates that the SSN is learning a robust mapping from the input to the output and so generalising well. In addition, this good generalisation performance is based on the same architecture as was used in the toy grammars (only the number of units being altered): this transfer from toy grammars to naturally occurring text has been a major source of difficulty for previous connectionist approaches to natural language (for example, see [43, 69]).

1.4 The SSN in Context

This introductory chapter began with an argument that evaluating the importance of a new machine learning architecture required two forms of comparison, qualitative and quantitative. Evaluating the SSN on these two criteria lead to the following two questions: How does the SSN's design compare with other approaches to learning to parse? How do the results achieved by the SSN compare with those achieved by other approaches? This section provides summary answers to these two questions.

1.4.1 Qualitative comparisons

In terms of machine learning approaches to natural language, the most natural comparisons for the SSN are other connectionist approaches, which fall into two main groups: those using the Simple Recurrent Network, and those using holistic encoding networks.

The Simple Recurrent Network (SRN) [18] is a popular architecture for connectionist language learning because of its ability to learn about sequences over time. However, its successes have been confined to two basic tasks: learning to predict the next word in a sentence [19, 21, 81] or to assess whether a sentence is grammatical or not [60, 62]. Each of these tasks only requires the SRN to output a single piece of information, either the predicted word for each input, or else to indicate the grammaticality after the whole sentence has been input. Neither of these tasks is structured in the sense of producing a representation such as a parse tree.

However, the internal representation of the SRN has been shown to encode, in distributed format, a rich amount of information about the input sentence. The major question is how to extract it. St. John and McClelland [47] illustrated how such a representation could be probed to provide semantic information about the general context of a sentence, even if such information was not part of the input. For example, a sentence such as 'Mary ate the spaghetti' would return the item "fork" in response to a probe about the instrument used; this response of course depends on the set of sentences used for training the network.

A further use of the distributed representation within the SRN is demonstrated in Reilly's [80] parser, which uses a second network to 'unpack' the distributed representation into a parse tree. This second network is known as a Recursive Auto-Associative Memory [77]. Ho and Chan [43] show

how a variety of such parsers can be constructed. They are collectively known as *holistic* parsers because they first encode an input sentence into a distributed representation for the entire sentence, and then decode this representation into a parse tree.

In terms of comparing such parsers with the SSN, the central question is the difference in the internal representations used by the two approaches. With the SSN, the parsing process is performed incrementally, outputting all the relations relevant to each word as it is input. With the holistic parsers, the parsing process is performed *en masse*, outputting all the information relevant to the entire sentence after it has been input. In addition, the holistic parser must process the distributed representation a number of times, each cycle corresponding to an item in the sequential representation of the parse tree. The distributed representations within the SSN and the holistic parsers therefore take on different roles. Within the SSN, the internal representation determines which action to perform in response to the specific input word; within the holistic parser, the internal representation must capture information necessary for reconstructing the entire parse tree.

However, the holistic parser, although capable of outputting parse trees, has two major limitations. The first is that the form of parse tree is limited because it must be represented as a sequence for processing by the parser. For example, the preorder encoding used by the Confluent Preorder Parser [43] forces its output parse trees to be of fixed valency, in which each node of the tree has a fixed number of child nodes. This limits the application of holistic parsers to more realistic corpora, which typically use more complex forms of parse tree. The second limitation lies in the holistic representation itself, which seems limited in its ability to handle more than certain toy grammars [43, 69]. The major aim of the experimental section of this thesis is to demonstrate the SSN's ability to transcend these two limits. First, by illustrating how the SSN can output parse trees of similar complexity to those used in a standard corpus. Second, by demonstrating that the SSN's abilities are not confined to toy grammars, but transfer to naturally occurring text.

There are two reasons for the SSN's success. The first is the use by the SSN of temporal synchrony to inherently generalise learnt information across structure. This generalisation ability means that the SSN can apply information it has learnt about words in one syntactic position to the same words appearing in other syntactic positions. With representations such as used by the holistic parser, information can only be generalised based on the similarity of the entire parse tree representation to another. The fact that this structural generalisation is made explicit in the SSN's representation of information across pulsing units makes it qualitatively different from the internal representations used by these holistic connectionist parsers. However, other ways of making these generalisations explicit are available. For instance, Hadley and Hayward [33] use a highly structured internal representation to enforce generalisations across the structure represented by the output units. The problem with this network is that, for anything beyond the example toy grammar, the network must be provided with a different internal structure.

The second reason for the SSN's ability to learn from naturally occurring text lies in its more efficient use of its internal representation. Specifically, the SSN parser incrementally outputs its parse tree as each word is input. This enables the SSN to focus its resources on the information needed to parse, instead of additionally having to memorise the preceding part of the sentence. The incremental nature of the SSN's operation makes it akin to symbolic parsers such as PARSIFAL [63], and is distinct from the all-at-once output strategy of holistic parsers.

1.4.2 Quantitative comparisons

Two sets of experiments with the SSN may be directly compared with earlier work. The first of these is the second toy grammar. This recursive grammar was used by Hadley and Hayward [33] to verify that their Hebbian connectionist network could learn to generalise across syntactic constituents. The SSN achieves similarly excellent results, demonstrating its similar ability to generalise across syntactic constituents. Where the SSN differs from this Hebbian model is that applying the latter directly to a different grammar would be difficult, whereas the SSN handles the change smoothly.

The second comparable set of experiments are those using a corpus of naturally occurring text. The experiments are conducted to produce a set of results of comparable format to those based on statistical parsers. Specifically, the same performance measure is used with the SSN as with statistical parsers. As is shown in Chapter 5, the SSN's best result is 80% average precision/recall on the testing set. Notable about this result is that it is similar to that achieved on the training set, indicating that the SSN has learnt a robust mapping from its input to output, without overfitting. Because these results are in the same terms as those of statistical parsers, it is possible to make comparisons between the two approaches. However, direct conclusions cannot be drawn without testing such a parser on the same corpus representation, size and contents.

In an extension to the work here, Henderson [39] has presented a slight variant of the basic SSN model and compared its performance directly with that of PCFGs on identical corpora. In those results, the PCFG, due to the restricted size of the training set, was only able to parse half the test sentences, with a precision/recall figure of 54%/29%. In comparison, the SSN was able to parse all the sentences and yielded a performance of 65%/65%. Even when counting only the parsed sentences, the PCFG only had a performance of 54%/58%, compared to the SSN's performance of 68%/67% on that subset. These results show that the SSN compares well on an empirical level with a well-established, though basic, parser.

This thesis has therefore provided empirical evidence that the SSN is an effective model for learning to parse. This may be explained by the presence of Temporal Synchrony Variable Binding which provides an inherent ability to generalise across output structures. The effectiveness is demonstrated by specific results with toy grammars and a corpus of naturally occurring text. The important point to note is that the SSN transfers its ability smoothly from the toy grammars to the natural text. This contrasts with other connectionist approaches to language learning whose performance on toy grammars has so far not been shown to scale up [43, 69].

1.5 Thesis Overview

As indicated in Section 1.3, this thesis is divided into two main contributions, theoretical and empirical. Accordingly, the first two chapters of this thesis consider earlier work in these two areas, and the next two consider the separate contributions of this thesis.

The theoretical discussion begins in Chapter 2, where the two extensions of the basic connectionist network which form the Simple Synchrony Network are discussed. In particular, the chapter contains a justification of why these particular extensions are selected. To do this, Chapter 2 is divided into the three kinds of ability required for learning about language: learning statistical information about static patterns, learning about patterns across time, and representing multiple entities. All three abilities are required for effective language learning, but it will be argued that no simple connectionist network to date has provided the necessary combination of all three in a trainable architecture.

Chapter 3 contains an overview of previous work in language parsing. This begins with a summary of classical parsers, and some considerations of the parsing process from a cognitive perspective. Statistical parsers are summarised next, and then a review is made of the major models of connectionist language learning. In particular the chapter shows that standard connectionist techniques have yet to rival the results achieved with the statistical algorithms. This is largely due to representational inadequacies, which may be overcome by using the SSN. Those connectionist models which can output the necessary representations have so far not transferred well from learning about toy grammars to naturally occurring text.

Chapter 4 contains the definition of trainable recurrent Temporal Synchrony Variable Binding (TSVB) networks. These TSVB networks are a combination of the architecture of Simple Recurrent Networks (SRNs) with the representational power of TSVB units, and can be trained with a novel extension of Backpropagation Through Time, the standard training algorithm for recurrent connectionist networks. Because TSVB units come in two varieties, pulsing and non-pulsing, there is a wide range of possible TSVB network architectures, and the chapter contains several examples from this range. In addition, the use of a bounded queue of phases is described, which provides a mechanism for retaining only the relevant phases for computation by the network, so making its resource requirements linear in the length of input sequence. The chapter concludes with the first of the experimental results, in which the specific generalisation abilities of TSVB networks are tested using two toy grammars. These generalisation abilities are additional to those inherent to the SRN architecture, and demonstrate that the TSVB networks and training algorithm as proposed do indeed combine the different abilities of the two extensions to the basic connectionist network. At this point the main theoretical contribution of the thesis has been described, and the experiments have demonstrated its soundness.

Chapter 5 contains a more ambitious set of experiments, attempting to transfer the SSN directly to learning about naturally occurring text. The chapter describes how an appropriate representation of the parse trees within the corpus can be developed for use by the SSN, and ends with a series of results achieved by the various SSN architectures with this corpus.

Chapter 6 concludes this thesis with an evaluation of the SSN and suggestions for further work.

Chapter 2

Connectionist Networks

Connectionist networks (also known as neural networks) are a popular architecture for learning within a wide range of domains. The basic principle being that complex representations can be constructed from the interaction of many simpler ones. Some of the motivation for this approach is based on an analogy with the brain, which is composed of many neurons, each of which appears relatively simple in computational terms, except that each has many connections with other neurons.

A more computer-science oriented motivation is that this approach provides a robust and general learning system; the same representation and training algorithm can be used in a wide range of applications, with only the training data being varied. In particular, excellent results have been achieved in applications which involve learning about static patterns, in which a given input pattern is matched to an output pattern. Typical examples include learning to identify people from photographic portraits [15], driving a car down a highway [78] and a system which learns to recognise spoken words [57]. This adaptability of connectionist networks rests on the fact that, given enough training data, a suitably-sized connectionist network can eventually learn to approximate an expressive range of functions [45]. Results on the power of connectionist representations were given from the early days of connectionist networks [65]. Much of the interest in connectionist networks is that these complex representations can be learned by the network through training with backpropagation [86].

However, in spite of these successes, the results of applying connectionist networks to more complex cognitive tasks have been disappointing. These more complex tasks include natural language processing (NLP) and, more generally, a range of problems requiring the inference of grammatical information. The distinction between these tasks and the ones involving static patterns is two-fold. Firstly, the patterns presented on the input form a temporal sequence, so that the output for a given pattern depends on the patterns encountered previously in the sequence. Secondly, the outputs themselves may refer to items encountered in earlier parts of the sequence. These properties are both present in natural language: a given noun in a sentence may be a subject or object depending on the other words in the sentence, and the noun will be the subject or object of a particular verb.

The aim of this thesis is to develop a new connectionist architecture which is appropriate for learning in domains such as NLP. In order to justify the design of this new architecture, and show why it is an important addition to the wide range of connectionist architectures already in existence, some coverage of the literature on connectionist networks is needed. Accordingly, this chapter describes connectionist networks in terms of their ability to learn and represent particular classes of problem. The chapter focuses on the three broad areas as described above: learning to recognise static patterns, learning to recognise patterns across time and learning about entities (to solve the reference problem).

Each section of this chapter deals separately with each of these areas. Section 2.1 describes the basic feed-forward connectionist network. This is composed of units which pass the weighted sum of their inputs through a sigmoid activation function and output a real value between 0 and 1. A training algorithm, backpropagation, is also defined, for training feed-forward networks, i.e. those where activation only passes from the input units towards the output. These networks are most suited for learning to recognise static patterns, i.e. those which are presented to the network in independent time periods.

Section 2.2 describes networks for learning about patterns across time, where the current output depends on the previous input to the network as well as the current input. Two basic approaches to this problem exist: time-delay neural networks, which use a buffer of inputs to present previous inputs alongside the current input, and recurrent networks, which use previous activations of the network as a context for the current input. As will be seen later in this thesis, the recurrent network is very popular for connectionist language learning. Section 3.4 in Chapter 3 discusses some of the work in this area with Simple Recurrent Networks [18, 19].

NLP additionally requires of an architecture the ability to learn about structured information, for instance, to output a parse tree representation of an input sequence of word-tags. Structures such as parse trees also encourage certain forms of generalisation, which led to the observation by Fodor and Pylyshyn [24] that a particular regularity in language, known as systematicity, could not be represented by the standard connectionist network. Section 2.3 covers a variety of connectionist approaches for representing structured information. In particular, Temporal Synchrony Variable Binding (TSVB) [92] is introduced, which will form the focus of Chapter 4, implementing TSVB in trainable connectionist networks.

2.1 Networks for Learning Static Patterns

This section describes standard feed-forward connectionist networks. Such networks are defined in terms of the individual units composing the network, their interconnections and how they can be trained. In this section the networks are restricted to being *feed-forward*, which means activation may only flow away from the network's input units towards its output units. A suitable training algorithm for such networks is also defined. For this kind of network, whether the input-output patterns are the same (as in auto-associative tasks) or not (as in hetero-associative tasks), the target for the network is a function mapping each input pattern to a given target output pattern independently of the other patterns in the data set, i.e. the target pattern for a given input is static.

2.1.1 Defining the network

The basic building block for a connectionist network is the *unit*, as depicted in Figure 2.1(a). Each unit has a number of *incoming* links, and one or more *outgoing* links. The incoming links carry activation from previous units, and the outgoing links carry activation on to later units. There is also a *bias* link, labelled θ, which is assumed, in all that follows, to be an incoming link from a unit whose activation is permanently 1, and therefore does not require special treatment (i.e. it is assumed to be present for every unit in all that follows, without needing explicit description). Each link carries activation from a *source* unit to a *destination* unit, scaling this activation by a *weight*. Thus, $o_d = w_{ds} \times o_s$, where o_d is the activation passed to the destination unit, o_s the activation of the source unit, and w_{ds} the weight on the link between them.

Standard feed-forward networks have three kinds of unit: input, hidden and output, as illustrated in the example network in Figure 2.1(b). Input units are those where the input pattern is presented to the network. Therefore, the output activation of an input unit, j, is that of its corresponding input pattern, in_j.

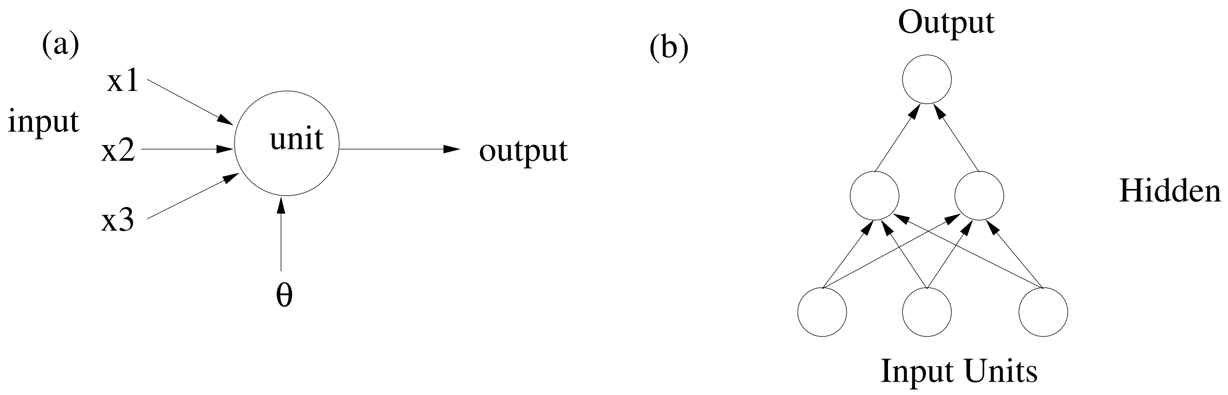


Figure 2.1: The standard connectionist unit, and a feed-forward network.

Each hidden or output unit, j, forms its *net* input by summing the activations received from its incoming links, indexed by the set of integers, Inputs_j.

$$\text{net}_j = \sum_{i \in \text{Inputs}_j} w_{ji} o_i$$

This net function is then passed through a non-linear thresholding function, known as the *sigmoid activation function.*

$$o_j = \sigma(\text{net}_j) = \frac{1}{(1 + e^{-\text{net}_j})}$$

This completes the definition of standard feed-forward connectionist networks; the next section addresses the training of such networks.

2.1.2 Training the network: Gradient descent

The trainable parameters within a connectionist network are, firstly, the weights of each link, and, secondly, the number of units and how they are interconnected. This section describes the standard gradient descent algorithm for training the weights upon each link of a general feed-forward network. There also exist a number of techniques for constructing networks by changing the number of units and their links (for example, cascade-correlation [22]). These techniques, although potentially helpful in reducing training times, are not used in the experimental part of this thesis, and so are not described here.

In order to train the weights within a fixed network architecture, the standard technique relies on *gradient descent*. The basic principle behind gradient descent is that the network's performance on a set of training data can be plotted against the possible values of its weights to form an *error surface*. Somewhere within that surface is a point of minimum error, and the purpose of the training algorithm is to locate that minimum point. Gradient descent does this by using the gradient of the error surface as a means for altering the weights within the network. Repeated adjustments of the weights are made to move the network across the error surface in a 'down-hill' fashion, aiming to locate a point of minimum error. Accordingly, there are three factors to consider: how the gradient information for the error surface is to be computed, how this determines the changes to be made to the weights, and when the changes are to be made.

In order to compute the gradient information, the error surface of the network must be found with respect to the weights. In a supervised learning task, the error of each unit is judged according to the target output for the network. Thus, a set of integers, D, indexes those units in the network which have a target output, d_k for unit $k \in D$. The output error of unit k is,

$$e_k = \begin{cases} d_k - o_k & \text{if } k \in D \\ 0 & \text{otherwise} \end{cases}$$

The aim of gradient descent is to minimise the sum-squared error of the network, which is defined as,

$$E = \frac{1}{2} \sum_{k \in U} e_k^2$$

where U represents the set of all units in the network.

The gradient of this function with respect to the weights is found through partial differentiation, and the change in the weight is assumed to be linearly proportional to this gradient. A standard derivation (e.g. Rumelhart & McClelland [85]; Mitchell [73]) leads to the following equations defining the change to the weight between arbitrary units i and j:

$$\begin{aligned} \Delta w_{ji} &= \eta \frac{\partial E}{\partial w_{ji}} \\ &= \eta \delta_j o_i \\ & \text{where } \eta \text{ is a learning coefficient.} \end{aligned}$$

The value for δ_j refers to the error on any individual unit of the network, and this is found from the sum of the unit's own error with respect to the target and the error on any units which it is input to; the sum is then scaled by the differential of the activation function. Thus, for a unit k (assumed to be other than an input unit),

$$\delta_k = o_k(1 - o_k) \left(e_k + \sum_{l \in U} w_{lk} \delta_l \right)$$

By convention, $w_{lk} = 0$ if $k \notin \text{Inputs}_l$, i.e. if no such link exists.
Having shown how the gradient of the error function is computed, and how this determines the weight changes to be made, all that needs determining is the time for computing these changes, and there are two basic choices. The *on-line* approach updates the weights after each pattern is seen by the network. *Batch* update waits until every pattern in the training set has been presented to the network, and then every weight is changed based on the total set of computed updates. The latter approach more accurately approximates the gradient of the total error surface of the network upon the training set. However, in trials, the resultant performance of the two approaches does not differ significantly, and the on-line approach has the advantage that it is less likely to fall into local minima. That is, areas of the error surface which are a minimum in the local, not the global, context. Such an effect is shown by Lawrence et al. [62].

2.2 Networks for Learning Dynamic/Temporal Patterns

The previous section has described the basic feed-forward connectionist network and its training algorithm, which is an excellent model for learning to compute an output pattern in response to an input pattern. This and the next section present extensions to this model which make it suitable for dealing with more complex problems typical of higher cognitive processing. In this section, the handling of patterns which extend across time is considered, and in the next, the representation of structured patterns.

So far, a connectionist network computes its current output directly from its current input. However, some problems require either or both of the input and output to represent a set of values presented over time, a sequence. For example, in processing language, the syntactic class of the current word may depend on the preceding words. There are three types of problem which require this temporal treatment:

Sequence Recognition The network is required to produce a particular output pattern when, or just after, a specific input sequence is seen. A typical example of this is speech recognition, where the word spoken is identified after processing the sequence of sounds comprising it.

Sequence Completion or Prediction The network is required to produce a sequence of outputs, completing the pattern begun by the input sequence. This is a generalisation to dynamic patterns of the auto-associative or pattern completion task, as performed by the feed-forward networks. An example of this would be continuing a time series after seeing the first few terms.

Temporal Association The network is required to produce a sequence of outputs in response to a specific input sequence, with the output sequence, in general, being different to the input sequence. This is a generalisation of the hetero-associative task to dynamic patterns, and includes sequence recognition and completion as special cases.

The problem of learning to parse natural language, as considered in this thesis, is an extended example of the first type of problem, sequence recognition. For example, constructing a parse tree requires forming the syntactic class for the current word, which is determined by the previous words in the sentence. The feed-forward networks considered in Section 2.1 are unsuitable for this kind of problem as they use only the current input to compute the current output. There are two basic solutions to this problem: time-delay neural networks and recurrent networks.

The most straight-forward is the time-delay neural network (TDNN), which uses a buffer on the input to maintain a copy of previous input items, and so present them to the network along with the current input. A standard feed-forward network can then be used, which treats the entire sequence as a single input value, and therefore computes its output based upon the entire sequence. Two basic problems with this approach are the need to predetermine the maximum length of input sequence to be considered at any one time, and the need for extra training and space requirements to deal with the additional units and links.

These problems are solved in the second approach, which augments a standard feed-forward network with context units to form a recurrent network. A context unit's activation is a copy of a hidden unit's activation from a previous time step. Input items are presented one at a time and the network uses its context units to preserve information computed by the hidden units over time, and therefore input items can influence computations on later items.

The following sections deal with these two approaches in more detail.

2.2.1 Time Delay Neural Networks

The most direct method of computing an output pattern from an input sequence is to retain the entire input sequence as input to the network. One technique for achieving this is to use a window over time. For instance, in an application where sequences of up to seven items in length are of relevance, the current input to the network will be the simultaneous presentation of the current input item, the previous input item, and the input items from up to seven previous time steps. An example of this type of network, NETtalk [89], is considered in more detail below. There are a number of similar approaches. For instance, a shift register can be used, which acts as a first-in-first-out buffer, holding the last seven items presented, each new item ‘pushing out’ the oldest. The network computes its output based upon the contents of this buffer.

Each of these techniques is similar in providing the network with links to input items across a sequence, and they share two common problems. The first is that the maximum length of input sequence considered at any time is restricted to the amount of space dedicated to the input, and the network must be preconfigured (and trained) on the expected sequence length. Thus, the network is able to handle sequences of arbitrary length, but not dependencies between items of the sequence at arbitrary separations. The second problem is that each position in the sequence has a separate physical link, and each link has to be trained independently. This implies that for the network to learn the consequence of the presence of a given input item, irrespective of its position, it must alter weights appropriately on links for every possible sequence position, which requires duplication of effort. For some tasks this may be beneficial, as a different effect may be required depending on the relative position of the input item in the sequence. For other tasks an identical effect irrespective of position in the sequence is required. For example, performing a logical function on the previous n inputs requires an identical effect for each element in the sequence irrespective of position, but a function where the value of each input decreases with age would not. Problems where we need identical performance independent of sequence position can make the training process longer, because examples of a particular input will have to be presented for each possible position of the input in the sequence.

NETtalk

One example application of a TDNN is the NETtalk project [89] which aimed to train a network to pronounce English text. The task for the network was to determine the relevant phoneme for each letter in the sample of English text presented to it. However, the rules for generating correct pronunciation of English are not the result of mapping individual letters directly to phonemes, as English is not a phonetic language. Instead, phonemes are selected based upon the current letter and its context, both the preceding and the succeeding letters are of relevance. The task is therefore one of sequence recognition, and NETtalk used a window of 7 characters.

The architecture of NETtalk is shown schematically in Figure 2.2 (after [41]). The figure illustrates how 7 consecutive characters from the text are presented to the network, with the output being computed for the central letter. Each input letter triggered one of 29 inputs (one for each character, plus some punctuation); these 7x29 input units fed into 80 hidden units, with 26 output units completing the network. The network architecture is a standard feed-forward network, and is trained using backpropagation. NETtalk was trained on 1024 words with the target pronunciations, and achieved 95% accuracy after 50 training epochs. On testing this on some unseen text, the network achieved 78% accuracy, producing quite intelligible text.

The generalisation performance of the TDNN from the seen text to the unseen is impressive, and demonstrates that such a network can learn the context of an input item with respect to a sequence. This result can be understood because the specific generalisations in the application match those provided by the network. Firstly, the window of seven letters onto the text was sufficiently long to provide the necessary context for each phoneme, and this will not change with different texts. Secondly, the letter for which the phoneme is being computed is always in the same location of the input window, and the preceding and succeeding letters are always in the same locations of that window. This means that no generalisation is required of information learned about letters in one position of the window to other positions of the window; instead, each window location corresponds at all times to a particular contextual position for the current phoneme.

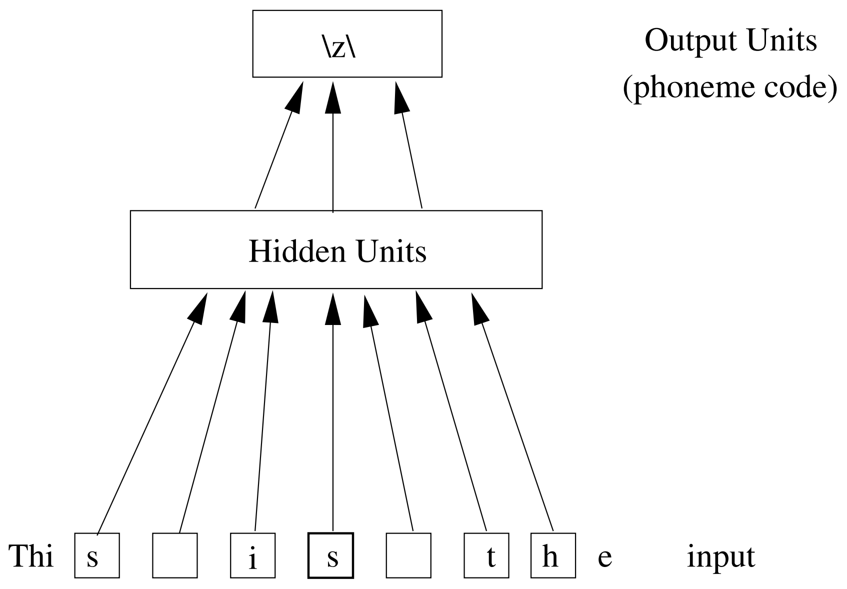


Figure 2.2: Diagram of NETtalk Architecture.

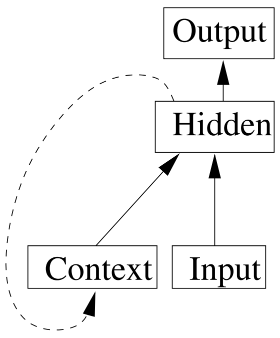


Figure 2.3: Diagram of Simple Recurrent Network

The next section deals with Simple Recurrent Networks, which do not have the same constraints as the TDNN, and are therefore suited for a different class of application.

2.2.2 Simple Recurrent Networks

The previous section presented one solution to the problem of working with input sequences. This involved maintaining a window or buffer allowing the network access to items across a period of the input sequence. The problems with this technique arise from the duplication of input units and links and the likelihood that this will lead to increased training time if the application requires a generalisation of learned information across different input positions. These problems are caused by the duplication of units in space to represent sequences spanning a period of time. Instead, by representing the context of the sequence in time, the network can itself carry forward information from previous input items to aid in computing output for the current input. The most significant example of this approach is that of Elman [18, 19], the Simple Recurrent Network (SRN), which uses context units to capture information from previous positions in the sequence. Jordan [49] also used context units in a slightly different architecture to Elman. The SRN is described here because of the experiments in natural language processing performed by, among others, Elman [19, 21], Lawrence et al. [60, 61, 62] and Reilly [81], and discussed in Section 3.4 of Chapter 3.

The basic architecture of an SRN is shown in Figure 2.3. On each time step the next input item is presented to the network. Activation is then passed from the input and context units into the hidden units and then to the output units, as in a standard feed-forward network. Finally, the activation of each hidden unit is copied into its associated context unit, as indicated by the dotted-link in the diagram.

The definition of a feed-forward network can be extended in a natural manner to a network using context units to implement recurrence, including a one-to-one function C which maps each unit to its associated context unit; the function C^{-1} is its inverse, mapping each context unit to its associated unit. The output of any unit j is defined as follows:

$$
\begin{aligned}
\text{net}_j(t) &= \sum_{i \in \text{Inputs}_j} w_{ji} o_i(t) \\
o_j(t) &= \begin{cases} \text{in}_j(t) & \text{if } j \text{ is an input unit} \\ o_i(t-1) & \text{if } \exists i.j = C(i) \text{ and } t > 1 \\ 0 & \text{if } \exists i.j = C(i) \text{ and } t = 1 \\ \sigma(\text{net}_j(t)) & \text{otherwise} \end{cases}
\end{aligned}
$$

Note how the second and third lines for the activation of unit j apply if j is a context unit; if so, then the activation of unit j is that of unit i in the previous time step, unless there is no previous time step (i.e. $t = 1$), when unit j's activation is 0.

Training Simple Recurrent Networks

Although one advantage that SRNs have over TDNNs is their compaction of a spatial representation of the input sequence into a temporal representation, for the purposes of training this is an inconvenience. Training requires the network to review all the steps in the computation which led to the current output, and this requires a review of the operation of the network over all previous time steps.

Training of SRNs is most simply performed using Backpropagation Through Time (BPTT) [85]. The first step in using BPTT is to unfold the network over time, and so retrieve the full history of activation through time.

In Figure 2.4 the SRN of Figure 2.3 has been unfolded to show the computation steps which produce the output dependent on the last three input items. The activation of context units at a time t is a copy of the activation of the hidden units at time $t-1$, and this is indicated by the dotted links. It is this link which provides for the passing of activation between time steps. Note that the context units will begin with zero activation, as there are no preceding hidden units. This unfolded SRN is a feed-forward network, and may be trained using standard back-propagation. The output error is fed back through the links, and an error is associated with every hidden unit at each time step. The one complication with this technique is that the link between, for example, input unit 1 and hidden unit 1 is duplicated at time steps 1, 2 and 3. The weight changes associated with this link at the three time steps are therefore added together, and the total change applied to every copy of the link. It is this complication which enables the SRN to compact the spatial representation of a sequence over time into a temporal one.

Weight-update equations can be given for recurrent networks trained with BPTT by a simple extension to those given for feed-forward networks. As before, the gradient information is computed from the error surface of the network with respect to the weights. In a supervised learning task, the error of each unit is judged according to the target output for the network. Thus, a set of integers, $D(t)$, indexes those units in the network which have a target output at time t, $d_k(t)$ for $k \in D(t)$. The output error of unit k is,

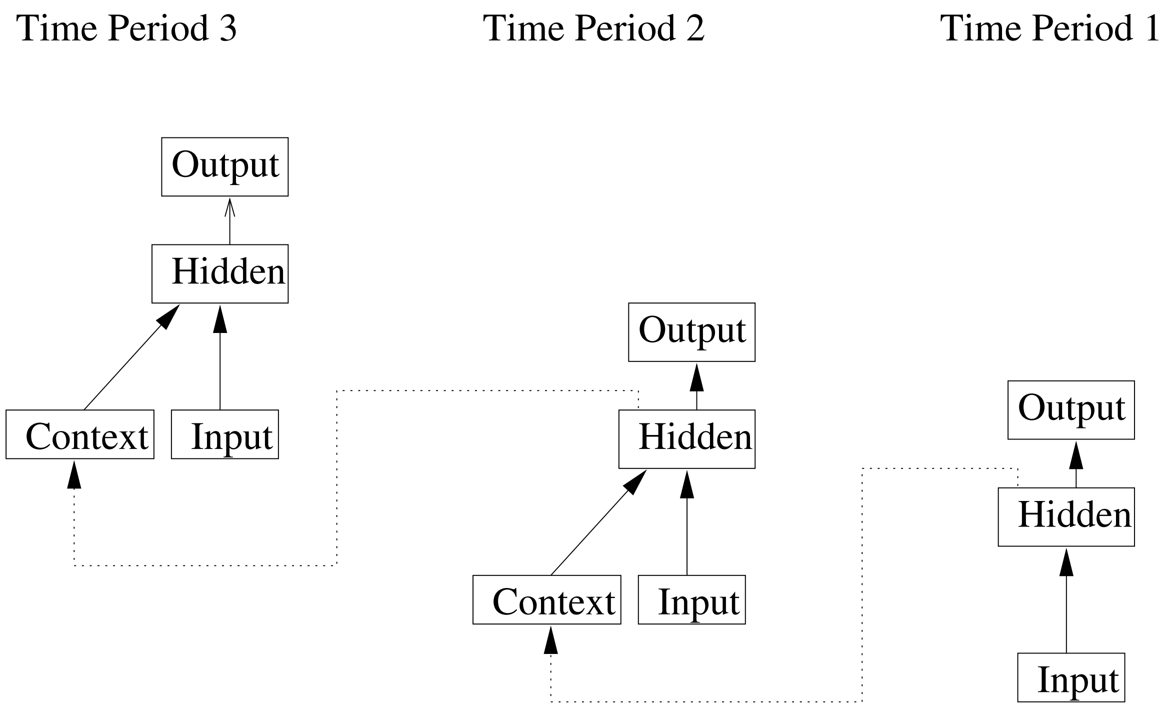


Figure 2.4: Diagram of Unfolded Simple Recurrent Network

$$e_k(t) = \begin{cases} d_k(t) - o_k(t) & \text{if } k \in D(t) \\ 0 & \text{otherwise} \end{cases}$$

The aim of gradient descent is to minimise the sum-squared error, E, of the network over a sequence of time steps $[t_1, t_2]$.

$$E(t_1, t_2) = \frac{1}{2} \sum_{t=t_1}^{t_2} \sum_{k \in U} e_k(t)^2$$

where U represents the set of all units in the network.

As before, the gradient of this function with respect to the weights is found through partial differentiation, and the change in the weight is assumed to be linearly proportional to this gradient.

$$\begin{aligned} \Delta w_{ji} &= \eta \frac{\partial E(t_1, t_2)}{\partial w_{ji}} \\ &= \eta \sum_{t=t_1}^{t_2} \delta_j(t) o_i(t) \end{aligned}$$

where η is a learning coefficient.

The value for $\delta_j(t)$ refers to the error on any individual unit of the network at time t. The input units and context units require special treatment. By definition, input units never have an error, and context units act as place-holders for passing activation between time periods and not as computing units in their own right. Therefore, context units are used during training to pass back to their associated hidden units the sum of the error on the hidden units which they are input to, i.e. the error on a context unit j is:

$$\delta_j(t) = \sum_{l \in U} w_{lj} \delta_l(t)$$

The error of non-context units is affected by three elements: the unit's own error with respect to the target, the error on any units which it is input to, and the error received from the next time step via any context unit. The sum of these values is then scaled by the differential of the activation function. Thus, for a unit k (assumed to be neither an input nor context unit):

$$\delta_k(t) = o_k(1 - o_k) \left(e_k(t) + \sum_{l \in U} w_{lk} \delta_l(t) + \delta_{C(k)}(t+1) \right)$$

By convention, $w_{lk} = 0$ if $k \notin \text{Inputs}_l$, i.e. if no such link exists. Also, $\delta_{C(k)}(t+1) = 0$ if $t = t_2$ or k does not have a context unit.

This procedure is, however, inefficient in its use of space; copies are made of the network for every time step in the sequence. Consequently, the unfolding is often kept to a fixed maximum to keep this within a limit. Once the network has been trained, the 'folded' form may be used in testing the performance on new data. Thus, the network takes up minimal space during operation.

Two other techniques for training SRNs are Real Time Recurrent Learning [103, 102] and Time-Dependent Recurrent Backpropagation [75] Each of these techniques requires large storage space, for the same reasons given above for the unfolding process – training involves a review of all steps of the computation which led to the current output value. The difference is that, instead of feeding the error backwards through time, Real Time Recurrent Learning propagates information about the gradient of the error surface *forward* through time. Unfortunately, this approach has greater computational complexity than BPTT, as reported by Williams and Zipser [104].

2.2.3 Comparison

In order to add the ability to handle patterns across time to the standard connectionist network, two proposals have been considered. The first, Time Delay Neural Networks (TDNNs), uses a window or buffer over the input sequence in order to present a sequence of input items to the network at one time. This has its limitations, particularly as the size of the window must be predetermined at the training stage. The second proposal is the Simple Recurrent Network (SRN), which uses context units to retain activation of its hidden units from previous time steps. The SRN provides effective learning of patterns across time, and does not have the limitation of a fixed bound on the length of temporal dependencies which it can learn. This use of context units also leads to a more compact representation.

The effect of this representation on information acquired during learning can be seen when comparing the unfolded SRN to the TDNN. The SRN is then duplicated as many times as there are items in the input sequence, and takes on the form of a TDNN. The difference is that the TDNN retains the sequence in spatially-separated units, whereas the SRN retains the sequence in temporally-separated units. This means that different weights are applied to different sequence positions in the TDNN, but the same weights are applied to every sequence position in the SRN. This generalisation of the information learnt about an input sequence can be seen by comparing the re-folded SRN with the TDNN.

The SRN therefore has the advantage over the TDNN in domains where identical performance irrespective of sequence position is required, and also where the relevant length of dependencies within a sequence cannot be predicted in advance. Such domains include natural language processing, which accounts for the popularity of the SRN in this area (see Section 3.4 in Chapter 3).

2.3 Networks for Learning About Entities/Structured Patterns

Although connectionist networks, such as those described in the previous two sections, have been successfully applied to a range of problems, further extension are required for applications involving higher-level cognitive tasks, such as natural language processing, especially when comparisons are made with other symbolic or statistical systems. One reason for this is the ability of a symbolic representation to represent structured information. Not only are the specifications of higher-level problems, syntactic parse trees for example, in the form of structured information, but also the structure encourages specific kinds of generalisation when learning. Applications of connectionist networks to both these points are considered in this section.

The difficulty connectionist networks have with structured information may be seen in the ability of a symbolic system to refer to another data structure by encoding its address into the current data item. This ability is derived from the use of arrays and lists as primitive data types in conventional computer programs (a point made by Pollack [77]). By this means, trees and sequential lists are readily encoded. However, this ability is not similarly available to connectionist networks, which do not have equivalent primitive data types. This has implications in the levels of abstraction which a connectionist network can handle. This difference arises in the property of systematicity, which is a specific kind of generalisation prevalent in higher-level cognitive abilities. Systematicity is manifested by the interchangeability of basic elements within a compositional structure. For example, a system representing the sentence 'John loves Mary' as a parse tree will automatically be capable of representing the sentence 'Mary loves John', due to the interchangeability of nouns within a sentence. This ability was used by Fodor and Pylyshyn (F&P) [24] in their critique of connectionism as an adequate cognitive framework for higher-level tasks such as natural language processing.

This section begins with a brief introduction to some possible forms of structured representation. A number of problem types involving structured representations is identified, and hence a careful focus is required within the following discussion. The major types of connectionist network developed for handling structure are covered, but because this thesis is concerned with natural language parsing, more emphasis is placed on those networks which handle features of relevance to this domain.

The first approach considered is the Recursive Auto-Associative Memory (RAAM) architecture of Pollack [77], which learns holistic representations of sentences, and was one of the first attempts to address the issue of representing structured data within a connectionist framework. Pollack argues for the uniqueness of connectionist representations, in that they can be manipulated without extracting their constituent elements. The RAAM manipulates structured representations such as parse trees by learning distributed holistic representations for sub-trees. It can also be shown that these representations do generalise in accordance with systematicity [7].

The Simple Recurrent Network (SRN) has been used widely in connectionist language learning (as will be discussed in Section 3.4), even though the typical output representations available preclude the use of parse trees. One approach to extending SRNs to handle structured information is by including generalised recursive neurons [95]. This extension has been used in encoding networks which learn transformations between, and classifications of, instances drawn from various structured representations [25, 26].

In order to clarify some of the claims for systematicity made by F&P, Hadley [30, 31] proposed a learning-based definition of systematicity. Later work has developed a connectionist architecture which learns generalisations in accordance with this proposed definition of systematicity [32, 33]. The architecture relies on a non-standard training algorithm and specific network design to enforce certain symbolic properties, such as localised activation values and non-distributed internal representations, which provide for the types of generalisation being tested for.

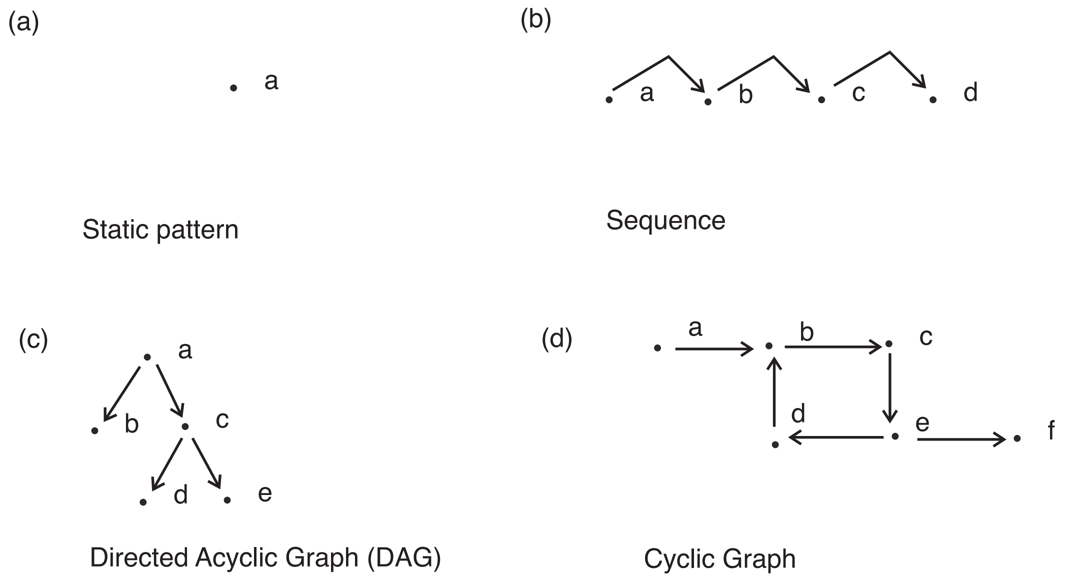


Figure 2.5: Four types of structured pattern

The final example covered here is that of Shastri and Ajjanagadde [92], who proposed a technique which directly extends the definition of connectionist networks so as to represent independent entities; this technique is known as Temporal Synchrony Variable Binding (TSVB). TSVB uses units which pulse within each time period, so that units pulsing in synchrony can represent features about the same entity. This use of pulsing units also enables features identified by the network to refer to other entities, and therefore TSVB networks can represent structured information.

This section contains a summary of these approaches, and suggests that the manner in which TSVB encodes appropriate generalisations makes it an interesting technique for expanding the achievements of connectionist networks into higher-level cognitive areas. However, no effective training algorithm for general TSVB networks has been previously proposed. This lays the foundation for Chapter 4, which considers how a training algorithm for connectionist networks employing TSVB can be developed.

2.3.1 Structured representations

A structured representation is a complex pattern, such as a list, tree or graph of variable size and complexity. Domains which require such representations pose two important questions. The first is the issue of representation. Within the connectionist framework, such as the models contained in the previous two sections, representations are typically not structured, and so some method is required for actually handling the information. The second question is that of generalisation. Domain-specific generalisations can be encouraged by the form of the structure. Ideally a connectionist network should be capable of recognising and utilising such generalisations where present, especially as they augment the standard ability to generalise by interpolation. Between them, these two issues suggest ways in which the connectionist networks from the previous two sections may require changing in domains requiring the use of structured representations.

Figure 2.5 illustrates four types of structured pattern, based on the admissable set of links between elements in the pattern. The figure illustrates:

the static pattern Each pattern is a single node containing its label but no links, i.e. each pattern is a fixed-length vector of real numbers. This form of pattern is handled by the basic feed-forward networks covered in Section 2.1.

the sequence Each pattern is a set of nodes, with links indicating the linear sequence. This form of pattern is handled by the networks covered in Section 2.2. Examples of the sequence include sentences composed of words, and lists composed of sequences of primitive elements.

directed acyclic graphs (DAGs) Here, each node may be connected to any other node, with the restriction that no cycles may be present. A typical example is the tree structure, such as used for parse trees.

cyclic graphs Here, each node may be connected to any other node without restriction. A typical example is the graph structure for a complex chemical.

Given these four types of structured pattern, a wide range of possible applications may be defined. Indeed, the four types of pattern permit 16 types of problem. For example, in the word-prediction task, the SRN maps sequences to single numbers, i.e. sequences $\rightarrow$ points. Or in classifying chemical structures, a cyclic input structure must be converted into a numeric output structure, i.e. cyclic $\rightarrow$ point. As will be seen below, different connectionist networks have been developed for the various types of problem. The interest in this thesis is with natural language parsing, which requires an ordered sequence of words to be converted into a structured output representation, in the form of a parse tree, i.e. sequence $\rightarrow$ DAG. Accordingly the major connectionist approaches to handling structure will be summarised below, but the discussion will be biased towards systems which are closest to the application of parsing.

Before summarising some connectionist approaches to handling structure, it is necessary to consider the role of domain-specific generalisations. One of the more prominent criticisms of the connectionist approach has been its ability to handle typical generalisations present in a domain such as natural language. Specifically, Fodor and Pylyshyn (F&P) [24] introduced the concept of *systematicity*, which identifies a class of regularity found in domains such as natural language. The ensuing debate centred on whether connectionist networks do or do not generalise in accordance with systematicity. The next several paragraphs review the main ideas behind systematicity, which will be seen to form a secondary motivating factor in considering the specific use of structured representations within connectionist networks.

F&P stressed two points which define a pattern of regularities for theories of cognitive processing. The first is that mental representations have a combinatorial syntax and semantics. A representation in a combinatorial syntax may be in one of two forms: structurally atomic forms consist merely of the basic units for the representation, whereas structurally molecular forms consist of some combination of the atomic forms. For instance, a sentence may be a simple word, 'Help', or a sequence of words, 'Help is coming'; the former is structurally atomic, and the latter structurally molecular. In addition, for a representation to have a combinatorial semantics, its semantic content must be a function of the semantic contents of its syntactic parts and their constituent structure. The second defining point is that operations on mental representations may be defined purely on the form of the representation.

These points are illustrated by a generative grammar for simple sentences, e.g. `S = N V N`. In F&P's terms, `S` is a structurally molecular representation formed by combining the structurally atomic '`N`', '`V`' and '`N`'. If this grammar generates the sentence 'John loves Mary' then it must also generate the sentence 'Mary loves John', because "John" and "Mary" are the same type of constituent. It is also possible to define, for example, the subject of the sentence based purely on the structure, i.e. the subject of `S` is the first `N`.

F&P claim this phenomenon of systematicity is widespread in cognitive processes, and any theory of cognition must both exhibit and explain it. As expressed by Minsky and Papert [72]:

Architecture	Approach	Type of problem
RAAMs	compressor + reconstructor	DAG $\rightarrow$ point $\rightarrow$ DAG
Encoding network	generalised recursive neurons	cyclic graph $\rightarrow$ point
Hebbian learning	tensor product	sequence $\rightarrow$ DAG
SHRUTI	temporal synchrony	DAG $\rightarrow$ DAG

Table 2.1: Summary of architectures considered, listing their basic approach to handling structured patterns, and the type of mapping between structured patterns

"no machine can learn to recognise X unless it possesses, at least potentially, some scheme for representing X". And F&P's arguments are based on the belief that connectionist networks do not have the potential for representing X, where X is combinatorial (syntactic) constituent structure, and hence cannot exhibit (semantic) "systematicity" of thought processes. This claim is strengthed in Fodor and McLaughlin [23] (emphasis in original):

> "the point of the problem that systematicity poses for connectionists ... is not to show that systematic cognitive capacities are *possible* given the assumption of a connectionist architecture, but to explain how systematicity could be *necessary* – how it could be a law that cognitive capacities are systematic – given those assumptions."

The critics require systematicity to be a nomological necessity of a connectionist network and not merely "wired in". The difficulty for connectionists is that F&P did not provide a formal definition for systematicity, leading to confusion as to what the networks should be achieving. This difficulty was addressed by Hadley [30, 31], who gives formal definitions for different kinds of systematicity, introducing the idea that systematicity is a phenomenon arising during learning. This idea is important, as connectionist networks are primarily models of learning. In Hadley and Hayward [32, 33] a connectionist network is given which satisfies this definition of systematicity.

Before moving on to some specific proposals which attempt to incorporate structured representations or an ability to generalise in accordance with systematicity into a connectionist framework, it is useful to consider the converse view in the language domain. On the issue of systematicity, Elman has suggested that the emphasis on localist representations and compositional grammars is misplaced [20]. Instead, the relationship between word meaning and syntactic structure may be less clearcut than Fodor and Pylyshyn have stated, and some of the generalisation properties they claim as an aspect of systematicity may arise from other considerations. Elman's argument is based on the semantic changes which a word derives from its context. This may be explained by an alternative linguistic model, which uses the concept of *accommodation*. Accommodation [58] describes the context sensitivity of constituents. For example, the meaning of the word "run" when applied to humans must be adjusted when applied to four-legged animals, such as horses. This adjustment of constituents is readily modelled by distributed representations, as found in a Simple Recurrent Network (SRN). A good example of this is found in one of Elman's experiments [19], where the precise internal representation of each word reflects the preceding words in the sentence.

Although these arguments are sound, it is still true that a large amount of the information contained within a word should be generalised across contexts, although it should be sensitive to the local details. In addition, a number of applications for connectionist networks require the ability to learn about and use more structured representations. Hence, a number of such connectionist architectures have been developed. A representative sample of these is presented below, and Table 2.1 summarises the type of problem each approach addresses.

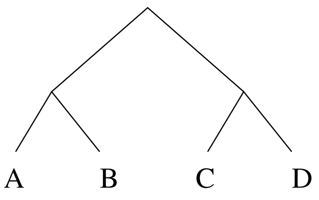


Figure 2.6: Example of a binary tree

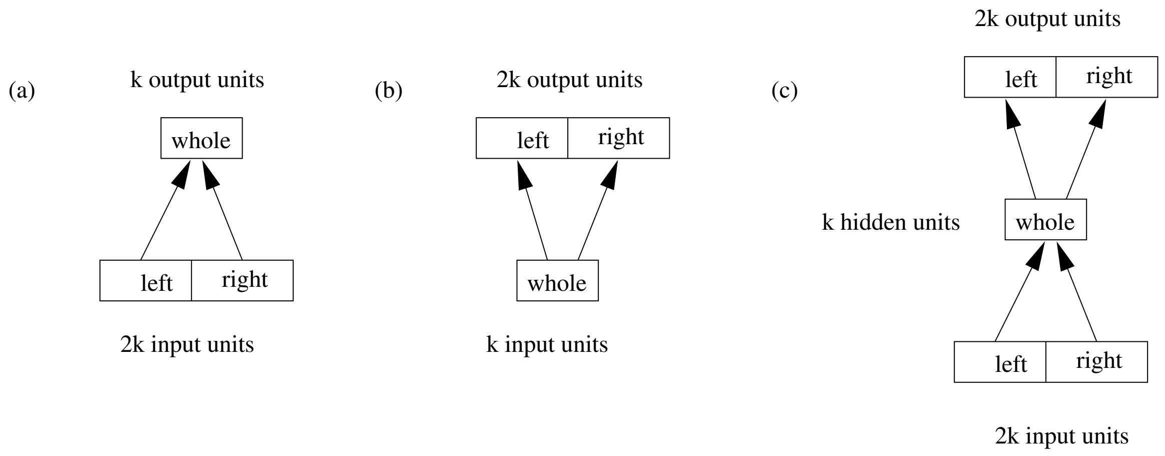


Figure 2.7: (a) Compressor, (b) Reconstructor, (c) RAAM architectures

2.3.2 Recursive Auto-Associative Memory

The Recursive Auto-Associative Memory (RAAM) [77] is designed to answer the question of how variable-sized recursive data structures, such as trees and lists, can be represented in fixed-width patterns. The RAAM automatically develops compact distributed representations for such compositional structures, as well as providing efficient accessing mechanisms for them.

In order to represent variable-sized symbolic sequences or trees, such as the binary tree illustrated in Figure 2.6, in a fixed-width form, Pollack defines two mechanisms:

Compressor which encodes small sets of fixed-width patterns into single patterns of the same size, so that the pattern (A,B) is encoded as R_1, the pattern (C,D) as R_2 and (R_1, R_2) as R_3. Note that the compressor is applied recursively.

Reconstructor which decodes these fixed-width patterns into facsimiles of their parts, and determines when the parts should be further decoded. Thus, R_3 is decoded to (R'_1, R'_2), R'_1 to (A', B') and R'_2 to (C', D').

The mechanisms for the compressor and reconstructor are assumed to be multi-layer perceptrons. Figure 2.7 illustrates (a) a compressor, (b) a reconstructor and (c) a combined compressor and reconstructor. The actual representations for the binary tree, the R'_i, are determined by the multi-layer perceptron itself, i.e. the actual representations are determined during training of the network to be the most suited for the encoding and decoding operations described above.

As an example, consider the representation of binary trees (which are one example of the more general class of DAGs referred to earlier). The RAAM is applied recursively as shown in Table 2.2. If the RAAM converges, then A' will become the same as A, B' as B, R'_1 as R_1 and so forth. This would imply that R_3 would represent ((A, B), (C, D)) because, firstly, the compressor would be a deterministic algorithm transforming the tree into this representation, and secondly the reconstructor

inputs		hidden	output	
A	B	R_1(t)	A'(t)	B'(t)
C	D	R_2(t)	C'(t)	D'(t)
R_1(t)	R_2(t)	R_3(t)	R'$_1$(t)	R'$_2$(t)

Table 2.2: Example of recursive application of RAAM to a binary tree

would be a deterministic algorithm transforming the representation back to a tree. A similar process is possible with sequences. The process is limited in extent only by the unavoidable inaccuracies in the equivalence of the quantities A and A' etc – these representations become equivalent only in the limit, and therefore infinite trees or sequences cannot be represented.

The interesting quality of the RAAM is that the representation for the binary trees and the mechanisms for encoding and decoding trees in this representation co-evolve. The final representation is trained as a good fit to the two tasks of encoding and decoding. It has also been found that some generalisation in accordance with systematicity is observed. That is, if the RAAM is trained to represent trees with constituents within one component of the tree, some generalisation of this representation to other components of the tree can be seen.

For example, Chalmers [7, 8] showed that the representations of the RAAM could be used in a further context, by training a feed-forward connectionist network to map between the active and passive forms of a sentence, based on the compressed representation. Chalmers trained the compressor to provide a compressed representation for the active form of a sentence, and the reconstructor to turn this compressed representation into its equivalent passive form. When testing the networks with novel sentences, generalisation of nouns across syntactic classes was observed. Chalmers argues that this ability shows how the network can work with semantic representations without extracting the syntactic constituents; an ability unique to connectionist systems.

In further work with this model, Sperduti has introduced the labelled-RAAM (LRAAM) [94, 95] which permits labels to appear at the vertices of the learnt structure. In addition, by including a single output unit after the compressor/reconstructor, the LRAAM model can be used for classifying structured representations. This has extended the range of domains to which RAAMs may be applied.

However, apart from the issue of representation, it is still unclear whether all the natural forms of syntactic generalisation required for handling language have been captured within the RAAM. Granted that the systematic substitution of nouns is observed, and some interesting properties of the connectionist holistic representation have been demonstrated by Chalmers, there is still the question of whether the RAAM can generalise across parse trees in a natural fashion. For example, given embedded clauses within a sentence, such as 'Mary loves John who sees Jane', it would be attractive to have a system which could generalise information learned about words in sentences with one embedded clause to those with more than one embedded clause. With the RAAM architecture, the nature of the recursive definition of the parse trees would restrict the accuracy of representations required to generalise to greater embeddings in this fashion. In spite of this, variants on the RAAM have featured in holistic parsers, which are discussed in more detail in Section 3.4.5.

2.3.3 Encoding networks

Section 2.2.2 discussed the Simple Recurrent Network (SRN) [18, 19] and its Backpropagation Through Time (BPTT) [85] training algorithm. The idea of BPTT is to unfold the SRN over the temporal structure of an input sequence. Recurrent links within the SRN pass activation across time

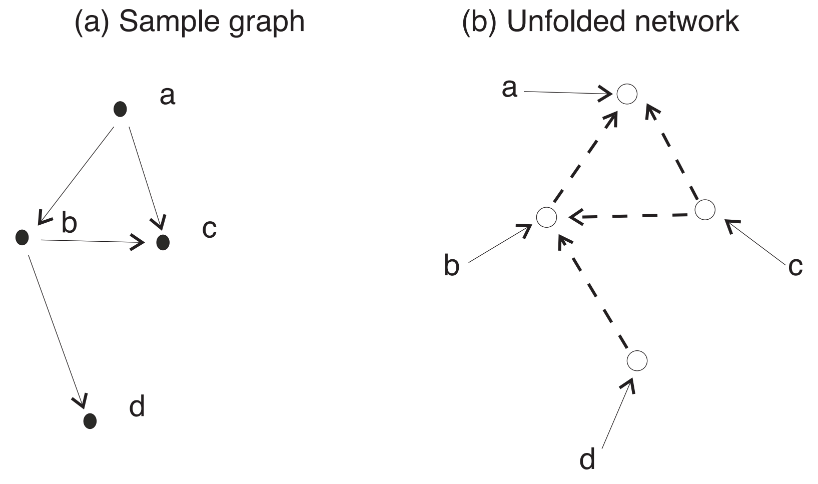


Figure 2.8: Unfolding a single generalised recursive neuron across a graph

periods. In the folded form of the SRN, this passing of information between time periods is achieved with context units, each context unit holding the activation of a unit in the previous time period.

A similar process is possible for arbitrary structured representations, but instead of unfolding over time (i.e. the elements in the sequence), the unfolding is done over structure (i.e. the elements in the input structure). In this case, BPTT becomes Backpropagation Through Structure (BPTS) [95, 26]. Also required is a change to the definition of the standard recurrent unit so that the recurrent links within the network extend across structure instead of time.

Sperduti and Starita [95] give a definition of such a unit, the *generalised recursive neuron* (GRN). The GRN differs from the standard recurrent unit (as described in Section 2.2.2) in that its recurrence is used for handling structure in the inputs. Thus, in its unfolded form, separate copies of the GRN are made for each node in the input structure. Activation is passed between the separate copies based on the links present between the different nodes in the input structure. This process is illustrated in Figure 2.8, which shows a sample graph and an appropriately unfolded GRN. The figure illustrates how one copy of the GRN (the circle) is made for every node in the graph. Dashed links connect the copies in accordance with the graph structure, and the solid links indicate where each node's label is input to the network. Note that the unfolded form is a feed-forward network, and hence may be trained using backpropagation.

A layer of GRNs can be used in place of the hidden units in a feed-forward network. In this case, the whole layer of GRNs is copied for every node in the input structure, and the links between the nodes pass activation from every GRN in the source node to every GRN in the target node. There is only one limitation to this approach to handling structure, and that is the assumption that every structure have a source (or root) node; this limitation is circumvented by algorithms, as described in [95], to add such a node to any structure without one.

Sperduti and Starita [95] discuss a number of training methods for networks using GRNs: the simplest being BPTS. There are two cases to consider. In the simpler case, where the input structure does not contain cycles, the unfolding process described above yields a feed-forward network. This network can be trained using standard backpropagation although, just as with BPTT, the separate copies of each weight must be made to agree with each other by adding together all the separate weight changes. In the more complex case, the input structure may include cycles, and so the above unfolding will not produce a feed-forward network. In this case recurrent backpropagation must be used, and details of this algorithm's application to GRNs is contained in [95].

The approach of GRNs is attractive in the way it redefines the SRN to handle structured repre-

```
S   ->  NP V NP
NP  ->  N | N RC
N   ->  Mary | Jane | Sally | Susan | Vicky | Fran | Abe
        | Bill | Carl | Dave | Earl | Fred
V   ->  likes | knows | treats | calls | draws | helps | races
        | sees
RC  ->  RP V NP
RP  ->  who
```

Figure 2.9: A recursive grammar

sentations. An arbitrary structured instance may be input to the network and a classification made based both on the content of the items within the structure, as well as the form of the structure itself. The class of applications for GRNs includes the classification of an input structure [95, 25] or conversion between equally structured representations [26]. Unfortunately, as the conclusion of [26] makes clear, these networks rely on the input and output structure being identical, which prevents them being applicable to the task of learning to output parse trees for a sequence of words.

2.3.4 Hebbian connectionist learning

Hadley and Hayward [32, 33] present an interesting connectionist architecture based on Hebbian learning. The network is designed to capture generalisations in accordance with a learning-based definition of systematicity known as *strong semantic systematicity*.

Strong Semantic Systematicity

Hadley [30, 31] defined *strong semantic systematicity* as a property of a network which can assign appropriate representations of meaning to novel sentences (both simple and embedded); a novel sentence is one which contains words in syntactic positions they did not occupy during training. For example, if a network is trained to identify the word "Mary" with the meaning "MARY" on a set of sentences in which "Mary" only appears in the subject position, strong semantic systematicity requires the network to assign the meaning "MARY" to the word "Mary" when used in the object position of a test set. Therefore, sentences of the form 'Mary loves John' can be used to train the network that "Mary" $\rightarrow$ "MARY", and this should be generalised to sentences of the form 'John loves Mary' or 'John loves Jane who knows Mary'.

This section continues with a description of a recursive grammar, which can be used to test for this property. This is followed by a discussion of the network structure and training algorithm. The experimental results obtained by Hadley and Hayward demonstrate their claim that the network satisfies the requirements of strong semantic systematicity.

Recursive grammar

Hadley and Hayward [32, 33] base their experiments on the context-free grammar of Figure 2.9. Example sentences are: 'Mary likes Bill', 'Mary who calls Susan likes Bill', 'Jane who calls Vicky knows Mary who likes Bill who helps Jane' etc. A parse tree for this last sentence is given in Figure 2.10. Note that the sentence begins with a symbol '<start>', which is used to introduce the sentence to the network, resetting activations where appropriate.

An embedding is a phrase such as 'who calls Vicky' which interrupts the simple sentence 'Jane knows Mary'. A multiple embedding occurs if the noun in the relative clause is followed by a

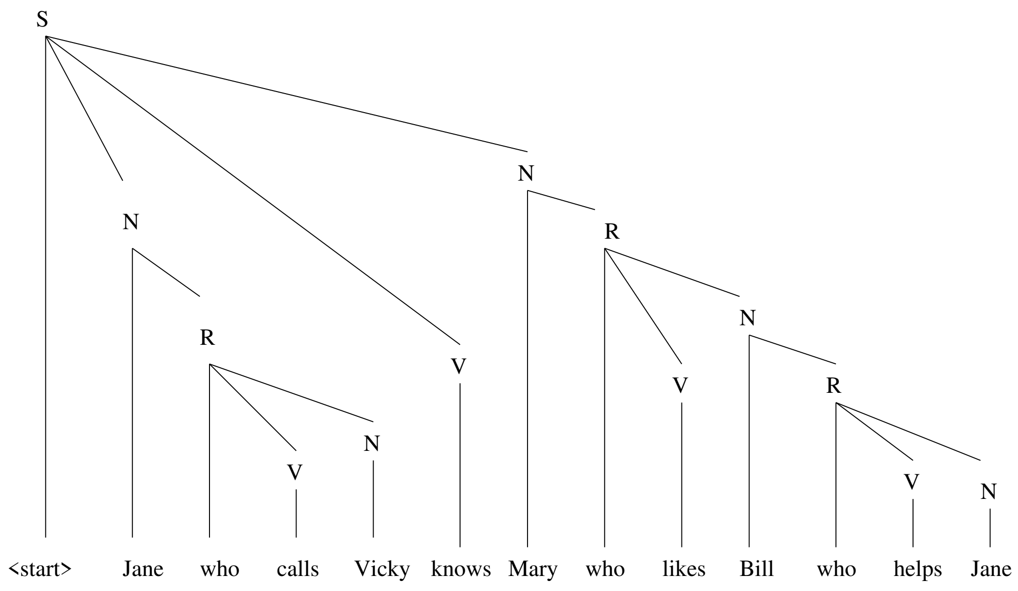


Figure 2.10: Parse tree for an embedded sentence

further "who", e.g. 'who likes Bill who helps Jane'. The complexity with this grammar lies in the possibility of multiple embeddings; there is no simple prediction, in terms of the number of words, of where the main verb is in a sentence. The multiple embeddings introduce a potential confusion in labels. When parsing the word "helps" in Figure 2.10, a decision is needed as to which relative pronoun it modifies. Three relative pronouns have been introduced at this stage, and the network representation must distinguish between them.

To test for strong semantic systematicity, a network is initially trained on a restricted sample of the sentences which may be generated from this grammar, and then tested on a broader sample. One demonstration of strong semantic systematicity involves training on sentences which have no embeddings, or a maximum of one. Further, the nouns within these sentences are restricted so that only one-third appear in all noun positions, one-third in only the subject position, and one-third in only the object position. Testing the network will involve increasing the number of embeddings the sentences may include (e.g. up to a maximum of three), and allowing all nouns to occupy the subject or object position without restriction. Therefore, the network will be tested on a significant fraction of nouns in syntactic positions which they did not occupy during training.

The experiments therefore test for the presence of two specific types of generalisation:

Substitution-type generalisations arise when nouns which appear in restricted syntactic positions of the training set appear in novel positions in the test set. The network is therefore tested on its ability to learn a class description, such as "noun", from instances of nouns, and substitute different members of this class at syntactic positions where one instance from this class has previously been observed.

Recursive-type generalisations arise when the level of embeddings is increased in the testing set relative to the training set. The network is tested on its ability to generalise from the attachment of a relative clause to a particular noun in the training sentences to attaching a relative clause to nouns of a more general description in the test set.

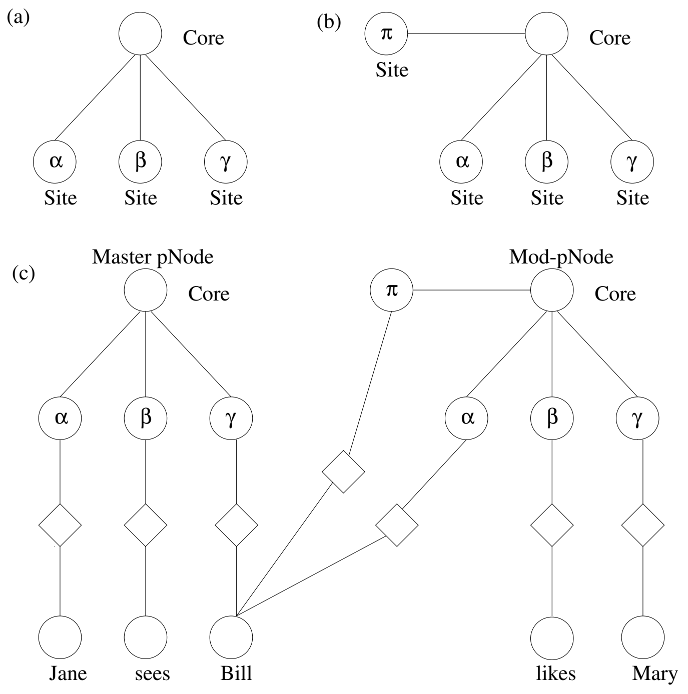


Figure 2.11: Representation of Hebbian connectionist network

A connectionist solution

Hadley and Hayward [32, 33] present a connectionist network which generalises learned information in accordance with strong semantic systematicity. Their approach combines connectionist methods with the classical insight that activating complex semantic representations entails activating their semantic constituents. The principal features of their model being:

- the model exhibits strong semantic systematicity,
- in training, two-thirds of nouns are *not* presented in all legal positions,
- the network is trained on a small subset of the possible corpus,
- training uses a variant on unsupervised Hebbian learning,
- after training, the behaviour of the network is transparent.

This section continues with a discussion of their model and training algorithm, and ends with results of their experiments with the recursive grammar from the previous section.

The network is designed to activate the semantic representation of a sentence, such as 'Jane sees Bill who likes Mary', when the words are presented on its input units. The overall structure of the network is, therefore, an input layer and a semantic (output) layer. The input layer is a linear array of 21 units, each unit corresponding to a single word. When a word is input, the corresponding unit is activated. The semantic layer, however, has considerable structure, and is composed of several types of unit.

The semantic layer is made up, primarily, from one 'Master pNode', as in Figure 2.11(a), and three 'modifier pNodes', as in Figure 2.11(b). The Master pNode is designed to represent the main proposition expressed by a complete sentence. In training and testing, its *core* becomes active to indicate that the Master pNode is the current proposition of the sentence under consideration. Its *sites* become bound to the constituents of that proposition; thus there are three sites, α for *agent*, β for *action*, and γ for *patient*. The modifier pNodes are designed to represent the subordinate proposition(s) expressed by a complete sentence, and have a similar structure to the Master pNode. Modifier pNodes have an additional site, π, which indicates a *modifier* site. Each site in the Master and modifier pNodes is known as a *thematic* site, because it identifies the thematic role of any bound word in the proposition.

Between the input and semantic layers there are two sets of connections: firstly, every input unit connects to the core of every pNode, and secondly, there is a layer of *concept* units. Every input unit connects to every concept unit. Connections between the concept units and thematic sites are mediated by *binding units*, which convey activation from the concept unit to the site unit. Concept units fall into two types, representing either actions or objects. The 'action concept units' are connected to every 'action thematic unit', and no other, i.e. every β site in the master and modifier pNodes. Similarly, every 'object concept unit' is connected to every 'agent or patient thematic unit', i.e. every α or γ site in the Master and modifier pNodes.

There are two aspects of the model of special interest from a connectionist perspective. The first is that the site units within each pNode form a competitive winner-take-all cluster, although the necessary inhibitory links remain at the virtual level (and are not shown). This means that, when deciding whether an input word could be bound to the agent or patient site, the one with the greatest activation 'wins', and the word is bound solely to that unit, leaving the other free to make another binding at another point. This implies that activation is only passed to this 'winning' unit, and not to any other. Similarly, the binding units compete with each other in trying to make the appropriate binding between the concept units and thematic sites. The second aspect of the model of interest is that, once a binding unit has entered into a binding, the concept and thematic sites it has bound 'drop out' of any subsequent competitions. Therefore, the model has built-in mechanisms for ensuring a localist representation of units is maintained, i.e. activation is not shared between units, but is restricted to the 'best' unit. Further, the binding units provide a mechanism for indicating a relationship between pairs of units. It is this enforced localist representation and clearcut binding mechanism which makes the network's behaviour transparent.

Figure 2.11(c) depicts the network after the sentence 'Jane sees Bill who likes Mary' has been presented on the input units. The diamond-shaped boxes indicate a binding from an input unit, through a concept unit, to a site unit. The Master pNode is shown with "Jane" bound to its agent site, "sees" to its action site, and "Bill" to its patient site. The modifier pNode has its modifier and agent sites bound to "Bill", which reflects the subordinate nature of the phrase 'who likes Mary' on "Bill". The modifier pNode's action and patient sites are bound to "likes" and "Mary" respectively.

Before considering the training of the network, we must first understand how a sentence is presented to the network. When each word of a sentence is presented, the corresponding input unit is activated. During presentation of a sentence, the Master pNode core is activated, to indicate that this is the proposition whose meaning is being considered. During training, this activation is enforced; once trained, the links from the input units to the pNode cores ensure the pNodes are activated during presentation of the input. If the sentence uses relative clauses, introduced by "who", then the modifier pNode core will be activated; for each relative clause, one of the three modifier pNodes will be selected at random for inclusion in the representation (the number of available modifier pNodes, three, is the restriction on the maximum number of embedded relative clauses any sentence may have).

Weighted links convey this activation between the core and sites in the pNodes, between the input units and pNode cores, and between the input and concept units. The binding units carry activation between the concept units and sites. A winner-takes-all competition within each layer ensures that only one concept unit and one site unit will predominate, bound by the binding unit. The trainable links are therefore those between the input and concept units, the input units and pNode cores, and between the core and site units within each pNode.

The links are trained using a variant on Hebbian training. Hebbian training updates the weight of a link based upon the activation of its source and destination units, tending to strengthen weights between simultaneously active units, and thereby discovering the strongest correlations. In this context, because spurious semantic units may become active during training, substantial weights may form for spurious correlations. Therefore, a better method is to integrate Hebbian learning with a simple form of competition.

The method used assumes that there is a fixed maximum (+1) for the sum of all weights on those links incoming to each unit. The initial value of the weights on these links is 0.001 (or a small random number). The weights are incremented on every link which connects that unit to an active unit. This occurs whenever there is an active input. The weights on each unit are incremented according to the following formula:

$$\begin{aligned} \Delta \quad &= \quad \frac{w}{\overline{w}} \times 0.0005 \\ \text{where} \quad & w \text{ is the weight of the link,} \\ & \overline{w} \text{ is the average of the weights on incoming links to the unit,} \\ & 0.0005 \text{ is a constant ensuring gradual learning} \end{aligned}$$

The ratio of the current value of the weight to the average of the weights ensures that 'above average' weights are rewarded. Therefore, as learning progresses, the winning links will accelerate their learning, and little weight is assigned to links reflecting spurious correlations. Also, if a unit has a number of equally significant correlations, none of the links predominates, as the ratio of each weight to the average remains near unity. Training is halted when all units have reached their maximum weightings, i.e. for every unit, the sum of weights on all the incoming links is 1.

The performance of trained networks as reported by Hadley and Hayward are very good. Sentences were formed as described above, so that the training set of 1370 sentences was formed from 75% of simple sentences, having no embedding, and 25% of complex sentences, having a single embedding. Additionally, two-thirds of the nouns were not presented in all legal positions, i.e. some nouns only appeared in the subject position of the sentence, and some only in the object position. Training typically finished before the first pass of the training set through the network was complete.

The network was tested on sentences using nouns in all legal positions, and, most importantly, with sentences extending to more than one embedding. The network performed perfectly on all the test sets, and this performance was confirmed for a number of different training sets. Therefore, the claim of Hadley and Hayward that their network satisfies strong semantic systematicity is justified.

Discussion

Hadley and Hayward [32, 33] have used a learning-based definition of systematicity, termed strong semantic systematicity, and have presented a network which generalises in accordance with this definition. Aizawa [1] addresses the question of whether this network counters the claim of Fodor and Pylyshyn [24] that connectionist networks cannot represent facts according to systematicity. The two sides of the debate can be characterised in that the connectionists are happy to *exhibit* a network with the desired property, whereas the critics require an *explanation* which is inherent to the definition of connectionism.

The criticisms rest upon the need for a connectionist network to be 'wired' to perform a particular function. The architecture selection and training mechanism for adjusting the link weights are integral elements of the connectionist framework. The justification for using a particular architecture and a particular training mechanism lies on empirical evidence, and not theoretical superiority.

Aizawa focuses on this element of choice. For example, if the precise form of Hebbian learning used by Hadley and Hayward was altered, the properties of the network would change. Similarly, if the nature of the competition between the binding units was altered, so would the properties of the network. And these changes would remove the ability of the network to manifest the property of systematicity. For systematicity to be a nomological necessity, such small implementational changes should not affect the presence of such a fundamental principle. Instead, systematicity should be present in the theoretical framework from which the learning algorithm derives.

A further criticism is that the network is preconfigured to work with one particular semantic representation of sentences. The construction of the pNodes, both master and modifier, is fixed from the start, and cannot be altered to match the training set. It would be a complex problem to extend the semantic representation to cover a large subset of natural language. Notable is the use of an 'object concept node' which is the source of the binding for any 'agent' or 'patient' thematic site. Any word which is an agent or patient of a sentence will activate an object concept node, and naturally generalise to the other kind of thematic site. According to Aizawa, the amount of structure within the network makes this architecture more classical than connectionist, as the network behaviour is sensitive to the structure of its representation; learning only alters the sequence in which the various constituents become active, to agree with the training set.

Hadley and Hayward argue that the structure and learning procedures of the network would, in a physical brain, have been evolved. Systematicity being a useful property, evolution would explain how a model capable of exhibiting that property does in fact possess it. Unfortunately, this argument is too powerful, as it may be used to explain almost any form of network with the desired property. This proposal therefore does not address the demand of the critics for an explanation for the property of systematicity inherent to the connectionist framework.

To conclude this section on Hadley and Hayward's model, the following points should be noted:

- Complex nodes are used for specific semantic properties and propositions.
- Substitution-type generalisation is achieved because the agent and patient sites in the pNodes are connected to a single 'object concept node'. Therefore, all words which can be an agent in the sentence, are also possible patients. Words where this is not the case, such as "he" or "her" must be treated by separate types of node.
- Recursive-type generalisation is achieved because there exist three banks of modifier pNodes, which enforce a limit to the number of embeddings which the system can represent. Generalisation across these three embeddings is achieved by the process of selecting at random the next modifier pNode to use. This naturally leads to problems if the number of embeddings can become larger (for instance, in a right-branching structure), and also leads to potential difficulties with sparse data problems; three copies require three times as much training for the same information to be encoded on every copy.
- The architecture and training mechanism are non-standard, in the wider field of connectionism, and it is unclear how these may be extended, not only to a wider linguistic capability, but also to other applications.

2.3.5 Temporal Synchrony Variable Binding (TSVB)

The connectionist networks described so far in this chapter have used the activation of individual units to represent features. For example, a unit labelled 'subject' will use an activation of 0 to represent the feature 'not subject', and an activation of 1 to represent 'is subject'. Missing from this representation is any means of identifying which entity the feature is referring to. This is an important limitation to the possible representations offered by connectionist networks, because such an identification underlies any ability to output structured information. The "binding problem" arises where it is necessary to represent multiple entities and multiple properties: a mechanism is required to represent which properties are bound to which entities. For example, on seeing two objects with the properties red, green, square and circle, some mechanism is needed to indicate which colour relates to which shape. One method is to provide for variables x and y to stand for the two objects. The scene may then be unambiguously described as: red(x) $\wedge$ green(y) $\wedge$ square(x) $\wedge$ circle(y).

In this section an extension to the standard connectionist framework is described which uses a different mechanism to solve this binding problem. The solution described here uses the synchrony of activation pulses to represent entities; if two units are pulsing synchronously then they are representing properties bound to the same entity. This proposal was originally made on biological grounds by Malsburg [99]. Shastri and Ajjanagadde (S&A) [92] developed a computational implementation of the same ideas, which they termed Temporal Synchrony Variable Binding (TSVB). Using this implementation, S&A constructed a connectionist model of reflexive reasoning, SHRUTI, which represented dynamic variable bindings as the synchronous firing of appropriate nodes in a network, and used these bindings to rapidly make a range of inferences from a large body of background knowledge.

This section explores the basic principles required to implement TSVB connectionist networks, and discusses some of the advantages such a representation confers in terms of an ability to automatically make certain generalisations.

Basic principles behind TSVB

TSVB is a mechanism for enabling features to identify which entity they are referring to by using the temporal synchrony of unit activations. The implementational requirements of this mechanism are encapsulated in the following three principles:

- Each time period is divided into discrete phases, each phase to represent a distinct entity.
- Pulsing units compute within each phase independently of other phases, and so compute information about distinct entities.
- Non-pulsing units compute across all phases, combining information about several entities.

S&A, in SHRUTI, provide one implementation of these ideas in a connectionist model of symbolic reasoning. Because S&A's primary concern is an ability to do symbolic reasoning, the model cannot learn and all features are assumed to be true (activation 1) or false (activation 0). S&A define several types of unit, but the two which most closely fit the pulsing and non-pulsing units described above are as follows:

ρ-btu A binary-threshold unit has an activation of 1 if, in the current phase of the previous period, the number of active inputs equals or exceeds the threshold, otherwise its activation is 0. For example, a ρ-btu unit with three inputs, a, b, c and a threshold of 2 will behave as in Figure 2.12(a). The output is active in phase 1 of period 1 because a and b were active in phase 1 of period 0 etc. This type of unit is a 'pulsing unit', because its output holds for a single phase only.

τ-or A temporal-or unit is a 'non-pulsing unit'. Its output in the current time period is a 1 if the number of active inputs in the previous period, irrespective of phase, equals or exceeds its threshold, otherwise activation is 0. For example, a τ-or unit with three inputs, a, b, c and a threshold of 2 will behave as in Figure 2.12(b).

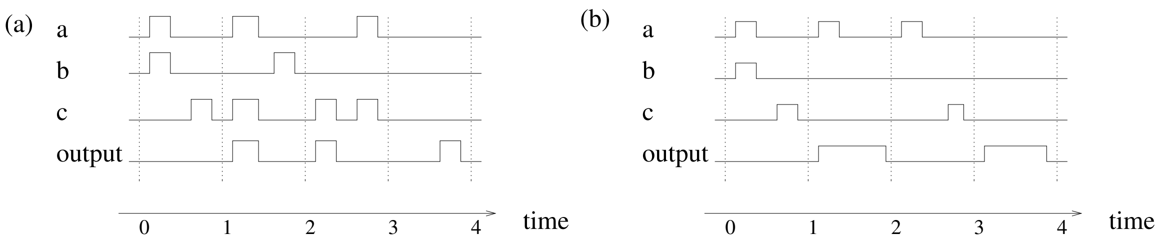
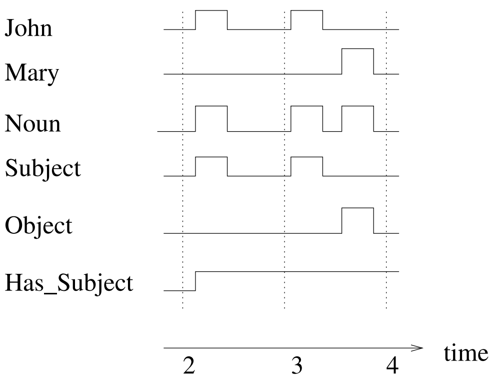


Figure 2.12: (a) Behaviour of a ρ-btu unit. (b) Behaviour of a τ-or unit.



In the first time period

the network represents the proposition:

John(x) & Noun(x) & Subject(x) & Has_Subject

In the second time period, it adds:

Mary(y) & Noun(y) & Object(y)

Figure 2.13: A network trace.

The use of synchronous activation pulses to bind features to entities can be illustrated with a simple example. Figure 2.13 shows a trace of activation values within such a network over two time periods. The network is parsing a sentence such as ‘John loves Mary’, and only the nouns are shown. For the two time periods, two *phases* are shown, illustrating the subdivision of periods into phases. “John” has been input in the first phase in the first time period, and “Mary” is input in the second phase of the third time period. Each of these words has the feature “noun”, and so activation appears on “Noun” in synchrony with that on “John” and “Mary”. “John”, as the first noun, is further described as a “Subject”, and “Mary”, as the second noun, is the “Object”. The synchronicity of the pulses makes it clear that the feature “Subject” refers *only* to “John”, and “Object” only to “Mary”. Therefore illustrating the property that the temporal dimension allows features to identify the entity (or word) they describe through synchronicity of activation pulses. Finally, the feature “Has_Subject” is an example of a global feature. It does not describe any entity in particular, but is true for all entities, and therefore its activation is true across the entire time period. Such features are necessary to pass information across phases. The last point to note is that no information is supplied by the position of the phase in the time period, or its relative ordering to other phases; it is only necessary for “John” and “Mary” to be allocated distinct phases for features to distinguish between them.

The use of time to represent variable bindings, and the ability to represent logical propositions with the ρ-btu and τ-or units, mean that a wide range of complex symbolic systems can be implemented in a TSVB connectionist framework. S&A proposed SHRUTI, a model of reflexive reasoning, to demonstrate how a large number of inferences can be made in parallel from the knowledge held in a background database. The speed with which an appropriate inference can be made from large amounts of background information using this system offers a plausible model of reflexive reasoning. This representational power has also been used to construct a parser for natural language [37], a model discussed in more detail in Section 3.5. However, neither of these models uses trainable TSVB networks, and this is because the implementation of TSVB in terms of binary-threshold units does not enable standard connectionist training techniques such as backpropagation to be applied. Instead, Shastri [91] provides a trainable architecture for SHRUTI based on the GRADSIM algorithm [100], which learns relational rules from a toy domain. The next chapter will describe a novel implementation of TSVB enabling standard connectionist training algorithms to be used. Before considering this, one important benefit of the use of TSVB must first be discussed.

Inherent generalisations in TSVB networks

The implementation of TSVB networks in [92] and [36], uses TSVB networks as an implementation of equivalent symbolic systems: systems for representing and manipulating structured information. This representational ability, however, is not specific to TSVB networks. Tesar and Smolensky [98] argue that TSVB can be interpreted as an implementation of tensor product variable binding (TPVB), using time rather than space. The use of TPVB has already been illustrated in Section 2.3.4 with the discussion of the Hebbian network of Hadley and Hayward [33]; their binding units mediate between the concept and action nodes, and so represent the relevant variable bindings. As was pointed out by Henderson [37], this equivalence between TSVB and TPVB is true only in the context of static representations. When learning is taken into account, the use of TSVB means that information learned about an entity in one binding is inherently generalised to other bindings of that entity, while this is not true of TPVB.

For example, consider a TSVB network parsing the sentence ‘John loves Mary’. The word “John” is the subject of the sentence, but is also a noun. Within the TSVB network, this binding is represented through temporal synchrony, i.e. the unit which represents “John” as a noun is active in synchrony with the unit which represents “John” as the subject; this is illustrated in Figure 2.13. Because computation is performed in the network by passing activation along the links in the network, the computations leading to these facts will be the same at all times, because the same link weights will be used. Thus the computation which leads to “John” being described as a noun will occur independently of the other features bound to the same phase, such as whether “John” is the subject or object. In other words, learned information is independent of variable bindings, and is therefore inherently generalised across all entities in the network. As Henderson [37] has shown, this generalisation ability explains, to a large extent, the properties of systematicity argued for by Fodor and Pylyshyn [24] and elaborated by Hadley [30, 31].

By contrast, with TPVB, links trained between the word and its binding unit will not generalise to links between that word and other binding units. As discussed in Section 2.3.4, Hadley and Hayward [33] include special mechanisms within their architecture (competition between units, random selection between binding units, and the “drop-out” of bound units from future competitions) to ensure that the binding units operate as variable bindings with the appropriate generalisations. Looking ahead to the TSVB networks defined in Chapter 4 of this thesis, in Section 4.2.2 the networks are tested on the same recursive grammar as that used in Section 2.3.4 for testing the Hebbian network. Comparing the results, it is evident that, although the TSVB network and Hadley and Hayward’s architecture provide for equivalent abilities, TSVB networks do so with a more parsimonious architecture.

2.3.6 Comparison

Given the above approaches to handling structure within connectionist networks, it is time to consider their potential for natural language parsing. The first point to note is that there is a difference between handling structured information on the *input* and generating structured information on the *output*. For parsing, it is necessary to map an input sequence of words into a parse tree, which is a mapping from a sequence to a form of directed acyclic graph, i.e. sequence $\rightarrow$ DAG. As is apparent from Table 2.1, the only architecture that directly handles problems such as parsing is the Hebbian connectionist network [32, 33]. However, this model is limited in its generality by the large amount of internal structure required for handling even a small range of sentence types. The RAAM class of models is a further possibility, capable of packing an arbitrary input structure into a distributed representation, and later unpacking that representation into a possibly different structure. In spite of the additional burden of creating this third representation, the RAAM has proven a popular model for parsing, as is discussed in Section 3.4.5, although their success has been confined to rather limited toy grammars. In addition, the need to pack the output parse tree into a distributed representation requires the parse tree to be represented in sequential form. This causes limitations in the form of parse tree, e.g. to being of fixed valency [42], as well as in the ability of the network to recognise structural regularities due to their implicit encoding within the representation.

The introduction of Temporal Synchrony by Shastri and Ajjanagadde [92] has led to some implementations of powerful symbolic systems [36, 92]. The technique additionally holds the promise of satisfying the dictates of systematicity [37] whilst maintaining many of the standard training and definitional practices of the connectionist community. This is particularly the case because the use of phases within such a network means that the structural properties of the output can be represented independently of the information about particular nodes within that structure, which may be held in the usual distributed activation across units. However, SHRUTI [92] has only been trained in a limited domain [91], and that with a highly structured architecture and non-standard training algorithm. Chapter 4 therefore considers TSVB further and develops a novel implementation with a view to constructing an appropriate training algorithm. This has the dual motivation of providing an effective training algorithm for TSVB networks, as well as developing a suitable model for the task of learning to parse.

The preceding material has considered a number of networks for handling structured representations. However, there is a wider range of connectionist style models for handling structured representations, better described as *hybrid* models because they incorporate elements from both connectionist and symbolic approaches [66]; hybrid models form an interesting alternative to the purer connectionist models considered within this thesis ([101] contains an up-to-date summary of such approaches). As an example, one of the more important sources of structure in AI and cognitive theories has been that of the *schema* or frame [71]. Essentially the schema provides a definition of a concept or rule which contains symbolically defined information, such as instantiated attributes. Sun [97] provides a computational implementation for symbolic structures such as schemata. Included in the implementation are mechanisms for representing dynamic variable binding and super- and sub-structures. However, the implementation suffers from being just that, an implementation of a pre-existing rule set into a connectionist framework. Therefore, just as with SHRUTI [92], although it is demonstrated that a connectionist network can handle large amounts of structured information, no satisfactory algorithm has been proposed for how the network could obtain that information from a training set.

2.4 Conclusion

This chapter has described connectionist networks in terms of three broad classes of problem. Firstly, the standard feed-forward networks, appropriate for learning about static patterns. Then the recurrent and time-delay networks, which could learn about patterns extending across time. Finally, a broad range of networks for addressing the problem of representing structured information, or multiple entities, within a connectionist framework.

In order to apply connectionist networks successfully to learning in complex domains such as natural language processing, all three of these problem classes must be addressed. However, as shown by Table 2.1 and the discussion above, no suitably general architecture has been proposed which successfully combines all these properties. This theoretical finding is reinforced in the next chapter where a number of approaches to parsing are considered, with the approaches drawn from classical symbolic, statistical and connectionist frameworks. The fact that connectionist networks do not learn to parse in comparable fashion to statistical parsers may be attributed to their poorer use of appropriate representations. This problem is addressed in Chapter 4, where two of the approaches described in this chapter are combined to form a new trainable connectionist network. These are the Simple Recurrent Network (SRN) and Temporal Synchrony Variable Binding (TSVB). These particular extensions have been chosen because they are indeed *extensions* to the basic feed-forward connectionist network. Therefore, greater power can be expected from a connectionist model incorporating them both.

The SRN is a network which can learn about patterns across time. It consists of a feed-forward network, which can learn about static patterns, augmented by the addition of context units. As was shown in Section 2.2, this approach is more powerful than Time Delay Neural Networks, because it can generalise in testing to sequences with dependencies of greater length than those used during training. The SRN also has a simple training algorithm which is applicable to arbitrary lengths of input sequence.

In a similar vein, TSVB is the most attractive of the various approaches for representing entities within a connectionist framework. For instance, where other approaches, such as the RAAM architecture, rely upon an encoding of the structure of a sentence within a standard connectionist representation, TSVB extends the definition of connectionist units to include a further dimension, time. Not only can a TSVB network convey information by activating a specified unit within the network, it can also convey information by controlling the time of its activation. This provides the considerable potential to not only learn about structured information, but also to inherently generalise information across that structure. The nature of TSVB's use of temporal synchrony for handling structure should be distinguished from the use by encoding networks [95] of the generalised recursive neuron: encoding networks handle an *input* structure, temporal synchrony is used both to accept and to *output* structure.

Although SHRUTI [92] has been trained in a limited domain [91], its architecture is still highly structured and relies on a non-standard training algorithm. The primary contribution of this thesis is contained in Chapter 4, with a demonstration of how TSVB units can be redefined in a standard connectionist manner, and then added to the architecture of an SRN. This new implementation of TSVB produces a range of network architectures which inherit the training algorithms and accumulated experience of standard connectionist practice, but with the added bonus of structured output representations. But before doing this, it is necessary to examine earlier work in parsing and so understand the ability and limitations of current systems for learning to parse.

Chapter 3

Previous Work on Parsing

One of the fundamental problems in applying computers to natural language is the inherent complexity of speech, the raw physical medium of human language. In order to begin, it is necessary to decide where on the spectrum of possible 'natural language processes' the computer model is to lie. Work has been done through the entire spectrum from the extraction of phones from sound [82], to the ordering of documents relating to their semantic content [51].

In this thesis, the use of natural language as an application for Simple Synchrony Networks (SSNs) is motivated mainly by the desire to test the SSN on a real world problem requiring the output of structured information. The task is to learn a mapping from a sentence, presented to the computer as a sequence of discrete words, to a representation of its syntactic structure, in the form of a parse tree. Although this treatment of language may be criticised as a cognitive and linguistic over-simplification, there are some points in its favour. Firstly, a large amount of stored information is in textual form, and it would be useful to have robust computer models for extracting its structure. Secondly, it may be argued that the inherent complexity in understanding language is contained precisely at this level of processing sentences to extract their structure, and so a computer model of learning to extract such information would be of interest to linguists. And finally, it has become a standard task for the statistical language community, and so is a suitable one for the SSN to demonstrate its potential for connectionist language learning; if this potential is realised, then the cognitive or linguistic plausibility can be developed.

This chapter discusses some previous work on parsing, and begins with a description of the problem of parsing. The earliest attempts at parsing involved constructing a symbolic processor for applying a set of grammatical rules which transform a sequence of input words into a parse tree. These parsers are typified by the PARSIFAL model of Marcus [63], and are discussed in the second section. The symbolic-style parsers had one major failing in that they rely on a pre-programmed set of grammatical rules. The most successful parsers which learn their parsing knowledge from a set of pre-parsed examples are found in statistical language learning, and a typical example is the Probabilistic Context-Free Grammar (PCFG). Statistical models of learning currently predominate in the area of natural language learning. Their strength rests on an appropriate representation of the grammar of a language combined with a careful collection of relevant statistics. The third section addresses a range of connectionist techniques for language learning, mostly based around the Simple Recurrent Network (SRN), as described in Section 2.2.2. The SRN is a popular choice for connectionist language learning because the SRN can learn to recognise patterns across time. In addition, the SRN forms the architectural framework of the SSN developed in this thesis, and therefore the ability of the SRN in acquiring complex representations for natural language should be considered before observing the benefits of adding Temporal Synchrony Variable Binding (TSVB).

Finally, this chapter describes a parser for TSVB connectionist networks developed by Henderson [36]. This parser uses a TSVB connectionist network to encode rules of sufficient form and

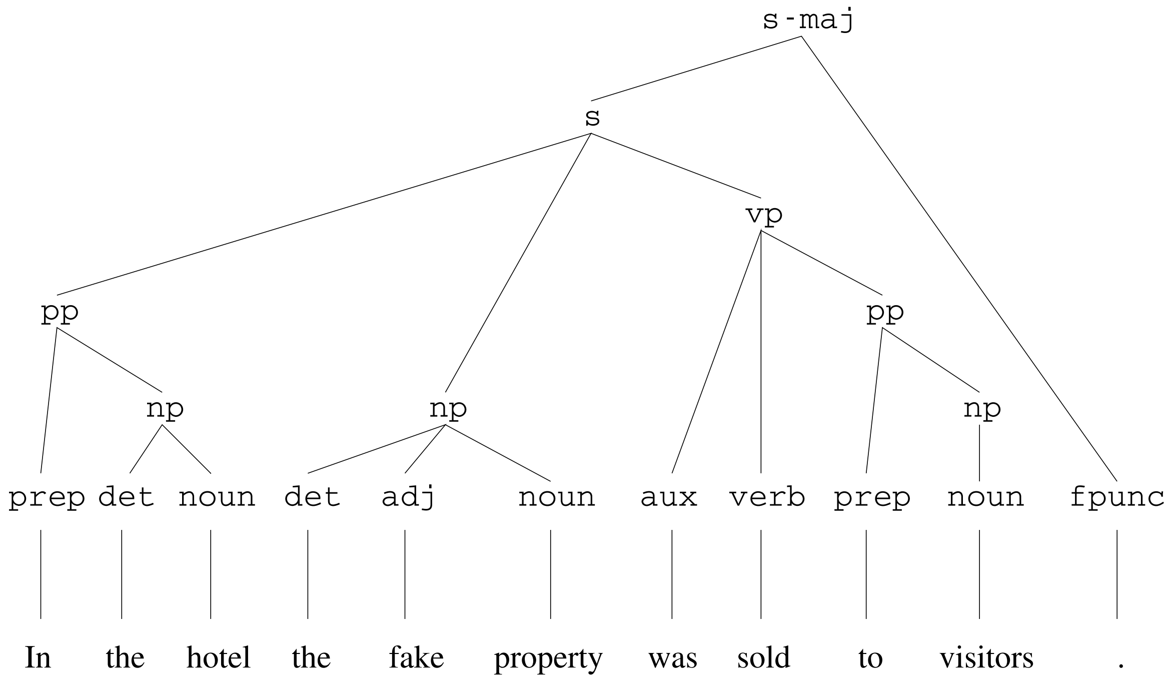


Figure 3.1: A syntactic structure.

complexity for parsing a significant subset of English. This demonstrates that TSVB networks have the representational power, in principle, to learn to parse samples of natural language. However, the ability of the SSN to achieve this representational promise is a function of the efficiency of its training algorithm and the quality of the input data.

This chapter therefore sets out to describe the current abilities of learning algorithms within the area of language parsing, and also establish the promise of TSVB networks for this task. The next chapter develops the SSN model, and then Chapter 5 describes experiments applying the SSN to learning to parse natural language.

3.1 The Problem of Parsing

Parsing is the process by which a sentence is converted into a hierarchical representation of the various relations between words within the sentence: this hierarchical representation is known as a *parse tree*. The parse tree is a representation of the *syntactic structure* which a parser extracts from a string of words. The parse tree is intended to contain much of the information which a human speaker appears to recognise in a given sentence such as: which noun phrase is the object of the main verb, and which noun does a given prepositional phrase modify. An example of a parse tree is given in Figure 3.1 (taken from [9]). The words for the sentence appear at the base of the tree, and each has an associated word-tag (or terminal symbol). The internal nodes of the tree identify the constituents forming the hierarchical structure of the sentence.

Some of the knowledge about this sentence represented within the parse tree is that: “in the hotel” does not modify “property” but “sold”, “the fake” modifies “property” and not “hotel”, and the “visitors” were buying property, not selling it.

The problem of learning to parse may be defined as follows. Given a set of training examples, consisting of sentences and their associated parse trees, formulate a mapping between the two so that parse trees may be constructed for novel sentences. The actual form of the mapping varies depending on the underlying representation used for the parse tree, and how the parsing process itself uses this representation in conjunction with the input sentence to produce an output.

Two distinct strategies may be identified. First, and most popular within the psycholinguistic tradition, has been an *incremental* strategy. In this case, the parse tree is created as each word is input (e.g. see [27, 50]), and the ability to backtrack to recompute earlier parts of the parse tree is restricted. PARSIFAL [63], considered below, is a classical example of a parser following this tradition. This incremental approach to parsing is attractive because it allows the meaning of the sentence to be calculated as the sentence is heard. However, not all sentences can be parsed without requiring information about later words to correctly categorise the current word.

The second strategy is to read the whole sentence into the parser's internal representation, and then compute an output parse tree from this representation. This approach is adopted by the holistic parsers [44] discussed below in Section 3.4.5. It has the disadvantage of not directly relating to psycholinguistic ideas of incremental processing. To its advantage though, this approach creates its parse tree from all the information available in the sentence. In addition, other forms of output may be generated from the same internal representation. For example, conversion may be made between active and passive sentence types [7, 8], or even between languages [14].

3.2 Symbolic Parsing: PARSIFAL

PARSIFAL [63] is an example of a classical parser motivated by considerations of syntactic competence. Previous parsers had adopted a highly search-oriented process, necessitating large amounts of backtracking (e.g. Augmented Transition Networks [106] and discussion in Chapter 6 of [2] and Chapter 9 of [105]). The major contribution of PARSIFAL was to substitute a limited amount of look-ahead for this search process, accepting that sometimes this process would lead to mistakes; the sentences that were misparsed could then be compared with those which humans have difficulty with, and so lead to tests of the parser as a model of syntactic competence.

PARSIFAL consists of three major elements. A *buffer* contains a set of input words, the *parse stack* maintains a list of constituents which the parser is currently processing, and the *rule-set* contains the set of operations which the parser can apply to the information within the buffer and parse stack. Three basic operations may be applied by the rule-set. These are to *create* a new node on the parse stack, to *attach* an input constituent to the top node of the parse stack, and to *drop* the top node of the parse stack into the buffer.

The limited look-ahead employed by the parser is due to its ability to use the first three words on the input buffer when determining which rule to use from its rule-set. This allows PARSIFAL to dispense with search, although certain sentences, in which the necessary information is beyond the three-word horizon, will not be parsable. Such examples include the familiar garden-path sentences, in which an initial sequence of words may form part of a question or imperative, e.g. 'Have the students who missed the exam take it today', 'Have the students who missed the exam taken it today?' [2]. The claim of PARSIFAL is that the problems caused by the limited look-ahead within the parser are analogous to the difficulties observed in humans processing the same sentences, making PARSIFAL an interesting model of syntactic competence.

3.3 Statistical Learning: Probabilistic Context-Free Grammars

The Probabilistic Context-Free Grammar (PCFG) is an example of a parser taken from statistical language learning. In Chapter 1 some of the differences between statistical and AI learning techniques were discussed. The most important of these being the emphasis of statistical learning on performance, with the consequence that statistical techniques tend to be specialists at their target domains. PCFGs provide simple statistical models of natural language and their parsing schemes

Non-terminal (symbol)	Examples
major sentence (`s-maj`)	‘In the hotel the fake property was sold to visitors.’
sentence (`s`)	‘In the hotel the fake property was sold to visitors’
verb phrase (`vp`)	‘was sold to visitors’
noun phrase (`np`)	‘the hotel’, ‘the fake property’, ‘visitors’
prepositional phrase (`pp`)	‘In the hotel’, ‘to visitors’

Table 3.1: Example non-terminal symbols.

often perform as well as other simple broad-coverage parsing systems for predicting tree structure from part-of-speech tag sequences [10, 48]. The other popular model in statistical language learning is the Hidden Markov Model, which, however, is used more for assigning word-tags to words; the PCFG is used to assign syntactic structures to these word-tags.

This section describes how the PCFG represents parse trees, its evaluation and levels of performance. The description of the algorithm is largely taken from [9] and the results from [11, 48].

3.3.1 Context-free grammars

The *context-free grammar* (CFG) is a collection of rules which specify the permitted range of structures within a language. It consists of the following (examples are taken from Figure 3.1):

- A set of *terminal symbols* – the words and punctuation appearing at the leaf nodes of the parse tree, e.g. “In”, “the”, “hotel”, etc.
- A set of *non-terminal symbols* – the parts of speech labelled `prep`, `verb` etc.
- A specific non-terminal designated as the starting symbol (`s-maj` in the example parse tree).
- A set of *rewrite rules* – each with a single non-terminal on the left-hand side and one or more terminal or non-terminal symbols on the right.

Some example non-terminal symbols and rules are given in Tables 3.1 and 3.2 respectively.

A sentence is described by specifying which of the rules and non-terminal symbols in the CFG account for the words in the sentence. Thus, for Figure 3.1 and Table 3.2, the first rule in the table is the only one with the starting symbol `s-maj`. The `fpunc` is the symbol for the full-stop in the sentence. The `s` is a further non-terminal symbol, and offers the expansions `np vp` or `pp np vp`; in this example the latter is the one to select. It can be seen that the structure of the sentence in Figure 3.1 gradually unfolds until all the words have been accounted for.

However, the CFG also has some significant limitations as a representation for natural language. These limitations include:

ambiguity with CFGs can arise when more than one sequence of rewrite rules may generate the same sentence. This leads to more than one possible interpretation of a given sentence, e.g. in the classic “Swat flies like ants”. Is the interpretation that flies are to be swatted like ants, or that swats fly in the same manner as ants (i.e. is ‘fly’ a verb or a noun)? Either or both is a possible outcome based on plausible sets of rewrite rules.

inadequacy with CFGs is present because they do not capture some of the more prevalent features of English, such as the agreement between a verb and its noun. To enforce agreement in the

s-maj	$\rightarrow$	s fpunc	det	$\rightarrow$	the
s	$\rightarrow$	np vp	noun	$\rightarrow$	hotel
s	$\rightarrow$	pp np vp	prep	$\rightarrow$	In
vp	$\rightarrow$	verb	adj	$\rightarrow$	fake
np	$\rightarrow$	det noun	fpunc	$\rightarrow$	.
pp	$\rightarrow$	prep np	noun	$\rightarrow$	property
vp	$\rightarrow$	aux verb pp	verb	$\rightarrow$	sold
np	$\rightarrow$	det noun noun	noun	$\rightarrow$	hotel
np	$\rightarrow$	noun	aux	$\rightarrow$	was

Table 3.2: Some context-free rules.

rewrite rule `s` $\rightarrow$ `np vp` would require a duplication of the rule, one forcing the nouns and verbs to be selected from singular examples, and a second forcing the selection to be made from plural examples.

A further problem lies in the use of long-distance dependencies, such as in 'Whom did Fred give the ball to?'. The "Whom" is the noun phrase from the prepositional phrase "to whom" at the end of the sentence. Thus, there is a dependency between the "to" and the "whom", which is at a long-distance, because the two words are at opposite ends of the sentence. These dependencies are also not captured by the standard CFG.

Some of these problems can be addressed by including the probability of a rule along with its definition, which leads to the definition of a Probabilistic Context-Free Grammar.

3.3.2 Probabilistic context-free grammars

The basic form of a rule in a CFG is $A \rightarrow \psi$, where A is a non-terminal symbol and ψ is a string of terminal and non-terminal symbols. The PCFG, as the name suggests, extends this representation to include the probability that the rule will occur: so the rules in a PCFG are of the form $P(A \rightarrow \psi|A)$. The probabilities are therefore arranged so that, for each non-terminal symbol, the sum of the probabilities of all the rules with that symbol in the left-hand side is 1. The probabilities mean that the likelihood of any given parse of a sentence can be calculated with respect to that grammar, by multiplying out the individual probabilities for each rule that is applied.

When training the PCFG, the initial step is to use the training corpus to create an actual grammar in PCFG form. One rather crude way to do this is to take each of the non-terminal symbols in the training corpus and produce a rule for that symbol. For example, if the first sentence has a node `s` with nodes `np` and `vp` as its immediate children, then the grammar needs the rule `s` $\rightarrow$ `np vp`. Having read off all the rules, the probability of each rule applying at any stage can similarly be calculated, according to how often each is used in the corpus.

The effectiveness of PCFGs in learning to parse is based on their use of probabilities in the rewrite rules, which enables the parser to accommodate itself to the statistical regularities in the corpus. This factor can be used in a number of techniques to provide better parsers than the simple one described above. One technique further constrains the parser to select the grammar which maximises the likelihood of the parses found in the training corpus. The usefulness of this constraint is illustrated by considering the following trivial grammar for English.

`s = word s`
`s = word`
`word =` w_1

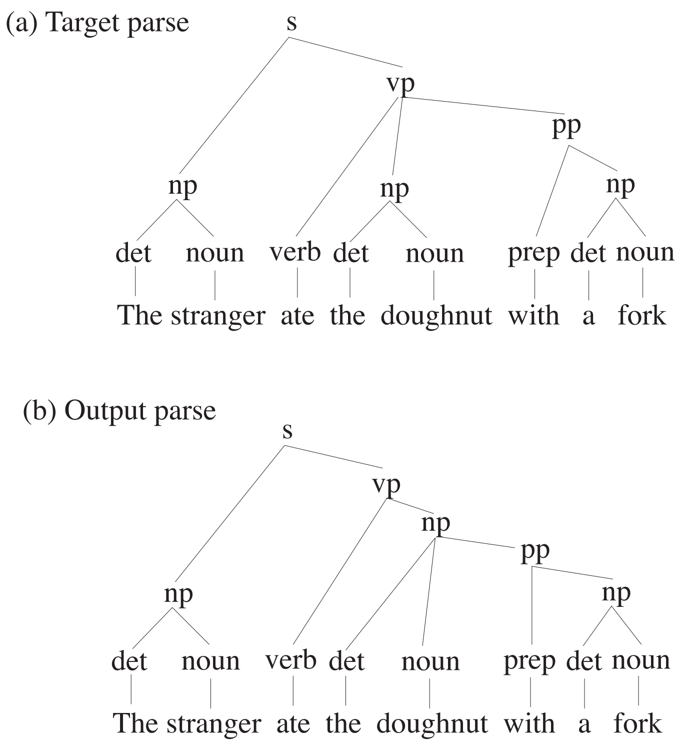


Figure 3.2: A comparison between (a) target parse tree, and (b) output parse tree.

`word` = w_2
`etc`

This grammar, because the word "the" is the most common word in English, predicts that the sentence "The." is the most likely English sentence! However, the constraint given above would reject this grammar, because it assigns high probabilities to sentences that do not occur in the training corpus, and must therefore assign low probabilities to those that do. Thus, the learning process can reject the trivial grammar compared to a better one, because it is less likely to generate the training set. (With the crude selection of a grammar used in the simplified PCFG training model given above, the initial grammar is instead forced to be a reasonable model of the training data, because the grammar is directly derived from the data.)

One useful feature of the PCFG is that, because the CFG is not a perfect model for English, and due to the flexibility of English constructions, some sentences in the corpus may not conform to the constructed grammar. This may be dealt with by allocating such rare sentences a low probability of outcome. Thus, the grammar need not reject such sentences, but their scarcity can be reflected in the low probability of their being parsed.

3.3.3 Evaluation of parse trees

Once the PCFG has been trained, it can be tested on a selection of unseen sentences. For each sentence in the test set, the parser is applied to find the most likely parse for that sentence. Typically, this stage can require searching through approximately a million possible parses for each sentence [11], and the one with the greatest likelihood is selected as the parse for the given sentence. The parser's predicted parse trees for this test set are then compared to the target ones, and the performance of the parser is evaluated based upon how close the parse trees produced by the parser match the targets.

In order to demonstrate how the evaluation process operates, consider the two parse trees in Figure 3.2: (a) is the target output for the parser, and (b) is the actual output from the parser. The question is: How best to quantify the degree of similarity and differences between them? The standard method is to measure the *precision* and *recall* on constituents.

The precision is the number of correct non-terminal constituents found by the parser (summed over all the sentences in the test set) divided by the total number of non-terminal constituents output by the parser. The recall is the number of correct constituents divided by the number in the target parse. A constituent is considered correct if it starts and finishes at the same place, and is labelled with the correct non-terminal symbol. Note that the terminal symbols are not included in the evaluation. (Precision may be compared with the standard measure of 'errors of commission'; and recall with the standard 'errors of omission'.)

For example, in Figure 3.2 there are 6 constituents in the output parse tree (a) (one `s`, three `np`'s, one `vp`, and one `pp`). The only incorrect one is the `np` headed by "doughnut", which should have ended after "doughnut" but instead ends after "fork". Thus the precision is 5/6, or 83%. As there are also 6 non-terminals in the target parse, the recall is also 83%. Note that the parts of speech (word-tags) are included with the terminal symbols and so ignored in computing the precision and recall. This distinguishes the parsing accuracy from the part-of-speech tagging accuracy [11].

Parsers which are given just the parts of speech, and asked to parse the sentences (i.e. without seeing the actual words), achieve about 72% average precision/recall [48]. Better results are achieved by giving the parser access to the words and using fine-grained statistics on how particular English words fit into parses. For such parsers, precision/recall is of the order of 87-88% [11]. These results are achieved using the Penn Wall Street Journal corpus [64], containing approximately 50,000 sentences of average length 23 words (roughly one million words in total).

3.3.4 Strengths and weaknesses

The most important restriction on the PCFG model lies in the construction of the initial grammar from the training part of the corpus. In the experiments described by Charniak [11] this procedure was achieved in the relatively crude manner described in Section 3.3.2. More sophisticated techniques exist, but rely on adding greater linguistic knowledge to the parsing model, increasing its specialisation. These parsers, although achieving good results with one corpus, do not always produce equally good results on further corpora.

However, it is also true that the PCFG model provides excellent results in the domain of learning to parse natural language. This is partly because, in spite of the validity of the criticisms given above, the actual impact of these erroneous elements on the precision/recall figures is small. This means that the parser is, in many respects, robust in the face of small inconsistencies. One of the reasons being that the parse output by the parser is the result of a competition between all the possible parses of a given sentence. This factor, for nontrivial sentences, means that the consistent aspects of a sentence will tend to outweigh the inconsistent aspects sufficiently to win the competition. The precision/recall evaluation of this parse operates at the level of constituents, not sentences or complete parses, and so mistakes at the constituent level will not necessarily lead to an adverse affect on other correct constituents within the sentence.

It should also be noted that the PCFG uses a classical form of representation which, in view of the discussion in Section 2.3.1, is worth emphasising. Essentially, the use of non-terminal symbols in the CFG, and their composition in rewrite rules, means that systematicity is an inherent element of their generalisation ability. Information learned about words or part-of-speech tags in one constituent (labelled by one non-terminal symbol) will automatically be generalised to further instances of that constituent (later occurrences of that same non-terminal).

To conclude, the PCFG offers a set of benchmark results achieved by a statistical language learning technique in the application of learning to parse natural language. The next section considers a different approach to language learning, that of the connectionist community.

3.4 Connectionist Language Learning: Recurrent Networks

In contrast to statistical language learning, connectionist language learning begins from a different premise: instead of modelling the grammatical representation of language, the network is left to work out its own internal representation based on the observed association of input to output. This approach is justified because connectionist networks are known to be capable of learning a wide range of possible mappings from input to output, given adequate training data. The use of a connectionist network therefore aims to demonstrate that a general learning algorithm can achieve a reasonable performance at the particular task of learning natural language. This section describes some applications in this spirit of a connectionist network to language acquisition.

The primary requirement for a connectionist network when dealing with language is an ability to learn about patterns across time. This ability is required for language learning because the output features for an individual word within a sentence may be determined by the preceding words in that sentence. Therefore the Simple Recurrent Network (SRN) [18, 19] is popular in this area because, as opposed to the feed-forward network, the SRN has context units which hold information from earlier parts of a sentence for use in processing the current word.

This section reviews the following areas of work. The first, by St. John and McClelland [47], introduced the Sentence Gestalt model, which learns about semantic roles and makes automatic inferences about sentences selected from a particular target domain. By contrast, the basic SRN was used in a series of experiments which focused more on the broader ability to handle grammatical information [18, 19, 20]. Elman's work was restricted to toy grammars, but demonstrated the effectiveness of the SRN for language learning. Lawrence et al. [61, 60, 62] amplified Elman's work with experiments requiring the SRN to learn more complex grammars. In addition, Reilly [81] presents some experiments with samples of real natural language, in place of the more limited domains addressed above. Although these approaches have relied on simplistic output representations, more complex tasks may be tackled. An important area of connectionist language learning is occupied by the holistic parsers, e.g. as summarised in [43], which use recurrent networks to output structured parse trees. Finally, some alternative approaches to connectionist language learning have been devised, employing various forms of structured network design.

3.4.1 Sentence roles, inference and selection

In the previous section on statistical language learning, a representation for natural language was proposed which assigned a syntactic structure to a string of words. This structure, a parse tree, was intended to encapsulate some of what the native speaker knows about his/her language. This representation would assign to each word its role within the sentence. For example, whether the word is a verb, noun or adjective. However, there is a further level of sentence comprehension at which the native speaker intuitively understands the meaning behind a sentence. The following connectionist model of some aspects of language learning is intended to capture this semantic level of sentence comprehension. In many ways it addresses the next stage from the simple parsing process considered above, because it converts the syntactic structure into a semantic structure. St. John and McClelland [47] therefore aim to 'develop a model that can learn to convert a simple sentence into a conceptual representation of the event that the sentence describes.' In particular, they are concerned with 'the conversion of a sequence of sentence constituents, such as noun phrases, into a representation of the event.'

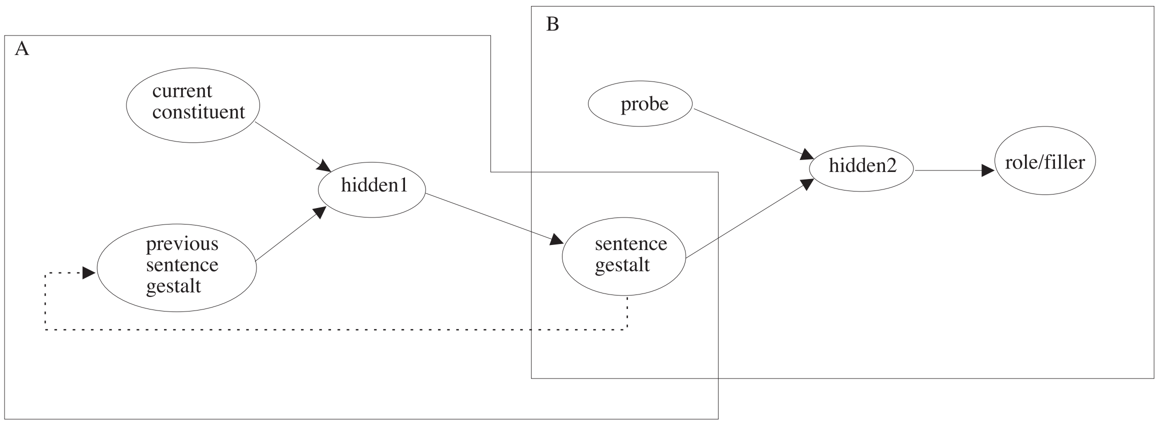


Figure 3.3: The Sentence Gestalt network: area A processes sentences into the sentence gestalt, and area B the sentence gestalt into the output representation.

When people read and understand a sentence, they fill out the semantic information contained in that sentence based on the words in the sentence. For example, with the sentence 'Bobby pounded the boards together with nails', people automatically infer 'with a hammer'. This process of automatic inference produces a number of beneficial effects, such as: the ability to disambiguate ambiguous words, to instantiate vague words, to assign thematic roles, and to elaborate implied roles. The model developed by St. John and McClelland is intended to learn these abilities, as well as show how an interpretation of a sentence is adjusted as each constituent is processed.

The Sentence Gestalt (SG) model developed by St. John and McClelland [47] is a two-stage network, illustrated in Figure 3.3. The first stage is a standard recurrent network, which learns the sentence gestalt information from a temporal sequence of constituents. Each constituent is either a simple noun phrase, a prepositional phrase, or a verb. It should be noted that only single clause sentences, without embeddings, are used. The second stage acts as a probe for information contained in the sentence gestalt. Each probe is a role/filler pair, and the sentence gestalt is probed by presenting either a role or a filler, from which the network is to supply the complete pair. For example, for the sentence 'The pitcher threw the ball', the possible role/filler pairs are: agent/pitcher, action/threw, patient/ball. Note that the role/filler pairs may not correspond to particular words in the sentence, but instead refer to information about the sentence as a semantic whole. For example, given an appropriate training set, the sentence 'Mary ate the spaghetti' would most likely have the filler "fork" in the instrument role. As is standard with outputs from connectionist networks, the actual output is a real number, and represents the likelihood of particular roles or fillers appearing in the location asked for. Thus, some degree of conceptual organisation can be noted by analysing the outputs of the network over a number of situations.

In their experiments, St. John and McClelland [47] train the SG model (with 85 input units for the sentence constituents and 100 units for the sentence gestalt and hidden layers) on small sets of events, around 100 sentences. Their results show that the SG model is successful in correctly assigning constituents to thematic roles based on syntactic and semantic constraints. Further, the SG model is able to use context to disambiguate meanings and to instantiate vague terms in ways appropriate to their context. These kinds of results, where the network aquires statistics about associations within the training set and combines the effects of multiple possibilities, are standard for connectionist networks. This ability is also behind the results achieved with the SRN described below, and provides a compelling justification for the use of such models.

3.4.2 Toy grammars

Elman [19] is interested in forming a computational model of language, and highlights the following questions:

- What is the nature of the linguistic representations?
- How can complex structural relationships such as constituency be represented?
- How can the apparently open-ended nature of language be accommodated by a fixed-resource system?

So as to illuminate possible solutions to these questions Elman [18, 19, 20] describes a series of experiments which investigate the ability of a connectionist model, the Simple Recurrent Network (SRN), in learning to predict words in sentences. The SRN has been described in Section 2.2.2, and is suited to experiments in language acquisition because of its ability to learn about patterns across time. Elman's experiments test the SRN's ability by training it on sentences from various toy grammars. The trained network is then analysed, to see what has been learned from the training data. This information provides some answers to the above questions.

There are two basic experiments which Elman performs. The first is a classification task, requiring the network to learn to predict words based upon the grammatical and semantic requirements of a sentence. This requires the network to learn how words cluster into semantic categories. The second requires the network to make more complex predictions based upon the grammatical agreement of verbs with nouns. In these experiments Elman established the basic protocol for connectionist language learning with SRNs, to learn to predict the next word in a sentence. This contrasts strongly with the complex output representations of parse trees formed by PCFGs, and also with the ability to extract role/filler pairs in the Sentence Gestalt model. However, this protocol does have the advantage of providing an unsupervised task to the SRN, without the necessity for a separate target output. It also reflects the general difficulty which connectionist models have in representing structured data.

In both types of experiment performed by Elman the task is to learn about sentences drawn from a toy grammar, which is designed to highlight particular features of language. The experiments train the network to predict the next word in the sentence. Therefore, if the sentence is 'dog sees cat', then the network must predict "sees" after being shown "dog" and predict "cat" after being shown 'dog sees'. The network's behaviour is not expected to be exact, as, for example, 'dog sees mouse' may also be a valid sentence. Instead, the network's output will reflect the relative probability of each word appearing next. Thus, 'dog eats bone' is more likely as a sentence than 'dog eats grass', and so, in predicting the third word after 'dog eats', "grass" should have a low probability, and "bone" a relatively high one.

The words of the sentence are represented in an identical fashion on the network's input and output units, with a localist representation using basis vectors, i.e. every word is assigned a vector in which a single bit is turned on. Thus, no information is encoded into the representation about any item's syntactic category. Any information required by the network in predicting a given word's occurrence must therefore be learnt from that word's context in the sentences of the training data. The training data is created by a sentence generating program, with the words in every sentence concatenated into a single stream. This stream of words is then presented to the network in consecutive training passes. One comment should be made here about Elman's training algorithm: instead of using Backpropagation Through Time [85], the context units are simply treated as additional input units within a feed-forward network, and the standard backpropagation algorithm for static patterns is used. This has a constraining effect on the lengths of dependencies which can be acquired, and later researchers, e.g. Lawrence et al. [62], prefer the more powerful Backpropagation Through Time.

In the first set of experiments, *category prediction*, the aim is to test the SRN's ability to infer categories of words from the training data [18]. For example, if the verb in a sentence is "eats", then any edible noun will be a suitable next word. However, because the information about which nouns are edible is not present in the training data, it must be inferred from the use of the words in sentences seen by the network. In order to test for this ability, the network is presented with a collection of two or three word sentences containing words from a variety of semantic categories. The sentences are of the form 'noun verb' or 'noun verb noun'. The nouns and verbs are chosen so that verbs may require a direct object, never have a direct object (i.e. be intransitive), or have an optional direct object. Further, the actual nouns and verb appearing in each sentence are constrained semantically. For example, the nouns were divided into "animates" and "inanimates"; animate nouns into the classes "human" or "nonhuman"; nonhuman nouns into "large animals" or "small animals", etc. Similarly, the verbs can be separated into semantic classes, requiring particular classes of noun as a subject.

An SRN with 150 hidden units was trained on a corpus of approximately 10,000 of these sentences, drawn from a vocabulary of 29 words. The network was then analysed for evidence of semantic categorisation. For each sentence, the network's mean hidden unit vector was used to determine which nouns, for example, were treated similarly by the network. Words are treated similarly if they have a similar likelihood of being predicted in a particular sentence. If the sentence began "boy eats" then the next word should be predicted to be an edible noun, and in this example the network assigns both "sandwich" and "cookie" an equally high likelihood. Elman found that the hidden units represented the distributional properties of the lexical items in the corpus, and therefore the SRN had formed an equivalent internal representation purely from the sentences present in the training set.

In Elman's second set of experiments, involving grammatical agreements, the sentence grammar was extended to include relative clauses, e.g. 'boys see dogs who see girls who hear'. It should be noted that these sentences are not claimed to capture all aspects of English grammar. They are deliberately simplistic to capture certain features of the language, in this case the problem of noun-verb agreement. For example, in the sentences 'boys who girls see hear' and 'boy who girls see hears', the required agreement in number of the final verb to the first noun must be preserved across the relative clause.

The corpus for this task is formed by extending the sentence generator program for the category prediction task with relative clauses and including a further constraint of noun-verb agreement. Sentences are therefore of the form 'noun verb' or 'noun verb noun', with the added complication that each noun may have an associated relative clause of the form 'who verb noun' or 'who noun verb'. Sentences must also obey certain grammatical rules. These are that the subject noun and verb in a sentence or clause must agree. Again, verbs fall into one of three classes, where a direct object is required, optional or precluded. Relative clauses extend the range of such interactions. Recursion of relative clauses, e.g. 'Boys who girls who dogs chase see hear', can extend the distance of agreements over considerable distances. Finally, the '.' marker for the end of a grammatical sentence can appear whenever appropriate.

For this experiment, Elman [19] used an extended form of the SRN, as shown in Figure 3.4. Two extra layers have been added, labelled 'Comp' for Compression. The Input and Output layers each have 26 units, and the Hidden and Context units each have 70 units. The Compression layers each have 10 units, and are designed to ensure that the mapping from Input-to-Hidden or Hidden-to-Output is distributed over several units.

Experiments were run in a similar fashion to those in the lexical categorisation experiment. A continuous string of 10,000 sentences was constructed, and the network trained on successive

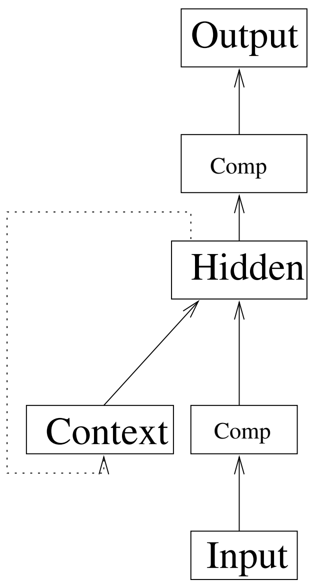


Figure 3.4: Diagram of Extended Simple Recurrent Network

passes. However, due to the additional complexity caused by the recursion, it was found that the network could not learn the full task all at once. Therefore, the network was first trained on sentences without any recursion, and then progressively trained on sentences composed of increasing layers of recursion. This approach, starting from the simpler data and working towards the more complex, enables the network to learn the basic interactions, such as agreement and verb argument, before extending these to apply across relative clauses. (See Elman [21] for more details on this approach, although Rohde and Plaut [83, 84] present evidence indicating that this approach is neither robust nor essential with more complex languages.)

Results from these experiments were analysed as before, by computing the mean hidden unit vector for the words in various sentences. Two points of interest came out of this analysis. The first is that, as planned, the SRN was capable of learning to predict the relative likelihood of the different words, reflecting the requirements of agreement and verb argument. The second is that the precise representation of each word differed depending on the context of that word. For example, in the sentences 'boys see boys' and 'boys who see boys see boys', the word "boys" is mapped to a similar hidden unit vector in each case. However, when the word appears as the initial subject of a sentence, it has a slightly different vector to the case when it is the object of a relative clause. Thus, each word has a similar vector, because it is the same word, but the vector changes slightly to reflect the grammatical context of that word.

This ability of SRNs to alter the basic representation of a word by small amounts to reflect its grammatical context is important. Elman [20] compares this ability to handle interactions among constituents to the *accommodation* model of language, put forward by Langacker [58]. Accommodation refers to the subtle changes a word undergoes based upon its context. For example, the word "runs" alters its meaning in the cases, 'the man runs', 'the cat runs' and 'the tap runs'. This ability, as noted before, is a general ability of connectionist networks to capture statistical information and combine information from different entities to produce an output.

In conclusion, the SRN has proven a viable model of language learning, capturing and using certain regularities in language, such as agreement and verb argument. However, the distributed internal representation raises questions as to the capacity of the network in terms of vocabulary. Each word has a range of hidden unit vectors which are categorised as that word, and movement of the word vector within this range is used to encode for the grammatical context of that word with further

Category	Examples
Nouns (N)	`John, book`
Verbs (V)	`hit, be`
Adjectives (A)	`eager, old`
Prepositions (P)	`without, with`
Complementiser (C)	`that, for`
Determiner (D)	`the`
Adverb (Adv)	`quickly`
Marker (Mrkr)	possessive `'s`

Table 3.3: Parts of speech

movements potentially encoding semantic changes. The number of available hidden unit vectors, each with its own range for grammatical context, will be constrained by the size of the network. Training for a large vocabulary with a large range of possible grammatical positions would require an equally large training set. Although Elman tests his SRNs on unseen sentences from the corpus, no attempt is made to limit the training corpus so that generalisation of the network across sentence structure may be observed, such as required by the property of systematicity demonstrated in the experiments of Hadley and Hayward [32, 33] (Section 2.3.4). The next two groups of experiments attempt to apply some of the strong points of Elman's work with SRNs to more complex examples of natural language.

3.4.3 Natural language grammatical inference

Lawrence et al. [60] investigate the ability of various machine learning techniques on the inductive inference of a complex grammar, concentrating solely on recurrent connectionist networks in Lawrence et al. [62]. The importance of this work, as compared to that of Elman from the previous section, is that the task is more complex involving samples of real natural language. Lawrence et al. [60, 62] require their networks to classify natural language sentences as grammatical or ungrammatical. The intention being that the final system should have acquired the discriminatory power of the Principles and Parameters (Government-and-Binding Theory) linguistic framework of Chomsky [13].

The training data contained 552 English positive and negative examples taken from an introductory GB-linguistics textbook. Most of this data is organised into minimal pairs, such as: `I am eager for John to win / *I am eager John to win`, with the '*', by convention, indicating an ungrammatical sentence. The networks are then trained to indicate whether a given sentence is grammatical or not. Due to the small size of the dataset, the words were first converted into the major syntactic categories assumed under GB-theory. Table 3.3 summarises the major parts of speech used.

Four types of network were investigated in [62]: locally recurrent networks, which are multilayer perceptrons with local feedback on each hidden node; a network with feedback from each output node to all hidden nodes; an Elman-style Simple Recurrent Network (SRN); and a fully recurrent network.

It was found that Elman's SRN achieved the best performance. (Note however that Elman only uses a limited version of Backpropagation, whereas Lawrence et al. use true Backpropagation Through Time.) Using an SRN with 20 hidden units, and an input window of the last two words in the sentence, the SRN could achieve 100% accuracy in training (99.6% average over 5 trials). On the test data this resulted in 74.2% accuracy, i.e. the SRN was able to discriminate between grammatical and ungrammatical test sentences fairly successfully. This performance is more impressive when it is considered that the training set is relatively small, and the data, selected by a linguist, are examples of minimal pairs, which need not have any generalisable properties.

This comparison of recurrent networks is interesting as it shows, experimentally, that the SRN is the more powerful of the various recurrent network architectures in this domain. Partly this is due to the connectivity of the SRN, e.g. as compared with the first network which only uses locally recurrent links on the hidden layer. However, this is not just a connectivity issue, as shown by comparison with the fully connected network, which did not learn as well although its architecture is an SRN with extra recurrent links from the output units. The difference, as Lawrence et al. show, lies in the shape of the error surface. The experimental evidence suggests that the shape of the SRN's error surface is more appropriate for the training algorithm used, i.e. Backpropagation Through Time.

3.4.4 Real natural language

The previous two sections have described the application of Simple Recurrent Networks (SRNs) to learning subsets of natural language, either sentences from toy grammars, or minimal pairs of grammatical/ungrammatical sentences. These experiments have shown that the SRN is capable of learning complex grammatical information. However, to be of real interest as a model for practical language learning or psycholinguistic modelling, these experiments must be extended to samples of real natural language, such as the datasets used for statistical language learning.

One example of such work is that of Reilly [81], in which an SRN was trained with a sample of 4000 sentences taken from the Wall Street Journal corpus. As Reilly explains, applying an SRN to such a corpus has two major problems. The first is the training algorithm, which is slow and inefficient. In order to address this, Reilly adapts Elman's starting-small training procedure [21], which trains the network on progressively longer dependencies. Reilly trains the network with the full corpus as training data on each epoch, but resets the context unit activations at different stages to control the length of dependency being trained. Initially this occurs after every word, but after successive numbers of training cycles this is increased, until finally the reset occurs only at the end of each sentence.

The second difficulty faced by the SRN is that the standard input-output format used is rather crude. Reilly instead uses a representation adapted from one by Zavrel and Veenstra [107], which reflects the semantic context of each word in the corpus. Thus, instead of using a 1-in-n vector, each word is representated as a distributed lexical context. Essentially, a context is determined for each word in the corpus by considering the nearest two words before and after it in the sentence. In order to make this manageable, only the most common 250 words in the corpus are used to determine context. Each word therefore has a four component context, one component for each of the neighbouring words, where each component may take one of 250 values (words). Using principal component analysis this has been reduced to a set of 25 lexical representation vectors which determine the context of each word.

Using these techniques, Reilly has shown that a trained SRN has some success on word-tag prediction (20% of the exact tags are predicted, 70% of the time the right word tag is one of the top 5 predicted, an untrained network achieves near 0% in each case). However, Reilly is less interested in the performance of the SRN than in its appropriateness as a model for psycholinguistic phenomena. One such phenomenon is the representation of ambiguous words by their context. This is tested by observing the network's response to an ambiguous word, such as "suit" in the sentence 'The suit was worn by the chairman'. By looking at the predicted vector by the network for the verb, it was clear that the network preferred a legal context for the word "suit" as in the sentence

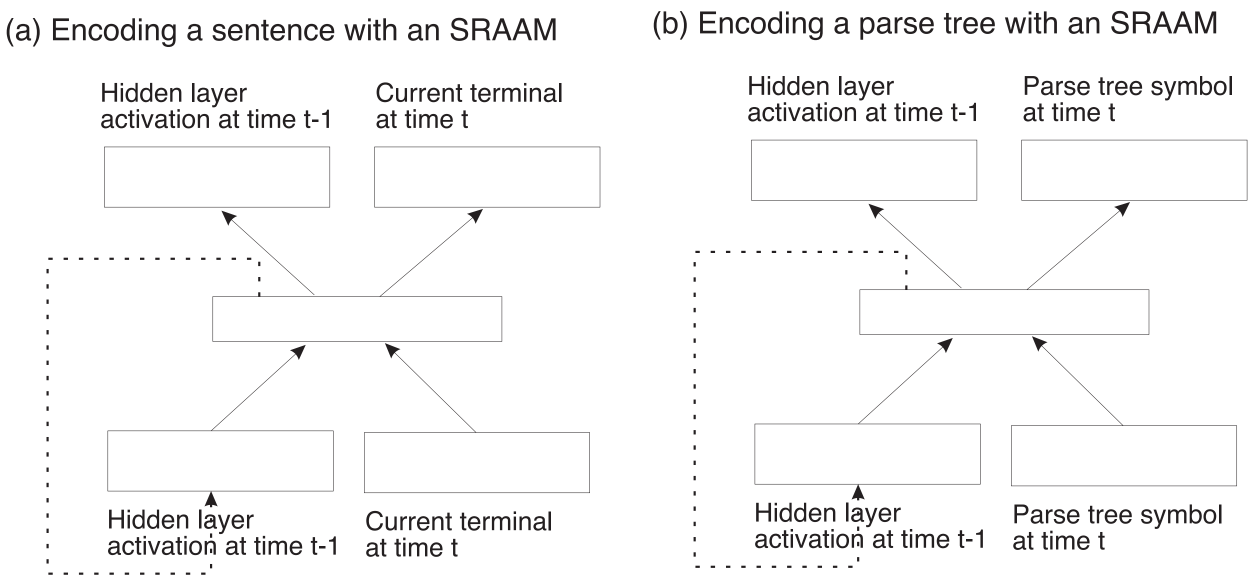

(a) Encoding a sentence with an SRAAM
Hidden layer activation at time t-1
Current terminal at time t
Hidden layer activation at time t-1
Current terminal at time t
(b) Encoding a parse tree with an SRAAM
Hidden layer activation at time t-1
Parse tree symbol at time t
Hidden layer activation at time t-1
Parse tree symbol at time t

Figure 3.5: Diagram of a Holistic Parser

'The suit was filed by the company'. This is because the training corpus had a predominance of sentences from a legal context.

Such analysis has some similarities with that of St. John and McClelland's Sentence Gestalt (SG) model, which used a probe to determine the SG-network's current contextual preference in a given sentence. This kind of analysis again demonstrates that the SRN is capable of acquiring statistical contextual information for different words, although the actual performance of the SRN with this contextual information is not particularly impressive.

3.4.5 Holistic parsers

In all cases, the use of SRNs for language learning in the previous subsections have employed unstructured output representations. The holistic parser augments the SRN with further mechanisms for generating structured output, such as parse trees. The basic concept behind an holistic parser is to present the input sequence sequentially to a recurrent network, and so encode it into a distributed representation. This representation may then be transformed, and a decoding process employed to extract a parse tree from the distributed representation. The parser gets its holistic nature from the internal distributed representation, which contains within it information about the entire sentence.

A variety of holistic parsers has been developed, and a summary and empirical comparison is provided in Ho and Chan [43]. The variety arises from the form of network used to perform the encoding or decoding, and whether an explicit mapping is necessary for converting the output from the encoder into the input for the decoder. For example, two common forms of encoder/decoder are the SRN or the RAAM (the latter as described in Section 2.3.2 of this thesis). In Sharkey and Sharkey [90], the SRN was used as the encoder, and trained on the sequence prediction task, just like the experiments in the previous subsections. A RAAM was used as the decoder, and trained to map a distributed representation into a parse tree. (Note that an extra encoder must be trained for the RAAM as well, to provide the distributed representation of the parse tree.) The third element of the parser is a standard feed-forward network, which takes the hidden layer representation of the SRN after reading the whole sentence and transforms it into the distributed representation for decoding by the RAAM. Reilly [80] developed a similar model, but instead of training the encoding SRN to predict the next word in the sentence, the SRN was trained to output the distributed representation

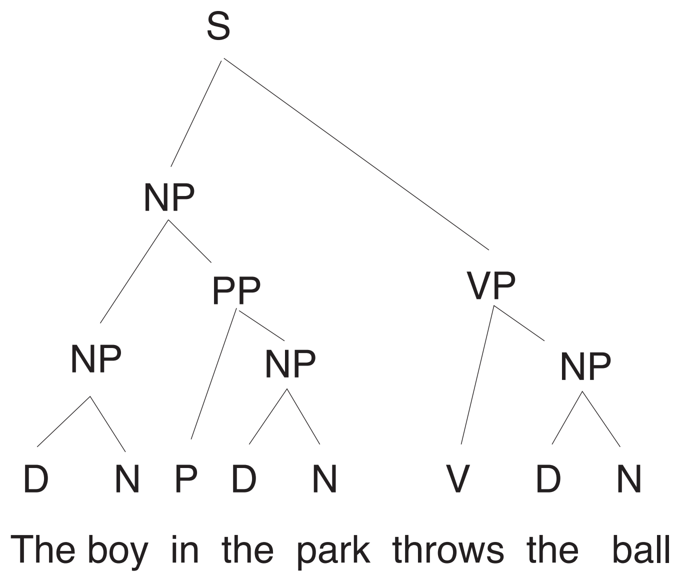


Figure 3.6: Diagram of Sample Parse Tree used in Holistic Parsing

for the RAAM.

The SRAAM (Sequential RAAM) [6] may also be used be used for encoding the input sentence. This network is essentially the same as an SRN, except with auto-association. As with the standard SRN, the input to the network is a sequence of words, along with the activation on the hidden units from the previous time period, as conveyed in the context units. The output of the SRAAM reproduces this input, outputting the current input word as well as the activation on the hidden units from the previous time period. After reading the whole sentence, the hidden-layer activation is used as the encoded form of the sentence. Two of these SRAAMs are used by Ho and Chan [43] in their Confluent Preorder Parser (CPP), illustrated in Figure 3.5, one for encoding and one for decoding. The two SRAAMs are trained together, so that their hidden layer representations for the sentence encoding and the parse tree are identical. This is achieved by training the two SRAAMs side-by-side, passing the error from each SRAAM down both the sentence SRAAM and the parse tree SRAAM. Thus, the representation produced by the sentence SRAAM can be used directly to generate the parse tree from the parse tree SRAAM.

What forms of parse tree may be represented within an SRAAM? In Ho and Chan [43] a simple context-free grammar is used to generate sentences such as that illustrated in Figure 3.6. Because this parse tree is to be represented within a connectionist distributed representation, its structure must be made explicit in its surface form. The sample sentence is therefore represented by its preorder form, in which the tree is read out, beginning from the root node, and proceeding down the left-branch to each terminal. The sample sentence is therefore represented as: (s np D N vp V np np D N pp P np D N). This sequential representation of the parse tree may then be encoded using the SRAAM, as described above. However, this approach is only applicable to fixed-valence trees, i.e. those in which the number of child links from each node is fixed. In this case, the tree is strictly binary. (Note that a null symbol is introduced in the rare cases where a termial node is not present. Further discussion of this representation is contained in Ho and Chan [42].)

Ho and Chan [43] contains an empirical comparison of a variety of holistic parsers on a simple context-free grammar. Their ability to generalise to novel sentences is tested, although the type of novelty is not controlled, e.g. in terms of systematicity, such as with the Hebbian connectionist network [32, 33] described in Section 2.3.4 of this thesis. In addition, their robustness across noise in the input is tested. Ho and Chan [43] also discuss some limitations in holistic parsers. The more

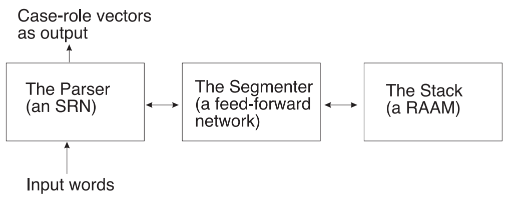

Case-role vectors as output
The Parser (an SRN)
The Segmenter (a feed-forward network)
The Stack (a RAAM)
Input words

Figure 3.7: The subsymbolic parser for embedded clauses

important of these, in terms of scalability of the model to real world data, include:

- the simplicity of the input representation prevents the use of semantics to disambiguate possible alternative parses for equivalent sets of input word-tags.
- difficulties in training and convergence inhibit the use of holistic parsers on larger corpora (the examples use only 112 sentences).
- the distributed representations scale poorly in practice, permitting only a limited number of possible sentence and parse tree structures.

In addition to these, the representation for the parse trees is a further limiting factor. The requirement that the parse tree be represented with fixed-valency would rule out many corpora typical for parsing with statistical parsers (or alternatively require many changes to be made to the target representation). Many of these restrictions stem from the basic principle behind the holistic parser, that a single distributed representation for the entire sentence should be used to encode all the information required for parsing.

3.4.6 Other approaches

There exists a multitude of other approaches to connectionist natural language processing, and a full review would go beyond the aims of this chapter. The SRN and holistic parsers covered above have been designed as rather generic architectures for coping with general problems in parsing, such as the use of patterns across time and the output of structured information. Other approaches involve including some linguistic knowledge explicitly into the connectionist architecture.

For example, the subsymbolic parser for embedded clauses (SPEC) [68] uses a network divided into three sections, as depicted in Figure 3.7. SPEC uses an SRN as a parser, a RAAM network as a stack, and a feed-forward network as the segmenter. The model handles case-role assignments only, treating sentences as (agent, action, patient) triples. Thus, the sentence ‘The girl who liked the dog saw the boy’ parses into two triples: (girl liked dog) and (girl saw boy). SPEC’s input is the sequence of words in the sentence, and its output is the triple pertaining to the current input word. So, for the example sentence, SPEC must output the second triple for the first two words, the first triple for the second four words (i.e. through the relative clause) then revert to the second triple for the remaining three words. The stack is used to contain the parser’s representation for the initial triple whilst it is processing the relative clause; once the clause is completed, the state of the parser when processing the initial triple is restored. This process must occur on the boundaries between relative clauses, and the segmenter is used to recognise the boundaries and control the movement of information to and from the stack.

SPEC uses a network composed of three basic elements, as illustrated in Figure 3.7. The parser is the familiar SRN, and reads the word representation as its input and generates the case-role representation as its output. The generalisations across structure required of SPEC are facilitated through three architectural features [68]. First, the output consists of specific case-role vectors, instead of the more comprehensive representation found, e.g., in holistic parsers. Second, the segmenter network breaks the input sequence in to smaller chunks. And third, the stack network memorises constituents over intervening embedded clauses. The stack is used to store the parser's representation for uncompleted triples whilst processing relative clauses; once the clause is completed, the triple is then retrieved and the state of the parser restored. This process must occur on the boundaries between relative cluases, and the segmenter's role is to recognise the boundaries and control the movement of information to and from the stack.

SPEC produces good generalisation behaviour on a limited range of sentences. One difficulty is that the quality of the encoding in the stack degrades as more items are added, and this leads to the system breaking down with multiple-centre embeddings; this is a common problem with RAAM-type networks, and is also found in holistic parsers [44]. This causes linguistically interesting phenomena with centre-embeddings [12], but here the surface phenomenon is given a different explanation, limited resolution in the internal representation. However, SPEC is restricted to producing just these case-role triples. Hence, scaling up such a model to produce the parse trees found in natural language corpora would require more than simply applying the network to further datasets [69].

Further examples of more structured connectionist parsers include the connectionist deterministic parser [53], which combines a symbolic parser rather like PARSIFAL [63], with a feed-forward connectionist network, and the neural network pushdown automaton [96]. However, these approaches, due to the extra structure, are not purely connectionist parsers, as they partially reimplement some properties of symbolic systems. Hence, this concludes the review of connectionist attempts at natural language learning.

3.5 TSVB Parser Representations: NNEP

The previous sections have described various learning algorithms used in language acquisition. This section describes some work which demonstrates that TSVB networks are an adequate computational framework for the task of parsing natural language; the question of learning with such networks is dealt with in the next chapter.

The Neural-network Node Equating Parser (NNEP) [36] is a connectionist implementation of a syntactic parser. As a computational framework it uses the TSVB networks introduced by Shastri and Ajjanagadde [92]. The use of this framework is motivated by the inability of traditional connectionist architectures to capture generalisations due to compositional structures, i.e. systematicity [24]. TSVB provides a means for achieving such generalisations across constituents, as discussed in Section 2.3, thereby capturing systematicity within the connectionist architecture. Further, Shastri and Ajjanagadde [92] provided an implementation of TSVB in the model of reflexive reasoning known as SHRUTI. This model is a connectionist system which supports the massively parallel use of knowledge, evidential reasoning and symbolic computation. These features are all useful for constructing a model of syntactic parsing.

Henderson [36] begins with a model for the grammar and output from the parser, known as a Structure Unification Grammar (SUG) [35]. It is not appropriate to go into the details of SUG here, but essentially the grammar supports the accumulation of partial information about the phrase structure of a sentence until a complete description of the sentence's phrase structure tree is constructed. The flexibility of SUG derivations arises from the simple mechanisms for combining partial descriptions of phrase structure trees. The idea is as follows. For each word in the sentence, a partial phrase structure is computed. This structure can specify various expectations for the non-terminal nodes within it, e.g. the requirement of having (or not having) some other particular node. The SUG derivation then takes these partial structures, and combines them, by equating nodes across structures. Thus, a verb structure would contain an empty non-terminal expressing the need for a noun; this non-terminal would be equated with the head node of the constituent describing the relevant noun. The only restriction placed on the derivation is local consistency, and completeness of the structure is a restriction only relevant to the final result.

The properties of SUG clearly makes it attractive for a connectionist parser. That partial phrase structures can be computed for separate words and then combined makes for a flexibility in processing which fits well with the parallel and distributed model of reasoning offered by the TSVB model. The connectionist parser, NNEP itself, is built up from three kinds of unit: input units, predicate units and grammar units. Essentially, the input units carry activation about the words in the sentence into the network. The predicate units hold the current parser state, and so act rather like conventional hidden units. In addition, the predicate units and their links represent various pattern-action rules, which are used by an arbitrator to determine the parser's next action given the current parser state and the next input word. Finally, the grammar units fire to indicate the parser's action. These units act rather like output units, as they provide the information on the parse tree; they also feed activation back so as to alter the state of the predicate units.

NNEP relies on the implementation of SUG in the TSVB framework. This implementation uses a highly structured distribution of computational elements between the input, predicate and grammar sections of the network described above. The separate computations must handle the patterns and actions for the combination operators, arbitrators between the possible rules which may fire, and a parser memory with a module for handling the forgetting of unused phases. Although the architecture is so highly structured, it relies on a distributed representation across the predicate units. Also, the overall structure conforms closely to a standard three layered connectionist network, with feedback links from the predicate units to themselves, as in Elman's SRN [18], and similarly from the grammar units to the predicate units, as in [49].

In order to demonstrate NNEP's adequacy as a syntactic parser, Henderson [36] used test sentences drawn from various potentially troublesome grammatical categories. Details cannot be given here, but suffice to say that NNEP does handle a wide range of naturally occurring English sentences. Furthermore, the computational limitations of NNEP, which include determinism, limited instantiations of any relation and a bounded number of variables, help explain linguistic phenomena such as classes of sentences which are cognitively hard to handle, e.g. centre-embedding and some long distance dependencies [38].

To summarise, NNEP [36] provides a concrete example of the ability of TSVB connectionist networks to handle rules for parsing a wide range of linguistic examples and naturally occurring English sentences. The aim of Chapter 5 is to test whether trainable TSVB networks, with unstructured sets of hidden units, can learn equivalent parsing capabilities.

3.6 Conclusions

This chapter has presented a range of statistical and connectionist techniques for language learning and parsing. The strengths of each type of algorithm can be summarised in the following manner:

- PCFGs use a structured output representation for parse trees, which, being a compositional grammar, inherently generalises in accordance with systematicity. The PCFG also gathers statistical information from a large input corpus.
- SRNs use a large training corpus to gather statistical information about the input data.
- TSVB enables a connectionist network to represent an expressive grammar in a system of symbolic rules.

In spite of their wide-spread use in connectionist language learning, the SRN does not provide comparable performance to PCFGs, basically because its output representation is not structured, as required to represent a parse tree. The reason for this limitation is that, in each time period, each output unit can only specify one piece of information, i.e. the amount of information output by the network is linear with respect to the number of input words. In order to represent a parse tree, the relationship of each word to each of the preceding words must be shown, which requires a quadratic amount of information with respect to the number of input words. The holistic parsers attempt to correct this limitation by including additional structure in the output mechanisms, which must 'unfold' the internal representation of the sentence into a parse tree. This has some severe limitations however, primarily due to the limited capacity of the internal distributed representation, and the need for recursive processing to extract all the levels within the parse tree. Hence, they have not demonstrated their capability with naturally occurring text, as typically used with statistical parsers.

The next chapter defines a trainable class of TSVB networks, and Chapter 5 considers whether the SSN, the most effective of these TSVB networks, can bridge the gap between connectionist and statistical language learning.

Chapter 4

Trainable TSVB Connectionist Networks

The previous two chapters have covered a wide range of connectionist networks and systems for parsing. In this chapter a new form of connectionist network is developed, which has specific features making it suitable for learning to parse. The motivation for developing this new model has been argued on theoretical and practical grounds. The theoretical grounds were covered in Chapter 2, where it was argued that earlier extensions to the connectionist framework for handling structured information either did not address the output of appropriate information for parsing, or else did not provide sufficient flexibility in their learning mechanisms for success beyond a specific toy grammar. On practical grounds, Chapter 3 has described earlier work in parsing. Consideration of classical and statistical parsers alongside connectionist approaches to natural language has shown that the latter suffer through not offering similar forms of output representation. It is argued that the main reason for this is the lack of a suitable connectionist model for learning in the types of domain of which parsing is an example. This chapter develops such a network by demonstrating how to construct a range of trainable connectionist networks through combining Temporal Synchrony Variable Binding (TSVB) with Simple Recurrent Networks (SRNs).

To summarise the description from Section 2.3.5, TSVB [92] is an extension to the standard connectionist architecture intended to solve the "binding problem". The binding problem arises when it is necessary to represent multiple entities and multiple properties: a mechanism is required to represent which properties are bound to which entities. TSVB uses the synchrony of activation pulses to represent entities; if two units are pulsing synchronously then they are representing information bound to the same entity. Shastri and Ajjanagadde (S&A) [92] implemented this idea by dividing each time period into separate phases, with each phase used to represent a separate entity. Using the phases to represent variables, and defining units to represent logical propositions across these variables, S&A constructed a model of reflexive reasoning, SHRUTI. Unfortunately, the implementation of TSVB in SHRUTI uses binary-threshold units with non-differentiable activation functions. This means that standard connectionist training algorithms, such as backpropagation, cannot be applied to such networks.

The approach described in this chapter begins with the same TSVB framework, but constructs a different implementation of TSVB connectionist units along with a suitable training algorithm for networks composed of these new units. This is achieved by defining TSVB units based on the standard sigmoid activation function. These units can then be used in a network with the architecture of an SRN. A novel extension of Backpropagation Through Time [85] is then developed so that the TSVB networks can be trained in a similar manner to standard connectionist networks. One side effect of using TSVB is the creation of two kinds of unit, pulsing and non-pulsing, which produce a wider range of possible network architectures than is usual with standard connectionist networks.

This range of architectures is explored in Section 4.1.3.

The implementation of TSVB described here is not the only one possible, and some variants on the basic TSVB network are described in Section 4.1.4. In particular, one computational inefficiency in the proposed implementation is addressed, which is that all phases are retained whilst computing the current sequence. This problem can be alleviated by only retaining phases which have been recently referred to by the output units. The addition of TSVB enables the network to represent a range of structured output representations, and this is illustrated in Section 4.1.5. Finally, some experiments with toy grammars are used to test the networks' ability to output appropriately structured patterns and also to ensure their ability to learn and generalise is robust. These results also offer a comparison of the different architectures, and one property of the most effective class of architecture is described. This chapter concludes by defining the Simple Synchrony Network (SSN) as a recurrent TSVB connectionist network without links from pulsing to non-pulsing units.

4.1 Defining TSVB Networks

Temporal Synchrony Variable Binding (TSVB) [92] is a connectionist technique for representing entities. The following three principles encapsulate the central ideas of the TSVB framework:

- Each time period is divided into discrete phases, each phase associated with a distinct entity.
- Pulsing units compute within each phase independently of other phases, and so compute information about distinct entities.
- Non-pulsing units compute across all phases, combining information about several entities.

This section shows how these principles can be used to define a range of trainable TSVB networks. Firstly, the two types of TSVB unit are defined, the pulsing unit and the non-pulsing unit, based on the standard net and sigmoid activation functions. Next, a training algorithm is developed through a novel extension of Backpropagation Through Time [85]. Because TSVB networks have two kinds of unit, pulsing and non-pulsing, there is a greater variety of possible network architectures. This range is investigated, and six different architectures for TSVB networks, arranged in two groups, are given. The representation by TSVB networks of structured output is illustrated, although specific examples are to be found in the experiments described in the rest of this thesis. Finally a number of possible variations on this implementation of TSVB networks are considered.

4.1.1 Adding TSVB to connectionist units

In this section it is shown how the principles of TSVB can be added to standard connectionist units in order to define the two kinds of unit in TSVB networks, pulsing and non-pulsing. To introduce the terminology and show the naturalness of this extension, the definitions for units in standard feed-forward and recurrent networks are first presented (fuller descriptions for these may be found in Sections 2.1 and 2.2.2 respectively).

In a standard feed-forward connectionist network, there are three kinds of unit: input units, output units and hidden units. Every unit, j, has an activation value, o_j, its output. For each input unit, j, this value is obtained from the input pattern:

$$o_j = \text{in}_j \qquad \text{if } j \text{ is an input unit}$$

Each non-input unit, j, receives input activation from a set of units, indexed by the set of integers, Inputs_j; one of these is the unit's bias input, and its activation is always 1. The activation value of unit j is computed from the activation received from these input units. The output activation, o_i, of each of these units is multiplied by the weight, w_{ji}, of the link and summed, forming the unit's net input:

$$\text{net}_j = \sum_{i \in \text{Inputs}_j} w_{ji} o_i$$

The unit's activation value is then computed by applying the sigmoid function to its net input:

$$o_j = \sigma(\text{net}_j) = \frac{1}{(1 + e^{-\text{net}_j})}$$

In order to extend these definitions to recurrent networks, time must be added. This can be achieved, without loss of generality, by adding context units to the network; each context unit's activation value is that which another unit had during the previous time step. The one-to-one function, C, maps each unit to its associated context unit, and, conversely, the function C^{-1} maps each context unit to its associated unit. For a unit j in a recurrent network, the definitions for its net and activation values at time t are as follows:

$$\text{net}_j = \sum_{i \in \text{Inputs}_j} w_{ji} o_i(t)$$

$$o_j(t) = \begin{cases} \text{in}_j(t) & \text{if } j \text{ is an input unit} \\ o_i(t-1) & \text{if } \exists i.j = C(i) \text{ and } t > 1 \\ 0 & \text{if } \exists i.j = C(i) \text{ and } t = 1 \\ \sigma(\text{net}_j(t)) & \text{otherwise} \end{cases}$$

These definitions can be extended to form an implementation of TSVB units in the following manner. First, the central idea underlying TSVB is to divide each time period into discrete phases, the number of which may vary over time. To retain a correspondence of phases with variables, the precise phase number should have no computational role in the network. With this restriction, the following two types of unit may be defined. The first is the *pulsing* unit, which computes in individual phases independent of other phases. Its output activation will be an $n(t)$-place vector, i.e. the activation, $\vec{o_j(t)}$, of a pulsing unit j at time t is formed from $n(t)$ values, $\{o_j^p(t) | 1 \leq p \leq n(t)\}$ where $o_j^p(t)$ is the activation of unit j in phase p at time t. The second type of unit is the *non-pulsing* unit, which computes across all phases equally in the current time period; its output activation, $o_j(t)$, at time t is constant across every phase in time t.

The net input and output activation for each type of unit within a TSVB network is defined based on the type of unit it is receiving activation from. Pulsing units in the network are indexed by a set of integers U_ρ, and the non-pulsing units by a set of integers U_τ. The output activation of a pulsing unit, $j \in U_\rho$, is defined in an analogous manner to a standard unit in a recurrent network. The difference is that up to $n(t)$ activations may be present within each time period, and input may be received either from other pulsing units or else from non-pulsing units. This latter fact is reflected in the function $R^p(i,t)$ within the computation of the unit's net input. This function returns the activation of unit i in phase p of time period t: if unit i is a non-pulsing unit ($i \in U_\tau$) its activation is constant across the time-period, and so $R^p(i,t)$ is $o_i(t)$; if unit i is a pulsing unit ($\in U_\rho$) its activation depends on the phase, and so $R^p(i,t)$ is $o_i^p(t)$. Hence, the activation of a pulsing unit i in phase p at time t is defined as follows:

$$\text{net}_j^p(t) \quad = \quad \sum_{i \in \text{Inputs}_j} w_{ji} R^p(i,t)$$

$$\text{where} \quad R^p(i,t) = \begin{cases} o_i(t) & \text{if } i \in U_\tau \\ o_i^p(t) & \text{if } i \in U_\rho \end{cases}$$

$$o_j^p(t) = \begin{cases} \text{in}_j^p(t) & \text{if } j \text{ is an input unit} \\ o_i^p(t-1) & \text{if } \exists i . j = C(i) \text{ and } t > 1 \\ 0 & \text{if } \exists i . j = C(i) \text{ and } t = 1 \\ \sigma(\text{net}_j^p(t)) & \text{otherwise} \end{cases}$$

Note that the net function for a phase p takes activation only from other pulsing units in phase p, or from non-pulsing units, whose activation is the same across all phases.

Similarly, the output activation of a non-pulsing unit, $j \in U_\tau$, at time t is defined as:

$$\text{net}_j(t) = \sum_{i \in \text{Inputs}_j} w_{ji} T(i,t)$$

$$\text{where} \quad T(i,t) = \begin{cases} o_i(t) & \text{if } i \in U_\tau \\ \sum_{p=1}^{n(t)} o_i^p(t) & \text{if } i \in U_\rho \end{cases}$$

$$o_j(t) = \begin{cases} \text{in}_j(t) & \text{if } j \text{ is an input unit} \\ o_i(t-1) & \text{if } \exists i . j = C(i) \text{ and } t > 1 \\ 0 & \text{if } \exists i . j = C(i) \text{ and } t = 1 \\ \sigma(\text{net}_j(t)) & \text{otherwise} \end{cases}$$

Note that the net function sums activation from pulsing units across all phases equally, as well as summing activation from other non-pulsing units. In this case, the function $T(i,t)$ takes on a similar role to $R^p(i,t)$ for the pulsing units, except here activation from the pulsing units is summed across their phases to compute their contribution for the entire time period.

These units can be used to construct TSVB connectionist networks. Before considering the derivation of a training algorithm for TSVB networks, there are some further points of this implementation which require discussion.

The original intention behind S&A's use of TSVB was for the phases to identify variable bindings. This intention is preserved in the above definitions for two reasons: the phase number plays no role in the computation of any phase's activation, and information can only be passed between phases through the non-pulsing units, which compute across all phases equally. The equivalence of phases and variable bindings in the current implementation can be shown in the following way. Variables are used in a representation to identify separate entities, and so distinguish information about different entities. The definition of pulsing units is such that information on each phase is kept independent from that on other phases. Thus, by inputting information to the network about separate entities in separate phases, the pulsing units will compute information on separate variable bindings. This property is maintained because the computation performed by the network does not vary if a different phase (variable name) had been used for the input entity. The reason for this is simply because, firstly, in no case is the actual phase number used in the computation. The pulsing units compute in each phase independently of its number and other phases. Secondly, because the weights applied by the links are not phase-dependent, the identical computation will be applied to every phase in the time period. The only computation which applies across all phases is performed by the non-pulsing units, and this combines activation from all phases equally. Finally, one implicit assumption which should be noted is that no resource constraints apply to the number of phases in any time period; a fixed network can therefore handle arbitrarily many entities.

However, the use of continuous activation values in TSVB networks also presents some difficulties. In particular, in certain network architectures the presence of continuous activations can restrict the ability of the network to generalise to increasing numbers of entities. This problem arises when a non-pulsing unit receives activation from a pulsing unit. As the outputs from the pulsing unit are real values, they will not be precisely 0 or 1, but, perhaps, 0.1 or 0.9. If the time period contains a number of phases in which the output of the pulsing unit is 0.1, these can accumulate to exceed the threshold value of the non-pulsing unit's activation, perhaps set to 0.5. In other words, a large set of near zeros can produce a similar response to a small set containing a one. One remedy to this problem is to have no links from pulsing to non-pulsing units within the TSVB architecture. This restriction is taken in Section 4.3 to define the Simple Synchrony Network, a subset of the class of TSVB networks. An alternative, the probabilistic-or function, is discussed at the end of Section 4.1.4.

This first section has shown how TSVB can be implemented in a manner consistent with the standard connectionist framework, using units with continuous, differentiable activation functions. The next section shows how a novel extension to Backpropagation Through Time can be derived, which enables these TSVB networks to be trained.

4.1.2 Training TSVB Networks

Backpropagation Through Time (BPTT) [85] is a standard algorithm for training recurrent networks. In this section, a novel extension of BPTT is developed for training recurrent TSVB networks.

When applying BPTT to a standard recurrent network one copy of the network is made for each time step in the input sequence. Extending BPTT to TSVB networks involves making a further copy of the network for every phase in the time period. The unfolding procedure is illustrated in Figure 4.1, where a simple TSVB network (a) is unfolded over two time periods and two phases (b). Note that both the pulsing and non-pulsing units are copied once per time period and that the pulsing units (shown stacked) are copied additionally once per phase. As with standard BPTT the unfolded network is a feed-forward network, and can be trained using backpropagation. However, every copy of each link must be updated so as to have the same weight, by summing all the individual changes to each copy of the link.

Weight-update equations for recurrent TSVB connectionist networks can be obtained by analogy with those for SRNs.

The TSVB network is trained on a *sequential supervised learning* task, so that at certain times some unit outputs of the network should match those of a teacher. Define $D(t)$ as the set of indices, k, of units for which a specified target value exists. This target value will be $d_k(t)$ for non-pulsing units and $d_k^p(t)$ for pulsing units; the latter requiring a target output to be specified for every phase, p, of the time period, t. $D(t)$ is assumed not to include any input or context units. The error for each unit is defined as follows:

for a pulsing unit,

$$e_k^p(t) = \begin{cases} d_k^p(t) - o_k^p(t) & \text{if } k \in D(t) \\ 0 & \text{otherwise} \end{cases}$$

for a non-pulsing unit,

$$e_k(t) = \begin{cases} d_k(t) - o_k(t) & \text{if } k \in D(t) \\ 0 & \text{otherwise} \end{cases}$$

The target for gradient-descent learning is to minimise the sum-squared error of every unit in the network over some sequence of time periods, $[t_1, t_2]$.

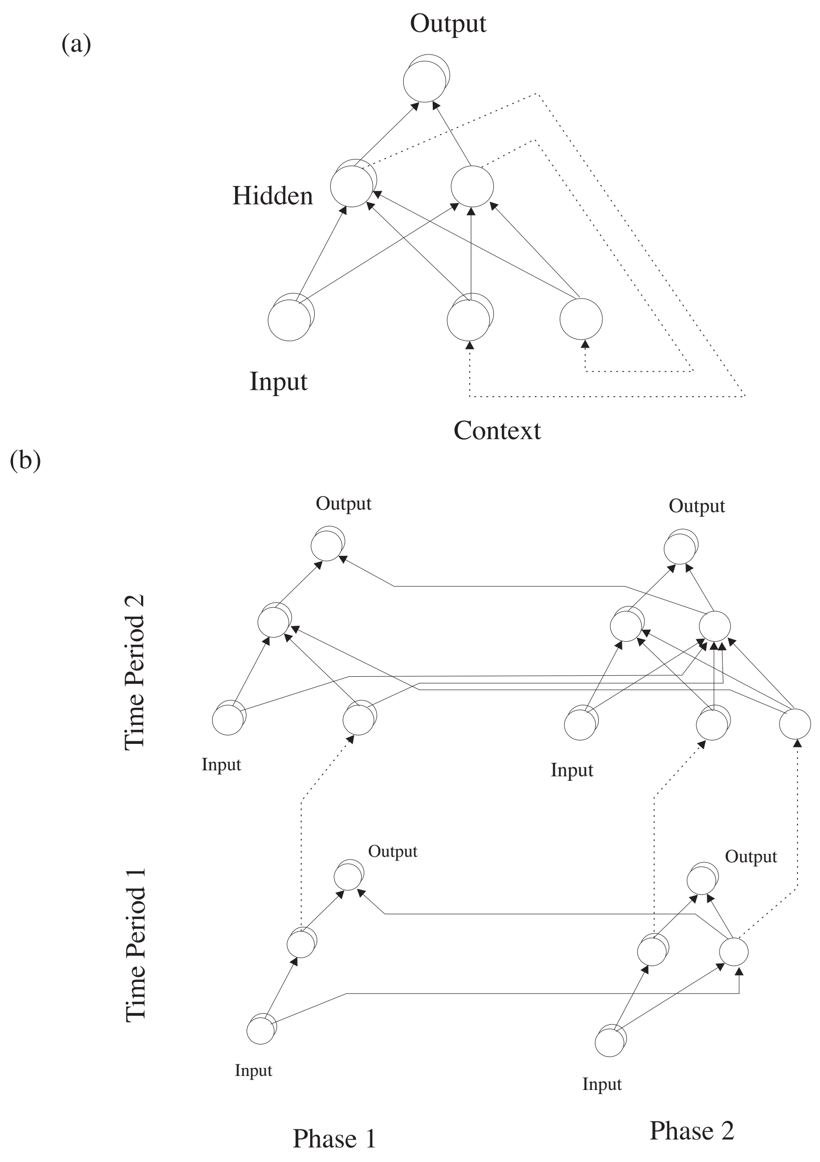


Figure 4.1: (a) A Simple TSVB Network, with one pulsing and one non-pulsing unit in the hidden layer, and (b) the network unfolded over two phases and two time periods. Pulsing units are represented by the stacked circles, and non-pulsing units the plain circle.

$$E(t) = \frac{1}{2}\sum_{t=t_1}^{t_2}\left(\sum_{k\in U_\tau}(e_k(t))^2 + \sum_{k\in U_\rho}\sum_{p=1}^{n(t)}(e_k^p(t))^2\right)$$

which yields the standard formula for the weight-update:

$$\Delta w_{ji} = \eta\frac{\partial E(t)}{\partial w_{ji}} = \eta\sum_{t=t_1}^{t_2}\delta_j(t)o_i(t)$$

In order to apply this to TSVB networks, account must be taken of the different types of units in the network, i.e. whether the units i and j are pulsing or non-pulsing units.

for a pulsing unit, $i \in U_\rho$

$$\Delta w_{ji} = \eta\sum_{t=t_1}^{t_2}\sum_{p=1}^{n(t)} R_\delta^p(j,t)o_i^p(t)$$

$$\text{where } R_\delta^p(j,t) = \begin{cases}\delta_j(t) & \text{if } j \in U_\tau\\ \delta_j^p(t) & \text{if } j\in U_\rho\end{cases}$$

for a non-pulsing unit, $i \in U_\tau$

$$\Delta w_{ji} = \eta\sum_{t=t_1}^{t_2} T_\delta(j,t)o_i(t)$$

$$\text{where } T_\delta(j,t) = \begin{cases}\delta_j(t) & \text{if } j \in U_\tau\\ \sum_{p=1}^{n(t)}\delta_j^p(t) & \text{if } j\in U_\rho\end{cases}$$

To complete these equations, the error value for each unit is needed, and the equations are analogous to those for recurrent networks. As before, the input units and context units require special treatment. By definition, input units never have an error, and context units act as place-holders for passing activation between time periods and not as computing units in their own right. Therefore, context units are used during training to pass back to their associated hidden units the sum of the error on the hidden units which they are input to, i.e. the error on a context unit for a pulsing unit j in phase p of time period t is:

$$\delta_j^p(t) = \sum_{l\in U} w_{lj}R_\delta^p(l,t)$$

and the error on a context unit for a non-pulsing unit j in time period t is:

$$\delta_j(t) = \sum_{l\in U} w_{lj}T_\delta(l,t)$$

The error for all non-context and non-input units is the sum of three factors: its intrinsic error with respect to any target output, the error fed back from units it is input to, and the error fed back from the next time step of the network if the unit has a context unit. Account must again be taken of whether the units are pulsing or non-pulsing:

for pulsing unit k,

$$\delta_k^p(t) = o_k^p(t)\left(1-o_k^p(t)\right)\left(e_k^p(t) + \sum_{l\in U} w_{lk}R_\delta^p(l,t) + \delta_{C(k)}^p(t+1)\right)$$

for non-pulsing unit k,

$$\delta_k(t) \quad = \quad o_k(t)\,(1 - o_k(t)) \left(e_k(t) + \sum_{l \in U} w_{lk} T_\delta(l,t) + \delta_{C(k)}(t+1) \right)$$

U is the set of all units, and it is assumed that $w_{lk} = 0$ if $k \notin \text{Inputs}_l$ and that $\delta_{C(k)}(t+1) = \delta^p_{C(k)}(t+1) = 0$ if $t = t_2$ or k does not have a context unit.

It is worth observing that the technique of 'unfolding' a network over time has great generality. The basic intuition is that BPTT unfolds a network over a single dimension, which is interpreted as 'time'. This interpretation arises because separate copies of the links are constrained to have the same weight, and can therefore be thought of as separate instantiations of the same link at different times. In applying BPTT to TSVB networks, the use of BPTT has been extended to *two* dimensions, one which is interpreted as time, and the second as phases within each time period. Note that the weights of links to/from pulsing units are unfolded in both dimensions, and those of links to/from non-pulsing units only in one of the dimensions (time). This process of extending BPTT may be continued, in principle, to unfolding over arbitrary numbers of independent dimensions. What is required is a sensible interpretation of these dimensions and how they will interact in a computational model. This property is referred to in Chapter 6 where further work on representing relations in TSVB networks is discussed. It has also been used in the Backpropagation Through Structure algorithm [95] discussed in Section 2.3.3, in which the network is unfolded so that a copy is made for each node within a structure, and links are made between copies to represent the relations between nodes within the structure.

4.1.3 TSVB network architectures

TSVB connectionist networks are composed of two types of unit, pulsing and non-pulsing. This means that a richer variety of architectures is possible than with standard connectionist networks, which have only one type of unit. Two broad classes of network are considered in this section. The first represents a direct introduction of TSVB units into the architecture of a Simple Recurrent Network (SRN) [18]. The second describes a different interpretation of the network architecture, extending the notion of non-pulsing units to the input layer itself, as well as incorporating the architectural restriction of not permitting links from a pulsing to a non-pulsing unit, as discussed at the end of Section 4.1.1.

In each case, the networks operate in an identical fashion, with the format for inputting information to the network being only a little different to that in standard connectionist networks. Consider a sequence of inputs 'a b c ...'. With an SRN, these inputs would be presented to the network in consecutive time periods, each input represented as a localist 1-in-n vector, i.e. one input unit representing that symbol would be activated, and the rest not. Thus, in time period 1, the SRN would receive symbol "a" on its input, and output information relevant to that symbol; in time period 2, it would receive symbol "b" on its input, and output information relevant to that symbol, but based on the whole sequence to that point, i.e. 'a b'; and so on through the sequence.

The experiments in this thesis follow the same pattern of presenting input to TSVB networks as with SRNs, with one difference. If the input units are pulsing units, then the symbol must be input in a specified phase of the current time period. The procedure adopted here is to input each symbol in a new phase, i.e. one unused by the input sequence to that point. Thus, in time period 1, the TSVB network would receive symbol 'a' on its pulsing input units in phase 1; in time period 2, it would receive symbol 'b' on its pulsing input units in phase 2; and so on. This procedure is also followed by those networks with both pulsing and non-pulsing input units. In such networks, it is assumed that there is one pulsing unit and one non-pulsing unit for each input symbol, i.e. as the input symbol is introduced in a new phase to its pulsing unit, it is also input to the network to

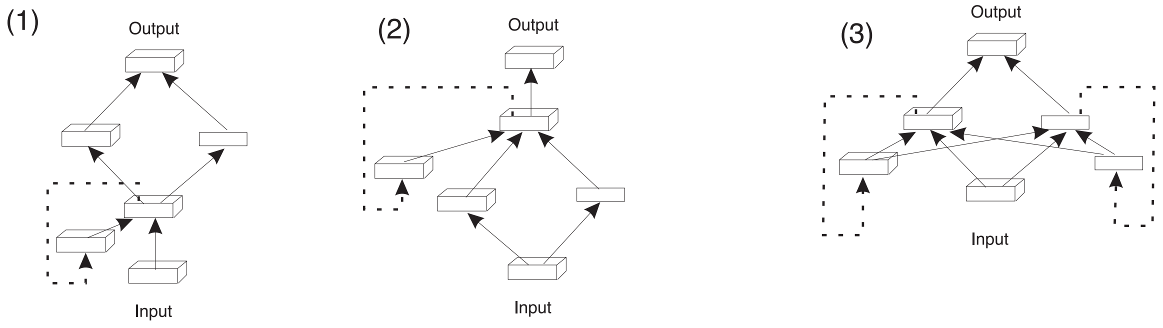


Figure 4.2: TSVB Networks: the block-shaped layers represent layers of pulsing units, the rectangular layers are layers of non-pulsing units. The dotted links are copy links.

its non-pulsing unit. The input for the non-pulsing units operates in an identical fashion to that for SRNs.

Once the data is input to the network, the rest of the network operates as defined in the previous sections. Pulsing hidden units compute information about individual entities, and non-pulsing hidden units combine information about separate entities, computing information about the situation as a whole. Finally, output is computed for the output units, which here are all pulsing units. This output computes a value for each output feature for each entity so far input to the network. The consequences of this are that the network can output complex structured information, and this ability is explored in Section 4.1.5 and the experiments which follow.

Recurrent TSVB networks (1)

The SRN [18] extends the standard feed-forward connectionist network by adding context units. Each context unit is used to hold a copy of the activation from the previous time step of its associated hidden unit, thereby providing the network with a memory for previous states. Extending this idea to TSVB networks suggests networks with pulsing context units, which provide a memory for information that has been input about entities. Non-pulsing units are used to transfer information between entities.

The question is where the non-pulsing units should go with respect to the context units. The three possibilities (after, before and during) are shown in Figure 4.2. Network type 1 uses non-pulsing units to transfer information between entities when determining the output for each entity. Network type 2 transfers information between entities when that information is input. Network type 3 transfers information between entities at every point of the recurrent component of the network. In addition, this last type provides a separate context layer of non-pulsing units, for information applicable to every entity.

Recurrent TSVB networks (2)

It was pointed out at the end of Section 4.1.1 that links from pulsing to non-pulsing units cause potential problems in generalising to large numbers of entities, and it might be worthwhile not to use such links. This seems like a crippling restriction to make, because such links to non-pulsing units are the only way to transfer information input to pulsing units between entities. However, by using an additional input layer of non-pulsing units, and using these to input information about entities that is relevant to other entities, a fully general computational architecture is retained. The pulsing layer of input units is used, as before, to input information relating to individual entities. The non-pulsing layer of input units is used to input information about any of these entities to accumulate

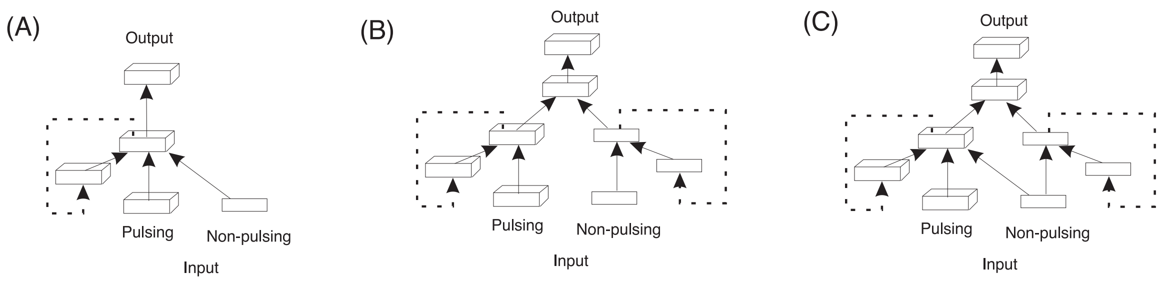


Figure 4.3: TSVB Networks with pulsing and non-pulsing inputs.

information relevant to every entity in the network. The precise input format used depends on the particular problem being dealt with. In the experiments in natural language described below, the non-pulsing inputs receive a copy of the information in the pulsing units, the separate words in the sentence, so that information about the sentence as a whole can be built up.

Given separate pulsing and non-pulsing inputs, the question is at point information from the two should be combined. Figure 4.3 illustrates the three possibilities allowed under the above restriction: before the recurrent pulsing units, after the recurrence and both. If the combination occurs after the recurrence, then a separate recurrence must be introduced for the non-pulsing units, as in types B and C. The illustrated network architectures for these types use a combination layer to help combine the pulsing and non-pulsing components of the network, but this added layer is optional.

There are some similarities between these two groups of networks with certain input representations. For instance, given that input activation is only supplied in a single phase in each time period, the operation of network type 2 is similar to that of type A. Note that the type numbers and letters, i.e. 1, 2, 3, A, B and C will be used to refer to the network architectures in Figures 4.2 and 4.3 throughout the rest of this thesis.

4.1.4 Short-term memory and other variations

Although the implementation of TSVB connectionist units presented in Section 4.1.1 appears the most direct implementation of TSVB in a standard connectionist framework, there are a number of variations and enhancements possible. The most important of these is the use of a bounded queue of active phases (called the ‘short-term memory’) to reduce the time complexity of the TSVB algorithm from being quadratic in the length of the input sequence to linear. Three subsidiary ideas are also presented, each of which, although not explored in this thesis, is a legitimate avenue for further research.

Short-term memory

The definition of TSVB connectionist networks as presented so far possesses an important source of computational inefficiency. Specifically, for every entity that is introduced in the input, one phase must be allocated for that entity within every subsequent time period. For applications involving language, one entity, and therefore one additional phase, will be introduced to the network for every word in the sentence. With samples of real language, the sentence lengths can become large and, in cases where some phases are not required in later processing, it is inefficient to recompute unnecessary phases at every cycle of computation. The experiments later in this section will illustrate two cases, one where every phase should be retained, and one where every phase is not necessary. The question to be answered is, when can the network determine that a phase is no longer required? In order to answer this question, some language-specific considerations are necessary. Specifically, the limitations of the human cognitive system suggest an extension to the TSVB connectionist network which can alleviate some of its inefficiencies in domains where human performance is a model.

Although TSVB connectionist networks in this thesis are basically presented as another algorithm with which computers can be made to learn, it is useful to look to other fields for possible extensions and improvements. Language is fundamentally a domain at which humans excel, and so some of the insights gained from studies of human learning can legitimately be considered and perhaps added to the algorithm. One of the more striking limitations of human mental abilities, e.g. [16, 70], is the relatively small number of items which can be held in short-term memory (STM). One application of this idea, developed by Baddeley [4], is the STM for verbal material, known as the audio-loop. This loop allows for a maximum of three items to be stored in the STM, and a decay mechanism means that older items will be removed from the loop unless refreshed by an attentional mechanism. The proposal here is to adapt the concept of a verbal STM and apply it to the TSVB connectionist network.

The present definition of TSVB connectionist networks uses a set of phases in each time period. During each time period, these phases are considered in turn for computing the activation of the pulsing units. The proposal here is to augment this model with a queue of 'active' phases, where 'active' refers to a phase which will be retained for computation at a later stage of the input sequence; this queue of active phases will be referred to as the STM. Two processes govern the management of STM, the first to add new phases to the queue, and the second to bring phases to the front of the queue. Note that an individual phase is identified by its referent number.

- As each phase is introduced, its referent number is pushed on to the head of STM.
- For every phase referred to in the output during the current time period, that phase's referent number is moved to the head of STM.

To illustrate this process, consider an input sequence of three items 'a b c' with each item presented to the network consecutively in phases 1, 2 and 3; let the computed output from all three items refer to item "a" alone. After the first time period, STM will contain phase 1. In the second time period, the introduction of phase 2 will add its referent number to the head of STM, leaving it as '2 1'. The subsequent use of phase 1 on the output will move this phase's referent to the head of STM, leaving it in the state '1 2'. In the third time period, introducing phase 3 places its referent at the head of STM, leaving '3 1 2'. Phase 1 is then referred to in the output, leaving STM as '1 3 2' for the next time period. As can be seen, the order of items in STM reflect the recency and the relevance of the phases; new phases and those referred to in the output will be constantly moved to the head of STM. In this STM-TSVB connectionist network, each cycle of computation only considers a fixed number of phases for computing the activation of the pulsing units; this means that phases outside of the STM are 'forgotten', enhancing the efficiency of the network.

During training, of course, the actual outputs of the network will tend to be incorrect, and therefore, if used for determining which phases should be placed into STM, this model would be unable to learn correctly. Specifically, if a phase never makes it into STM, then no training error can be propagated back in this phase. One way to address this problem is not to use the STM during training. However, this is undesirable for two reasons: first, it would lead to a mis-match of the network's operation during training and testing, and second, the STM is proposed to alleviate computational inefficiencies, which are most prominent during training. Therefore preferable is to address the problem by using the network's *target* output during training to determine the placement of phases into STM. Thus, only the relevant phases will be present in STM, and so the error fed back will be concentrated on these phases alone.

For applications to natural language, this STM-TSVB network is a very natural model due to its bounded nature. Such bounds are prevalent in cognitive functions [16, 70], and so may be expected in any higher-level domain related to a cognitive activity; a good survey of such ideas is presented by Cowan [16]. Language parsing in particular is an example of a domain where cognitive limitations may be important. For example, Henderson [36] has argued that a maximum of 10 phases are required for parsing natural language. The way this functions is clearly demonstrated with noun phrases. For example, 'The fast blue car sped down the road.' In parsing, all that needs to be retained from the phrase 'The fast blue car' is the fact that a noun-phrase has been seen. In this instance, the first word, "The", introduces the constituent for the phrase. As the words of the phrase are encountered, this constituent is continually moved to the head of STM, while intermediate words (the adjectives), which are no longer required, move to the end of STM and will, given a long enough sentence, be lost from consideration.

For TSVB networks, the use of STM has three important benefits:

- Speed : long sentence lengths cannot take up more computational time than allowed for by the maximum size of STM, i.e. processing time becomes linear in the size of the input sequence instead of quadratic.
- Cleaner output : fewer phases in the network means that there is less competition for the outputs, and so spurious errors are less likely to occur.
- Cognitive plausibility : the STM is a requirement for any adequate account of language learning, and so, besides the computational benefits, adds to any cognitive attractions of the algorithm.

The STM reduces the amount of computation which the network must perform by restricting the number of phases on which computation is performed. A different mechanism has been used to limit the amount of training within SRNs by restricting the number of unfoldings in each epoch, e.g. Reilly [81]. However, this has the effect of preventing the network from learning dependencies greater than the number of unfoldings provided (which is partly the intention, as explained in Elman [21]). Long dependencies need not, however, be lost due to the STM's bounded size. For example, consider the case where a word near the end of the sentence refers to a word near the beginning. The length of this dependency may exceed the length of STM, but if intermediate words have referred to the earlier word, then it will have been moved to the head of the queue at various points, and so need not necessarily be lost. The STM only loses dependencies in which no intermediate references have occurred, so that the long distance reference is the only reference to occur. As will be seen in Chapter 5, these are rare in the example corpus of natural language used in this thesis.

However, there is one limitation to the use of STM, and that is with those TSVB architectures where links are permitted from pulsing to non-pulsing units. In such a case, a later error fed back to the non-pulsing unit should be distributed to every phase on the pulsing unit, and this cannot be done with the definition given above. However, this limitation also coincides with the architectural restriction discussed at the end of Section 4.1.1, and so the STM is only applicable with the type A, B and C TSVB architectures described in part (2) of Section 4.1.3. (Although it can be used with *trained* networks of the more general type.)

There is also a potential danger in attempting to use an STM indiscriminately, and that is in those applications which require recall of, for example, long strings of numbers or non-related words. The first of the toy grammars used below, an abstract version of the prepositional-phrase attachment problem, is such an application, where the only reference to an earlier phase is made on seeing the last input item, requiring every preceding noun to be retained. In this case, no claim is made for the linguistic accuracy of the problem, the experiment being a test for one type of generalisation present in TSVB networks.

However, it is possible, by pre-testing the training data, to determine how many phases a TSVB network should require when operating optimally. This means that, firstly, a TSVB network with STM can be configured to use an optimal number of phases for any given task. And secondly that tests can be made on some of the computational requirements for a domain such as language, e.g. the amount of working memory required for effective learning and performance.

Other variations

There are a number of other possible variations, and in this section three variations which were considered at some stage of this thesis are presented briefly. Each of these is a plausible avenue for further development in exploring to the full this class of TSVB connectionist networks, and, of course, others may be thought of.

Firstly, the definition of the non-pulsing units is such that information is combined across each phase before passing the result through a sigmoid function. However, the role of the non-pulsing unit is to compute information across all phases, about all the entities together. This can be achieved in a simpler manner by using non-pulsing sum-units, which output their net input, the sum of their input in all phases. This removes the need for the scale factor, $o_k(1 - o_k)$, in the weight update equations, and so error is not diluted during backpropagation through non-pulsing units. However, an adequate arrangement of pulsing units is required to compensate for the lower computational power of the non-pulsing sum-units.

Secondly, it was noted that the use of continuous activation values in TSVB networks can lead to problems with links from pulsing to non-pulsing units. These problems arise where an accumulation of near zero outputs from a pulsing unit are summed over the time period by a non-pulsing unit and so exceed its threshold value. One solution is not to permit this type of link in the architecture, and this restriction is part of the definition of the Simple Synchrony Network. Alternatively, a function on the non-pulsing unit's input can be used to correct, or at least disguise, this problem. Such a function should replace the standard addition function with one which biases the sum of its inputs to 0 or 1. One example is the probabilistic-or function, defined as:

$$A + B - AB$$

where A and B are two inputs (the result can be doubled to produce output in the same range as standard addition). The idea would be to use this instead of the standard summation function when computing the net input across the time period for the non-pulsing unit.

The probabilistic-or function is associative, and therefore generalises to increasing numbers of phases. Also, it has the property of stretching the input values towards the extremes, i.e. adding a bias away from values near 0.5. Therefore, this function can be used to encourage noisy internal representations to take up values closer to binary values, a tendency which can be increased by taking fixed powers of A and B before applying the function. Further, the function is continuous and differentiable, and so can be incorporated into the weight update equations. However, this is only an ameliorating device: in order to entirely remove the problem of pulsing to non-pulsing unit links, a fully binary internal representation should be used. This, however, returns to the original implementation of TSVB for SHRUTI, and the impossibility of using standard connectionist training algorithms.

A further variation would be to remove the restriction that phases are computed independently of each other. The current implementation is motivated by the identification of phases with variables, and variables in logical propositions are independent of one another. However, removing this restriction enables the network to combine information about entities in a time-dependent way, supplementing the interaction of entities provided in the non-pulsing units.

4.1.5 Representing structure with TSVB networks

When discussing the limitations of previous connectionist attempts at handling structured representations in Chapter 2, an argument was made for a new connectionist architecture for handling mappings between sequences to directed acyclic graphs, such as parse trees. A similar argument was made in Chapter 3 with respect to connectionist approaches to natural language parsing. This section demonstrates that TSVB networks can answer these problems, that TSVB networks can convert input sequences into output representations for parse trees.

Before starting, two basic features of the input-output format in TSVB networks need emphasising. First, in all six of the TSVB network architectures illustrated in Section 4.1.3, the output layer is composed of pulsing units. In what follows, all output values are assumed to be thresholded to simple 0s and 1s, because the forms of structured representation considered do not require continuous output activations. Second, output is generated on these pulsing units in response to each element in the input sequence. In order to appreciate the class of output representations supported by TSVB networks, it is worth calculating just how much information may be output for a given input sequence. (This analysis deals with the standard TSVB network first, i.e. there is no STM bound on the number of phases.)

As described in Section 4.1.3, each element in the input sequence is presented in a new, previously unused phase. This input representation is basically the same as the standard SRN, i.e. it is a pure representation of sequences, with consecutive elements in the sequence presented in consecutive time periods. However, with TSVB networks, the pulsing inputs convey an extra piece of information: the phase in which the element has been input. This use of different phases allows each input element to be a separate entity within the output structure.

In the first time period, only one phase has been introduced to the network, and so output can only be computed for one entity. In the second time period, two phases have been introduced to the network, and so output can be computed for two entities. Similarly, in the n^{th} time period, output can be computed for n entities. By summing all these elements, $O(n^2)$ outputs are computed for a sequence of length n. This contrasts strongly with the SRN, which can only provide one set of output during each time period, and so only $O(n)$ outputs can be computed for a sequence of length n.

The next question is, What to do with these extra outputs? As discussed in Chapter 3, there are several kinds of parse tree, ranging from basic taggers to highly structured, hierarchical parse trees. The basic tagger is exemplified by the SRN. In this case, only one item of information is required for every input word, and so the extra information provided by TSVB is not required. A second type represents simple relations between words, with two pieces of information required from the parser: one about the current word and the other about any previous word to which it is related. This second type of representation is comfortably handled by the TSVB networks. The information about the current word is output in the *same phase as the current word* and the information about its relation is output in the *phase of the word to which it is related.* For example, the structure illustrated in Figure 4.4(a) has links from the second to the first items and from the fourth to the second items. When computing about the fourth item in the sequence, the item will have been input on phase 4 and the computation would be proceeding in time period 4. Information about the fourth item, such as its syntactic class, would be output during phase 4 of the current time period; information about the relation of the fourth item to the second would be output during phase 2 of the current time period, and no activation would be produced in all other phases (i.e. 1 and 3).

Although the output of relations is interesting, the resulting structure is still relatively ‘flat’. More interesting parse trees, such as those used in statistical and classical parsers, include hierarchical arrangements of information. Such hierarchies can also be output by TSVB networks. The key idea is to allow the network to output a relation between its current input and the *new phase*:

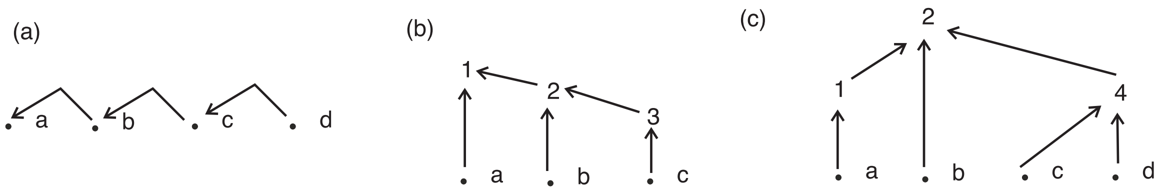


Figure 4.4: Some sample structures which can be output by TSVB networks.

this is not difficult, as output units for relations will be present for indicating relations to previous phases, and so these same outputs are simply allowed to provide information in the new phase as well. This phase can now be considered as a *node* within a parse tree. For example, the tree in Figure 4.4(b) shows an input sequence of three items, 'a b c', and an associated structure with "1" as the tree's root node. This structure may be input in the following manner. First, when "a" is input, the network indicates a relation with phase 1: this is the new phase, and is interpreted as a new node in the tree with the input "a" attached to it. Second, "b" is input. The network indicates a relation with phase 2, which is interpreted so that input "b" is attached to the new node; the network also outputs a relation with phase 1, which is interpreted so that the new node is attached to that introduced in the first time period. Similar occurs with the third input, producing the illustrated tree structure. It is also possible to label the nodes within the tree: in addition to output units for relations, output units are included for possible labels. The label for the node introduced by the new phase is then output on the label outputs in the new node's phase.

This interpretation of the pulsing outputs has one inherent limitation: all relations must be between the node introduced by the new phase and either some other node or the current input. This forces the output of the TSVB network to be *incremental*, outputting every relation between its new phase and any earlier phases. This does not prevent some interesting trees from being output, such as illustrated in Figure 4.4(c). Note that these trees require the *direction* of the relation between the new phase and the earlier phase to be indicated, which can be done by providing separate output units for the two cases (either the new phase is the parent node or the old phase is the parent node).

This demonstration that TSVB networks can output incremental representations of parse trees makes it clear that the new architecture has met its major theoretical goals of being capable of generating parse trees from an input sequence of words. The experimental sections of this thesis basically validate that this ability to generate parse trees from input sequences can be learnt. This chapter continues with experiments using two toy grammars. The first toy grammar uses the simpler type of structure, as in Figure 4.4(a), to indicate which noun in a sentence is referred to by a prepositional phrase. This grammar tests the ability of TSVB networks to generalise a relation learnt for particular lengths of sequence to longer lengths. The second toy grammar requires the networks to generate a more complex hierarchic representation. This grammar tests the ability of TSVB networks to generalise across specific features of the output structure. The next chapter introduces a more representation for parse trees, and discusses how the class of representations achievable by the TSVB network compares with representations for describing real natural language corpora.

Finally, it should be noted that this chapter focuses on testing the basic TSVB network with no resource bounds. The next chapter uses the STM to limit the computational demands of learning to parse from samples of real text. However, the use of the STM means that, instead of $O(n^2)$ items of information being output by the TSVB parser, only $O(n)$ items will be output. This, however, does not limit TSVB networks to equivalent representations to SRNs. The apparent anomaly is readily resolved by considering that TSVB adds to the SRN the *ability* to refer to any entity in the output, and so generate relations between nodes and words within an evolving parse tree. However, the *number* of such relations that are required in a given sentence is only a small subset of the total possible. In such a case, it is possible to consider resource limitations in line with the restricted possibilities of the domain. These restricted possibilities arise, in language, in two ways. First, the number of relations that might be output at one point will be limited by the number of phases in the STM. Second, the number of words which must be maintained in memory because they are needed in a relation at a later point in the sentence. The fact that each of these bounds is a feature of natural language is due to the domain's cognitive nature, and is a point returned to in Chapter 6 when discussing the importance of the STM.

It is also worth emphasising that the TSVB network with STM is capable of generalising over entities, as discussed in Section 2.3.5, because it uses time to represent the relations in the parse tree. This generalisation ability is an inherent property of the TSVB connectionist framework [37], and is additional to the generalisation abilities of the basic SRN. Section 2.3 contained more discussion on the value both of representing structure and of taking advantage of the generalisations implied by that structure.

Next, the performance of the TSVB networks is empirically validated on two toy grammars which each illustrate the representation of structured outputs and test for different generalisation abilities in the networks.

4.2 Experiments on Specific Generalisations

This section presents results from experiments training the different TSVB network architectures from Section 4.1.3 to parse sentences from two toy grammars. The grammars are chosen to highlight specific kinds of generalisation which arise when handling structured information. These include the ability to handle increasing numbers of entities and the ability to generalise across structure. These generalisations are important because they can occur only in networks handling structured information, such as the TSVB networks, and so are not naturally present in standard recurrent connectionist networks. It is assumed that the generalisation ability of recurrent networks is retained in the TSVB networks. This is because the only difference between the TSVB network and the SRN is its use of phases within each time period to represent separate entities; by presenting every input item within the same phase, the TSVB network will function identically to a standard recurrent connectionist network (although small differences may be observed due to the arrangement of hidden units).

The first experiment in this section uses an abstracted form of the prepositional-phrase attachment problem, which requires the network to select the noun in a sentence modified by the prepositional phrase. The objective of this experiment is to test the ability of TSVB networks to generalise across increasing numbers of entities. It also illustrates how TSVB networks can output a simple structure based on relations, as shown in Figure 4.4(a). The second experiment uses a recursive grammar, and follows the experiments performed by Hadley and Hayward [33]. This experiment tests the ability of TSVB networks to generalise across syntactic positions, i.e. to generalise across structured information. This experiment also illustrates how TSVB networks can output a hierarchic output structure, as shown in Figure 4.4(b).

The aim of these experiments is two-fold: first to ensure that TSVB networks can learn generalisations in the expected manner, and second to determine whether different network architectures produce different levels of performance. As will be seen, the experiments do demonstrate an ability to learn appropriate generalisations and also show that one type of architecture is superior in the levels of performance achieved.

4.2.1 Learning generalisations across multiple entities

This first experiment uses an abstract version of the prepositional-phrase attachment (pp-attachment) problem to test for an ability to learn generalisations across multiple entities. This section defines the pp-attachment problem and gives experimental results from training several examples of each of the six TSVB network architectures on this task. These results enable comparisons to be drawn between the various TSVB network architectures.

Prepositional-phrase attachment

The problem of pp-attachment is illustrated by the classic phrase 'The silly robot saw the red pyramid on the hill with the telescope' (e.g. [105]), where the prepositional phrase "with the telescope" must be attached to one of the preceding nouns: should it be attached to "robot", "pyramid" or "hill"? From a linguistic perspective, the problem of pp-attachment is resolved through many interacting constraints, both syntactic and semantic. These kinds of constraint, involving an interaction of the statistical significance of different options, are of the sort which connectionist networks are good at, given enough training data. In order to bring out the difference between the TSVB networks and SRNs, an abstracted form of the problem is used in which the relevant constraints are readily acquired by the TSVB network during training. However, the aspect of these experiments which makes them difficult for standard connectionist networks is the problem of *reference*, where a given prepositional phrase must indicate which from a preceding set of nouns it refers to: this experiment therefore requires the TSVB networks to output a simple form of parse tree, like that illustrated in Figure 4.4(a). In testing, the TSVB networks will be required to generalise what has been learnt about a particular size of input phrase to larger ones, i.e. the networks must generalise to an increasing set of entities.

The abstracted version of pp-attachment used here highlights the problem of reference. The task is simplified to one involving two types of noun and a single prepositional phrase, abstracting away from other words. The prepositional phrase can modify (attach to) nouns of one type (called N_1) but not the other (called N_2). For simplicity, one input is used per type, n_1 for N_1, n_2 for N_2, and p for the prepositional phrase. Phrases consist of a string of n_1's and n_2's, possibly followed by a p. The problem of pp-attachment is to determine which of the nouns in a sample phrase the prepositional phrase, p, should modify. For example, in the phrase '$n_2 n_1 n_2 p$', p must modify the second of the three nouns.

Three variations of this problem are considered here, all using the same input-output format. Three input units are used, n_1, n_2 and p, and three output units N_1, N_2 and m. The m output is used to indicate the modified noun. The input phrase is presented to the network in consecutive time periods, one word at a time, and each word is introduced in a previously unused phase of the current time period (as well as to any non-pulsing inputs). Output is computed for that word in the current time period, and then the next word is presented in the next time period. Activation on the output units has a different purpose for noun classes and the m outputs. These are illustrated in a trace of a sample phrase, shown in Table 4.1.

The numbers in each column of Table 4.1 indicate the phase number on which that unit is active. In the first time period, n_1 is presented on the inputs, indicated by activating n_1 in phase 1 (for the pulsing units, if any non-pulsing units are present, then they also receive input). The output for this time period is the classification N_1, indicated by activating output unit N_1 also in phase 1. In the second time period, n_2 is presented on the inputs in a new phase, indicated by activating n_2 in phase 2. The output for N_2 is activated in phase 2, and that for N_1 in phase 1 is carried forward. In the third time period, p is presented on the inputs, indicated by activating the input unit p in phase 3. The output for m is activated in the phase of the noun which p is modifying, i.e. phase 1. The outputs for N_1 and N_2 are again carried forward in their respective phases. The final output therefore shows

	Input			Output		
Period	n_1	n_2	p	N_1	N_2	m
1	1			1		
2		2		1	2	
3			3	1	2	1

Table 4.1: Trace of input-output representation for the phrase '$n_1 n_2 p$'.

Time Period	Pulsing Inputs			Non-pulsing Inputs	Outputs		
	Phase				Phase		
	1	2	3		1	2	3
1	n_1			n_1	n_1		
2		n_2		n_2	n_1	n_2	
3			p	p	n_1, m		

Table 4.2: Extended trace of input-output representation for the phrase '$n_1 n_2 p$'. Note that the column for non-pulsing inputs is optional.

the classes of the different nouns, with the m output unit referring to one of these nouns – the noun modified by p. Note that this extends naturally to longer phrases; a second appearance of n_2 in period 3 would be indicated by activating the relevant input unit in phase 3, and the output for N_2 would be active in both phases 2 and 3 in subsequent time periods.

This representation may perhaps be made clearer by considering the extended trace of the sample phrase in Table 4.2, which shows the active input and output units in each phase of each time period. The units are referred to by name, and it is assumed that within the specific set of inputs or outputs referred to, only the named unit is active, and all other units are inactive; this corresponds to the standard 1-in-n vector representation, although in rare cases more than one unit may be active in a given phase, as occurs in the last line for the output in phase 1 in the example. The table has four major columns. The first gives the time period, the second represents the bank of pulsing input units, the third the bank of non-pulsing input units, and the fourth the bank of pulsing output units. Note that the third column, describing the bank of non-pulsing input units, only applies to network types A, B and C; for types 1, 2 and 3 this column should be ignored (the network types refer to the figures in Section 4.1.3). Within each column, separate columns specify individual phases.

The representation relies on the m output unit, a pulsing unit, to make a reference to a specific noun selected from a set of nouns forming the earlier part of the phrase. The referent noun is determined by the particular function which the network has been trained to compute for making this selection. The object of these experiments is to test whether this learned function is generalised to a larger set of nouns. In principle, this generalisation should be inherent to the TSVB network because all that varies within the network for a larger set of nouns is the number of phases within the time period; the weights used by the network to compute which noun it is referring to, and the units on which to indicate the referent, are the same for every phase in that time period. Therefore the network is expected to generalise to increasing numbers of entities, and this is tested in the experiments by varying the size of the set of nouns from which this selection is to be made between training and testing. In order to avoid any biases of the network towards certain selection functions, the criteria for selecting a noun from the input set are also varied.

The first of the three experiments uses the simplest selection criterion; only one example of the first class of nouns, which p does modify, can appear in any phrase. The network's ability to generalise to greater numbers of entities is tested by adding an increasing number of nouns of the second class (which p does not modify) to the phrase. The network is trained on all phrases up to a certain length, e.g. the set of phrases up to length 3 is: 'n_1', 'n_2', 'n_1p', 'n_1n_2p', 'n_2n_1p'. In testing, the network is presented with phrases of increasing length. The network's performance on increasing lengths of phrase is then measured.

The second and third experiments use more complex selection criteria, allowing more than one noun of the first class (which p can modify) to appear within the phrase: the second experiment requires the network to learn that p always modifies the most recent n_1 in the phrase, and the third that p always modifies the first of these nouns. The training sets for these experiments consist of all phrases up to length 3. In both cases adding the phrase 'n_1n_1p' and all phrases with three nouns to the training set given above for the first experiment. Again, the test set requires generalisation to phrases with more than 3 nouns. For example, the phrase '$n_1n_2n_2n_1p$' requires the network to attach p to the last n_1 in the second experiment, and to attach p to the first n_1 in the third.

Experimental results

All of the TSVB network architectures described in Section 4.1.3 were trained and tested on all three versions of the pp-attachment task. In each case, several network sizes were trained for each architecture, and, for each size, three networks were trained, each with a different seed for randomly creating its initial weights. A constant learning rate, $\eta = 0.05$, was used for all networks. Training proceeded in all cases until the network's output for every training phrase was within 10% of its correct value; though in some cases the networks did not converge. Results are reported by giving the network's type and size (all layers had the same number of units), the average number of epochs required in training (a '+' sign by the number of epochs indicates that not all the networks converged), and the average number of errors in the output when tested. The activation value of a given output unit is counted as an error if its value is more than 40% from its target value. Results are shown separately for the noun-type outputs (N's) and the modifier output (m).

The detailed results are contained in the following tables:

Simple Two sets of experiments were run for the simplest case, where only a single modifiable noun may appear in the phrase. The first (experiment 1a) trained the networks on phrases up to length 3 (i.e. phrases with up to two n's and one p), testing on phrases of up to length 8 (i.e. phrases with up to seven n's and one p). Results from this experiment are shown in Table 4.3. The second (experiment 1b) trained the networks on larger phrases, up to length 6, testing on phrases of up to length 11. Results from this experiment are shown in Table 4.4.

Recency This experiment requires the network to learn that the prepositional phrase should only be attached to the most recent modifiable noun in the phrase. The training set consists of all phrases up to length 4, including phrases with more than one example of n_1; the p must modify the last n_1 in the phrase. The networks are tested on progressively longer phrases, containing up to 6 nouns. For example, the phrase '$n_1n_2n_2n_1p$' requires the network to attach p to the last n_1. Results are shown in Table 4.5.

Anti-recency This experiment is identical to the previous one, except that the network must learn to attach the prepositional phrase to the first modifiable noun in the phrase. The training set is identical, with all phrases of up to length 4, but p must now modify the first n_1 in the phrase. Results are shown in Table 4.6.

Network Size	Training Epochs	Test Phrase Length (% correct)							
		5		6		7		8	
		N's	m	N's	m	N's	m	N's	m
Type 1									
No network converged during training									
Type 2									
3	633	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
6	500	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
10	367	100.0	100.0	99.5	99.4	99.2	99.6	99.0	99.0
Type 3									
3	1031	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
6	715	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
10	471	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
Type A									
3	403	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
6	340	100.0	100.0	100.0	100.0	100.0	100.0	99.3	97.5
10	263	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
Type B									
3	1067	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
6	403	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
10	287	100.0	100.0	100.0	100.0	100.0	100.0	99.3	97.6
Type C									
3	675	100.0	100.0	99.2	99.4	97.8	96.3	96.5	93.3
6	450	100.0	100.0	100.0	99.4	99.7	97.2	99.7	97.6
10	293	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0

Table 4.3: Experiment 1a: networks trained on phrases up to length 3. Test phrases of length n contain $n - 2$ n_2's with a single n_1 in all possible arrangments, followed by the p. Results are averaged over three networks.

Discussion

Experiment 1 contains the cleanest test for the ability of TSVB networks to generalise to increasing numbers of entities. However, the strict requirements of the task make this a difficult problem for the networks: the network's only function is to memorise the list of input entities, and then select the appropriate one on seeing the prepositional phrase. In order for the recurrent part of the network to memorise the list effectively, high weights must be learned on the recurrent links. This is easier with longer training sequences, which explains the superior performance of the networks in experiment 1b (Table 4.4) over those in experiment 1a (Table 4.3).

In experiment 1a (see Table 4.3) most of the network types coped fairly easily with this task. The exception being networks of type 1. These networks, during training, reached a static performance level, with only a small continued decrease in output error: for example, a network with 3 units in each hidden layer, after 3,700 epochs, possessed 36 N-type errors (errors on the N_1 and N_2 outputs only) and 12 m-type errors (errors on the m output only) with a sum-squared error of 16.9. A second attempt was made using a lower learning rate of $\eta = 0.005$, to facilitate gentler learning. This led to some improvement, but still no fully trained networks were achieved. For the other network types, some of the networks reliably produced perfect generalisation to the various test sets. In particular

Network Size	Training Epochs	Test Phrase Length (% correct)							
		8		9		10		11	
		N's	m	N's	m	N's	m	N's	m
Type 1									
No network converged during training									
Type 2									
3	203	100.0	100.0	99.6	97.5	98.2	92.1	96.3	88.9
6	50	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
10	50	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
Type 3									
No network converged during training									
Type A									
3	430	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
6	65	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
10	52	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
Type B									
3	171	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
6	69	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
10	53	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
Type C									
3	190	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
6	100	100.0	100.0	100.0	99.8	100.0	99.2	99.8	98.5
10	50	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0

Table 4.4: Experiment 1b: networks trained on phrases up to length 6. Test phrases of length n contain $n - 2$ n_2's with a single n_1 in all possible arrangments, followed by the p. Results are averaged over three networks.

the type 3 architecture produced perfect generalisation in all cases.

For experiment 1b (see Table 4.4) a similar pattern to that in experiment 1a was seen. However, the type 3 networks now failed to learn the task at all, in a similar fashion to that of the type 1 networks. Training failed to produce any improvement after a certain point, and the process was terminated: for example, with a network with 3 units in each hidden layer, a reduced training rate of $\eta = 0.005$ was used to try to encourage gentler learning, but after 84,100 epochs performance was only 29 errors out of 645 outputs, with a sum-squared error of 13.00. When tested, this network produced no errors for the length 7 test set, and then 10 errors for the length 8, and more for the other lengths. For the other network types, performance was enhanced over that in experiment 1a; as can be seen, these levels were very good, with few networks producing any errors at all.

In experiment 2 (see Table 4.5), neither the type 1 nor type 3 networks produced any successfully trained networks. This task is obviously a more complex one for the rest of the networks, and no network size produced a reliably perfect generalisation performance, although in all cases performance was in excess of 90%. However, one example of type 2 (with 10 units in each layer), two examples of type A (one with 3 and one also with 10 units) and one example of type C (with 6 units in each layer) did produce perfect generalisation across the three test sets.

As with experiment 2, in experiment 3 (see Table 4.6), neither the type 1 nor type 3 networks produced any successfully trained networks. Again the task is a more complex one for the rest of

Network Size	Training Epochs	Test Phrase Length (% correct)					
		5		6		7	
		N's	m	N's	m	N's	m
Type 1							
No network converged during training							
Type 2							
3	10933	99.8	97.0	99.7	96.0	99.3	95.5
6	730	100.0	100.0	100.0	99.8	100.0	99.6
10	533	100.0	99.7	100.0	99.3	100.0	98.9
Type 3							
No network converged during training							
Type A							
3	18766.7+	100.0	100.0	100.0	99.8	99.7	99.3
6	750	100.0	100.0	100.0	99.7	100.0	99.3
10	276.7	100.0	99.7	100.0	99.4	100.0	99.1
Type B							
3	1883.3	100.0	99.5	99.0	97.3	97.4	91.6
6	953.3	100.0	99.0	100.0	99.0	100.0	97.3
10	803.3	100.0	99.2	100.0	97.0	100.0	95.9
Type C							
3	466.7	100.0	100.0	99.7	100.0	99.2	100.0
6	400	100.0	99.7	100.0	99.2	100.0	99.2
10	466.7	100.0	99.5	100.0	98.7	100.0	97.4

Table 4.5: Experiment 2: with recency. Test phrases of length n consist of all possible arrangements of n_1's and n_2's followed by the p. Results are averaged across 3 networks.

Network Size	Training Epochs	Test Phrase Length (% correct)					
		5 (/675)		6 (/1953)		7 (/5292)	
		N's	m	N's	m	N's	m
Type 1							
No network converged during training							
Type 2							
3	2400	100.0	98.6	100.0	97.0	100.0	95.0
6	683	100.0	100.0	100.0	95.5	100.0	89.2
10	483	100.0	99.5	100.0	97.5	100.0	95.0
Type 3							
No network converged during training							
Type A							
3	3400+	98.9	95.3	98.0	91.4	97.2	87.6
6	433.3	100.0	99.5	100.0	98.1	100.0	97.2
10	400	100.0	99.3	100.0	95.8	100.0	80.2
Type B							
3	1733.3	100.0	96.5	100.0	94.3	99.5	91.7
6	433.3	100.0	97.8	100.0	97.6	99.5	97.3
10	803.3	100.0	98.0	100.0	94.8	100.0	91.4
Type C							
3	500	100.0	98.7	99.5	98.1	98.1	97.2
6	333.3	100.0	98.5	100.0	96.7	99.7	94.9
10	300	100.0	98.3	99.9	95.8	99.3	94.1

Table 4.6: Experiment 3: with anti-recency. Test phrases of length n consist of all possible arrangements of n_1's and n_2's followed by the p. Results are averaged across 3 networks.

Phase	Word	Outputs S	Su	V	Ob	R
1	<start>	1				
2	Jane	1	2			
3	who		2			3
4	calls			4		3
5	Vicky				5	3
6	knows	1		6		
7	Mary	1			7	
8	who				7	8
9	likes			9		8
10	Bill				10	8
11	who				10	11
12	helps			12		11
13	Jane				13	11

Table 4.7: Trace of a sentence parse with embeddings (the numbers under the output units indicate in which phase that output unit is on; no number means the unit is off throughout the time period).

the networks, and no network size produced a reliably perfect generalisation, although in most cases performance exceeded 90%. The best individual performances included one example of type 2 (with 10 units in each layer) which only produced a single error in the last test set, and one example of type A (with 6 units) which produced only one error in the test set of length 6 and seven errors in the test set of length 7.

The first question these experiments were intended to answer was whether any TSVB network could generalise across increasing numbers of entities. This has been demonstrated with the results particularly in experiment 1b. The last two experiments tested this same ability but requiring a more complex function to be learnt, and this obviously caused some problems to the networks. However, both the type 2 and the type A produced some networks which generalised well, although not reliably so across different weight seeds.

The second question for these experiments was whether there was any empirical difference between the different network architectures. Again, the answer has to be yes. The most dramatic difference being the failure of the type 1 and type 3 networks to handle the problems at all. This in spite of type 3's success on experiment 1a. For the other network types, there is not a significant amount of difference: all use roughly similar amounts of training, and produce similar patterns of generalisation results. The fact that the type 2 and type A networks do slightly better on the final experiments is interesting, but not conclusive: they also happen to be the simpler of the network types, and this fits with the usual trend in connectionist networks, i.e. the smaller networks tend to generalise better.

The fact that some of the network architectures do learn this task well demonstrates that TSVB networks can generalise to increasing numbers of entities. In the next set of experiments, a more complex grammar is used to test these networks for a different set of generalisations.

4.2.2 Learning generalisations across syntactic constituents

A more complex toy grammar was used by Hadley and Hayward [32, 33] to test whether a specific Hebbian connectionist network learns in accordance with the property of strong semantic systematicity. Their recursive grammar is given in Figure 2.9 of Section 2.3.4 and can be used to test for two particular kinds of generalisation. Firstly, the network can be tested for an ability to recognise a *substitution* of words of the same syntactic class. This occurs, for example, when a noun seen only in subject positions of training sentences appears in the object position of a testing sentence; information about that noun should be generalised to the noun in the new syntactic category if other nouns have been observed to appear in both subject and object positions. Secondly, the network can be tested for an ability to recognise a *recursive* structure. For example, after learning about sentences with a single relative clause, generalisation should occur to test sentences with multiple relative clauses. These two generalisation abilities are like those in compositional grammars, and the experiments here follow Hadley and Hayward's procedure (described in Section 2.3.4) for testing a network for them. In addition, this experiment requires the TSVB networks to output a hierarchical form of parse tree, like that illustrated in Figure 4.4(b).

By analogy with Hadley and Hayward, target input and output representations are defined as follows. There are 22 binary input units, each one representing a separate word. Five output units are used: 'S' (representing sentence), 'Su' (subject noun), 'V' (verb), 'Ob' (object noun) and 'R' (relative pronoun). The target output for a sentence is an incremental representation of its parse tree, so that the combination of all the outputs for all the words in the sentence will form the output parse tree. For example, the sentence 'Jane who calls Vicky knows Mary who likes Bill who helps Jane' will be processed as follows (refer to Table 4.7).

The sentence is begun by activating the relevant input unit of the network for the input symbol '<start>' in time period 1 in an unused phase, phase 1 in the table (as well as to its non-pulsing input, if any). This introduces the phase for the root or 'S' node in the parse tree, and also resets the network. The first word 'Jane' is then input in time period 2 in the next unused phase, phase 2, by activating the relevant input unit, and, as this introduces a subject noun, the 'Su' output unit becomes active in phase with the input word. Also, as the noun is part of the main sentence, it is attached to the 'S' node in the parse tree, and this is indicated by activating the 'S' output in phase with the '<start>' symbol. As the table shows, each word in the sentence is associated with a unique phase number, because it is introduced to the network in a new, previously unused phase. This phase number is used by the output units to indicate precisely which word in the sentence the current output is related to. This representation makes clear how embedded clauses can use the same set of output units and yet unambiguously identify the appropriate embedding level. For example, the verb "helps" in phase 12 is part of the relative clause introduced by the "who" from phase 11, which is modifying the noun "Bill" in phase 10.

Experiments analogous to those described in Section 2.3.4 from [33] were performed. The recursive grammar from Section 2.3.4 was used to generate a training set of 1370 sentences, either simple or with one relative clause. Only four of the twelve nouns were permitted to occupy any syntactic position, four were constrained to appear in subject position, and four object position. In addition, only 25% of the training set was composed of sentences with one relative clause, the other 75% being simple sentences with no relative clauses. After training, each network is tested on a set of 6000 sentences, comprising 1500 sentences with equal numbers with zero, one, two, or three levels of embedded relative clauses and nouns occupying any legal position; no sentence in the training set can appear in the test set.

The networks were trained with a learning rate, η, of 0.05 and an output tolerance of 0.1 (10%). Table 4.8 shows the experimental results: the network size is followed by the number of epochs required (an X indicates those networks did not converge). For the test set of 6000 sentences, the percentage of errors at the 40% level are given. All results are the average of three networks of the same size but different weight seeds.

With these experiments, networks of types 1 and 3 did not converge during training. The performances of the four network types are all quite close, though types B and C produce about half

Network Size	Training Epochs	Percentage correct
Type 1		
No network converged during training		
Type 2		
5	X	0.0
8	33	96.5
12	13	98.0
Type 3		
No network converged during training		
Type A		
5 (2 nets)	150	95.2
8	32	97.4
12	17	98.0
Type B		
5	25	98.8
8	18	98.5
12	12	98.1
Type C		
5	7	98.2
8	13	98.8
12	6	98.3

Table 4.8: Results for recursive grammar experiment: figures given are averaged across three networks.

the number of errors of types 2 and A. The important conclusion from this experiment is that the networks do generalise very well to the test set. As discussed in Section 2.3.4, the grammar and the training and test sets are used by Hadley and Hayward to test for evidence of generalising across syntactic structures. The test set contains sentences with words in novel syntactic positions and also with a greater depth of recursive structure. The performance of the TSVB networks on this problem demonstrate that they can handle generalisations across syntactic structure. As a connectionist architecture, they may also be compared with the Hebbian connectionist network introduced by Hadley and Hayward [33] specifically to learn these types of generalisation. This point was discussed in Section 2.3.5. (This experiment, with some early results, was described in [54].)

4.3 Conclusion: Defining the Simple Synchrony Network

This chapter has presented a new implementation of Temporal Synchrony Variable Binding (TSVB), which defines TSVB units with differentiable activation functions. A novel extension of Backpropagation Through Time has been developed, which enables such TSVB networks to be trained, and a range of such TSVB network architectures has been described. Using two toy grammars, generalisations specific to TSVB networks have been tested, and it has been seen how the different architectures achieve different performances. In particular, the restriction that no pulsing unit can provide input activation to a non-pulsing unit, as in the type A, B and C networks, has been shown to provide superior generalisation performance. Note also that, because only one item is input at a time, the type 2 network effectively does not possess such links either; only one phase can be active on the input units at any time and so the non-pulsing units have only the one active phase on their input, making the summation a redundant operation. This explains the equally good performance of the type 2 network in this specific context. This observation, supported by the theoretical argument at the end of Section 4.1.1, is the basis for the following definition:

Simple Synchrony Networks are recurrent TSVB connectionist networks *without* links from pulsing to non-pulsing units

Note that this definition only rules out the presence of weighted links whose source is a pulsing unit and whose target is a non-pulsing unit. This does not mean no *interaction* occurs between the two types of unit. Such interaction is a result of training, where the same output error is fed back to both the pulsing and non-pulsing elements of the circuit. Because this error is the result of a computation based on information in both the pulsing and non-pulsing parts of the network, the backpropagated error will force the two separate elements to recognise and adapt to the computations performed by the other. This means that the weights within the separate parts of the network are trained in relation to the weights in other parts. Hence, the computations performed at separate pulsing and non-pulsing units will complement each other.

The generality of the Simple Synchrony Networks as a computational architecture depends on how the input format interacts with the two paths through the network, before they are combined to form the output. Essentially, enough information must be passed down both the pulsing and non-pulsing paths so that their interaction provides the output with enough information to compute relationships between the different entities in context with the rest of the sentence. In the case of the type B and C networks, the non-pulsing element is effectively a Simple Recurrent Network (SRN), and so, with an appropriate input representation, this element will compute with the same generality as an SRN. For the types A, B and C, each entity input to the network enters a recurrent pulsing element of the network. This ensures that information about each entity is retained over the lifetime of the sequence. The non-pulsing information, in the case of the type A and C networks, is combined with these entities in every time period, and so information is built up for each entity in context with the rest of the sequence. For the type B network, the context of each entity is only supplied at the combination layer, just before the output layer; however, this seems adequate in the experiments performed here. In all the experiments in this thesis, the input is a sequence of words or word-tags within a sentence. In each case, the separate words (or word-tags) are input both on the pulsing inputs, as separate entities, and on the non-pulsing inputs, to provide a context. Hence, with this representation, the architectures convey enough information about both entities and context to the output units.

The Simple Synchrony Network (SSN) combines elements from a number of different areas. In particular, the SSN combines the ability of SRNs to learn about patterns across time with the ability of TSVB to represent multiple entities and generalisations across them. These abilities make the SSN capable of handling a wide range of domains. Specifically, these abilities are suited to problems such as natural language processing requiring the output of structured information. The next chapter explores the ability of the SSN to learn to parse sentences drawn from a corpus of naturally occurring text.

Chapter 5

Experiments in Learning to Parse Natural Language

The Simple Synchrony Network (SSN) defined in the previous chapter is a new connectionist architecture which combines the ability of Simple Recurrent Networks (SRNs) to learn about patterns across time with the ability of Temporal Synchrony Variable Binding (TSVB) to represent multiple entities and generalisations across them. The basic generalisation abilities of the SSN have been tested on two toy grammars. These experiments have shown that the SSN can generalise information to increasing numbers of entities (as in the pp-attachment problem) and across syntactic constituents within a parse tree (as in the recursive grammar). In this chapter the SSN is applied to a more complex problem using natural data: learning to parse sentences drawn from a corpus of naturally occurring text. This application is an example of a real world problem requiring the output of structured information. This chapter describes the natural language corpus and input/output representation used for the experiments, and then presents results from experiments with the SSN in learning to parse from this corpus.

5.1 Data for Parsing with the Simple Synchrony Network

This section describes the data and input/output format used in the experiments for training the Simple Synchrony Network (SSN) to parse samples of natural language. The data set used is a corpus of sentences taken from samples of real natural language, and the bulk of this section is taken up with describing an input-output representation suitable for use by SSNs when applied to this task. The intention is to emulate the format of experiments performed with the Probabilistic Context-Free Grammar (PCFG) described in Section 3.3, but with a model of connectionist learning. As shown in Section 3.4, previous attempts at connectionist language learning have not produced results comparable to those of the statistical language learning community, instead relying on restricted output representations or toy grammars. The experiments described here use the SSN to generate parse tree representations from samples of naturally occurring text, and the parser is evaluated in the same terms as statistical parsers.

Because these experiments are partly exploratory, designed to investigate the ability of the SSN to learn to parse samples of naturally occurring text, a relatively small corpus has been used; the SUSANNE corpus contains around 130,000 words compared with the several millions of the Penn Treebank. For this reason, word-tags are used as the input to the network, instead of the actual words. The use of word-tags means that, firstly, a smaller database is required for learning the same grammatical information. This is because the occurrence of the word-tag “NP”, for example, will be more frequent than the occurrence, individually, of the words “Mary”, “John” etc, which are instances of that tag. Therefore, the information to be learned about the word-tag "NP" should be more readily available to the network, and not need inferring from the individual nouns within the database. Secondly, the actual number of input units in the network, and therefore the number of links from these units to the rest of the network, is reduced. These two factors have the desirable effect of reducing training times. Therefore, the input-output mapping for the network to learn will be one from a sequence of word-tags to a target parse tree.

This section continues as follows. Firstly, the SUSANNE corpus is described, with particular reference to the information relevant to syntactic parsing. Secondly, the appropriateness of the SUSANNE parsing scheme is discussed with relation to the specific properties of the SSN. As will be seen, a small modification is required to the SUSANNE representation of parse trees to accommodate a limitation in the representations of output structure currently used by the SSN. Thirdly, the GPS output representation is defined, which allows the network to define the constituents of the parse tree. This representation retains the complexity of the language learning task, but takes into account the restrictions imposed by the SSN's representation by converting the corpus from one parsing scheme (that of SUSANNE) to a different parsing scheme (similar to a dependency grammar). The process of converting the SUSANNE corpus into this output representation is then detailed. Finally, when the SSNs are tested, their outputs must be compared with the target parse tree obtained from the corpus. This evaluation process, based on the same precision/recall measures used for evaluating the PCFG, is also described.

This chapter continues with Section 5.2, which describes the experiments performed on this database.

5.1.1 The SUSANNE corpus

The SUSANNE corpus (Release 4, November 1994) is used in these experiments as a source of pre-parsed samples of natural language; the acronym SUSANNE stands for 'Surface and Underlying Structural Analyses of Naturalistic English'. The database is composed of texts taken from the Brown Corpus, but is parsed using a classification scheme extended from one devised at Lancaster University. The corpus is described in detail in Sampson [87] and the original work at Lancaster University is covered in Garside et al. [28]. This section describes the format of the SUSANNE corpus and the information in the corpus used in the experiments. Section 5.1.4 explains how the corpus is converted into the format required in the specific experiments discussed in this thesis.

The SUSANNE corpus is the result of applying a scheme for representing all formally specified aspects of English grammar to a sample of the Brown Corpus. This scheme provides for an information-rich description of any English sentence. The corpus is divided into four genres, each genre containing sixteen texts. A sample of the corpus is shown in Table 5.1.

The data is arranged into six fields, shown by the six columns in the table. The first is the *reference* of that line within the corpus, and is a unique code for that line. The initial letter, 'A' in this example, indicates the genre of this piece of text, and is the same as those used in the original Brown Corpus – 'A' refers to 'press reportage'. The next two numbers, '01' in the table, refer to the text number of which this particular sentence is taken. The rest of this field forms the unique identifier for this line within the corpus. The second field is the *status* of this line. It is not usually used, and '-' is the most common entry, although it can indicate an error in the text, or that the word is an abbreviation or symbol. The third and fourth fields indicate respectively the *word-tag* and *word* of the particular word or symbol appearing at that point of the sentence. The fifth field, *lemma*, shows the word in canonical form, i.e. without grammatical inflections. For example, the seventh line in the table shows the word 'said' and its canonical form 'say'. Finally, the sixth field, *parse*, gives information about the parse tree for the sentence.

The experiments with the SSN use a target input-output mapping based on mapping a sequence

reference	status	word-tag	word	lemma	parse
A01:0010a	-	YB	<minbrk>	-	[Oh.Oh]
A01:0010b	-	AT	The	the	[O[S[Nns:s.
A01:0010c	-	NP1s	Fulton	Fulton	[Nns.
A01:0010d	-	NNL1cb	County	county	.Nns]
A01:0010e	-	JJ	Grand	grand	.
A01:0010f	-	NN1c	Jury	jury	.Nns:s]
A01:0010g	-	VVDv	said	say	[Vd.Vd]
A01:0010h	-	NPD1	Friday	Friday	[Nns:t.Nns:t]
A01:0010i	-	AT1	an	an	[Fn:o[Ns:s.
A01:0010j	-	NN1n	investigation	investigation	.
A01:0020a	-	IO	of	of	[Po.
A01:0020b	-	NP1t	Atlanta	Atlanta	[Ns[G[Nns.Nns]
A01:0020c	-	GG	+<apos>s	-	.G]
A01:0020d	-	JJ	recent	recent	.
A01:0020e	-	JJ	primary	primary	.
A01:0020f	-	NN1n	election	election	.Ns]Po]Ns:s]
A01:0020g	-	VVDv	produced	produce	[Vd.Vd]
A01:0020h	-	YIL	<ldquo>	-	.
A01:0020i	-	ATn	+no	no	[Ns:o.
A01:0020j	-	NN1u	evidence	evidence	.
A01:0020k	-	YIR	+<rdquo>	-	.
A01:0020m	-	CST	that	that	[Fn.
A01:0030a	-	DDy	any	any	[Np:s.
A01:0030b	-	NN2	irregularities	irregularity	.Np:s]
A01:0030c	-	VVDv	took	take	[Vd.Vd]
A01:0030d	-	NNL1c	place	place	[Ns:o.Ns:o]Fn]Ns:o]Fn:o]S]
A01:0030e	-	YF	+.	-	.O]

Table 5.1: A Sample from SUSANNE - A01

of word-tags to a parse tree. Therefore, the relevant fields from the SUSANNE corpus are the third and the sixth, the word-tags and the parse, respectively. These two fields are now described in more detail.

Word-tags

The word-tags used in the SUSANNE corpus are an extension of those used in the 'Lancaster' tagset [28]. The extended word-tags provide a rich source of information about the words within the corpus.

The word-tags appearing in the corpus begin with two or three capital letters, which define the original word-tag from the Lancaster tagset. The lower case letters are the refinements added with the SUSANNE scheme. For instance, "revealing" is tagged "VVG" (present participle of verb) in the Lancaster scheme, but as "VVGt" (present participle of transitive verb) in the SUSANNE scheme. There are 353 distinct word-tags used in the SUSANNE scheme, which have a hierarchical arrangement, e.g. word-tags for nouns begin with 'N' and those for verbs with 'V'.

Because the additional grammatical distinctions used by the SUSANNE scheme would partially lead to a recurrence of some of the sparse data problems inevitable with the use of words, the experiments reported here restrict themselves to the first part of the word-tag, i.e. the two or three capital letters which define the original word-tag from the Lancaster tagset. This avoids the problems with sparse data, when a particular word-tag does not occur frequently enough within a dataset for useful information about it to be learned. It also leads to a more parsimonious set of input data, as the SUSANNE extensions are present only in restricted settings to cover specific semantic uses of words. Thus, the word-tags used in the experiments are those of two or three letters from the Lancaster tagset.

Parse information

The sixth field of the corpus provides the information for the grammatical structure of the texts as a sequence of labelled parse trees. Each line in the corpus represents a separate leaf node in the tree.

The canonical form for the text is a sequence of "paragraphs" separated by "headings". The symbol "<minbrk>" is used as the standard separation between paragraphs, and may be seen in the word field of the first line of the example sentence in Table 5.1. Each heading or paragraph, conceptually, is a labelled tree with a root node labelled "O" ("Oh" for a heading), and with a leaf node labelled with a word-tag for each SUSANNE word or trace.

This tree is represented in the standard way as a bracketed string, with the labels of nonterminals written 'inside' both the opening and closing brackets. This bracketed string is then adapted in the following manner for inclusion in successive SUSANNE parse fields. If an opening bracket immediately follows a closing bracket, the string is segmented at that point, so that one segment appears on each leaf node – e.g. between the reference lines `A01:0010f` and `A01:0010g` in Table 5.1, where the segment representing the subject noun phrase ends, and the verb occurs. Secondly, wherever the word-tag appears within the segment, that tag is represented by a full-stop. Therefore, each line of the parse tree contains one full-stop, which corresponds to a terminal node labelled with the contents of the word-tag field.

There are three types of information contained in the nonterminal node labels of the SUSANNE scheme: a Formtag, a Functiontag and an Index, in that order. The Formtag is separated from the other two types by a colon. The Functiontag is always a single alphabetic character, and the Index is a three digit number; these two items are used to represent semantic relationships between different clauses. The labels on the constituents in the later experiments use some of the information contained in this field. Specifically, the first letter of the Formtag is used, as it gives the primary

Clause tags	Phrase tags
Main clause (S)	noun phrase (N)
adverbial/nominative clause (F)	verb phrase (V)
particle/infinitive clause (T)	adjective phrase (J)
reduced clause (Z)	adverb phrase (R)
miscellaneous verbless clause (L)	prepositional phrase (P)
special 'as' clause (A)	determiner phrase (D)
'with' clause (W)	numeral phrase (M)
	genitive phrase (G)

Table 5.2: Parts of speech used in the SUSANNE corpus for labelling constituents

classification of that bracket; the tags used are shown in Table 5.2 – note that they are an enlarged set of the standard categories in Government and Binding theory, which were give in Table 3.3.

5.1.2 Appropriate parse trees for learning

Johnson [48] has warned that the representation used for a parse tree within a corpus need not be the one most suited for learning by a parser. In support of this, he shows that a simple relabelling of the nodes within a parse tree can amount to an 8% change in performance, halving the different between a simple PCFG parser and the best broad-coverage parsers available. These results suggest that some flexibility should be used in determining the form of target parse tree for training a parser. But more importantly, the form of the parser may instead constrain the form of parse tree which it can be trained on. Such a constraint was met with the holistic parsers, in which the requirement of a sequential encoding of the parse trees meant that each tree had to be of fixed valency. This section discusses the impact a limitation in the current input/output format of the SSN has on the form of target parse tree.

This limitation of the SSN is the number of nodes which may be used in the output parse tree. As discussed in Section 4.1.5, each node within the output parse tree is represented as a separate phase within the SSN, and each phase is introduced at the time of input of each word. Thus, the number of nodes within the parse tree is directly proportional to the number of input words; in this thesis, it is assumed that only one new phase may be introduced per word, so the output parse tree can only possess as many nodes as there are input words. This limitation applies not only to the complete sentence, but to every constituent within it: the number of nodes required in the parse tree representing each constituent may not exceed the number of words forming that constituent. This has the effect, within a hierarchy, of forcing each constituent to have more words than the total of the words in all its subconstituents; this is because each constituent must have at least one direct word child.

Assuming that an adequate number of nodes have been introduced, the SSN is perfectly capable of representing the (acyclic) sets of relations between them typical of parse trees. As discussed in Section 4.1.5, the SSN's pulsing outputs can be used to output a relationship between the current word and the new node just introduced, or between the current word and any of the nodes introduced by previous words. This enables the SSN's output to represent arbitrary forms of hierarchy within a parse tree. Hence, the SSN is capable of outputting any form of parse tree, restricted only by the limitation on the number of nodes. The remaining question, with respect to the SUSANNE parsing scheme, is the severity of this limitation.

It is not difficult to find examples which violate this limitation, a common one being the S–

VP division. For example, in the sentence ‘Mary loves John’, a typical encoding would be: [S [NP [N Mary]] [VP [V loves] [NP [N John]]]]. The linguistic head of the S (the verb “loves”) is within the VP, and so the S does not have any word-tags as immediate children. The solution adopted here is to collapse the S and VP into a single constituent, producing: [S [NP [N Mary]] [V loves] [NP [N John]]]. The same is done for other such constructions, which include adjective, noun, determiner and prepositional phrases. Although this procedure is a simplification, it has the linguistic precedent of bringing the resulting representation closer to that of a dependency grammar, as described in Melčuk [67].

The actual severity of this limitation may be quantified in terms of the training data used below. In Section 5.2 a training set is selected from the SUSANNE corpus, and it is possible to count the type and frequency of each change made to the SUSANNE corpus with respect to the SSN’s limitation in the number of output nodes. The most common changes are caused by the verb phrases: the S–VP change, which occurs once per sentence, appears 265 times, and the similar change within relative clauses appears 28 times. Apart from these two, the changes are minor. For instance, the next most common problem is the use of an extra quote tag around a sentence, e.g. [S [Q [NP Mary] ...]]. This occurs 13 times, and is a redundant encoding, and hence safely ignored. Finally, there are a series of less frequent changes, reflecting various linguistic constructions. 12 of these occur in the entire training set. As this set contains over 1,580 constituents, it can be seen that, practically, the limitation of the SSN is not too severe. In addition, it would be possible to artificially reintroduce most of the verb phrases on output, leaving only around 30 irrecoverable changes.

What is at issue in this thesis is not, however, the precise nature of the parse tree representation produced by the SSN parser. Instead it is the ability of the SSN parser to *learn* to generate parse trees from a corpus of naturally occurring text. As shown in earlier chapters, other architectures, such as the RAAM, can be used to generate parse trees (albeit with some restrictions). However, the task of scaling up a parser based on RAAMs from certain toy grammars has yet to be demonstrated. More important than the precise form of representation output by the parser, which is largely an artifact of linguistic coding schemes, is the ability to learn appropriate generalisations for natural language. And this is what will be tested later in this chapter. For now, a representation for a slightly simplified form of the SUSANNE parse trees is introduced in the next few subsections. Chapter 6 contains some suggestions for altering the SSN’s output representation to better support the parse trees within the SUSANNE corpus.

5.1.3 The GPS output representation

This section introduces a specific input-output representation which uses a sequence of word-tags as an input and a sequence of parent-child relationships to describe an output parse tree. The sequence of parent-child relationships enables the SSN to incrementally output information about the parse tree in response to each input word-tag. Once every word-tag in the sentence has been processed, the total sequence of outputs will describe the entire parse tree.

In order to build up a parse tree incrementally from the separate word-tags in a sentence, the network should output the set of parent-child relationships which define each word-tag’s position in the parse tree. Figure 5.1 illustrates the sentence ‘Mary saw the game was bad’, which is represented as a sequence of word-tags as ‘NP VVD AT NN VBD JJ’. The sentence structure (S) contains separate constituents for the subject noun (N) and object clause (F), which contains a further noun phrase (N); the parent-child relationships between the nodes for each of these constituents is shown by solid lines. The *parent* node of each word-tag in the sentence is the unique node which that word-tag is directly connected to. For each node in the parse tree, its *children* are those nodes which connect to it. For example, the children of the ‘S’ node in Figure 5.1 are the ‘N’ node, the word-tag ‘VVD’, and the ‘F’ node. When parsing, the network is presented with a word-tag on its input, and

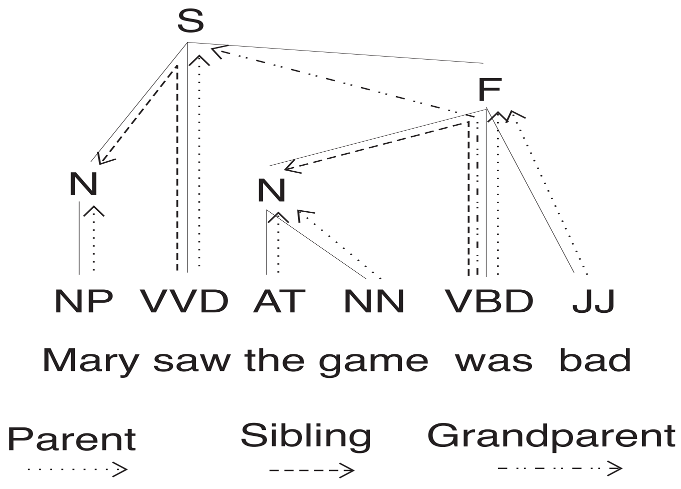


Figure 5.1: A sample parse tree. The solid lines indicate the parse tree itself, the dotted and dashed lines the relationships between the words and nodes.

the desired output is a representation for the parent-child relationships relevant to that word-tag. In the experiments with the SSN, each word-tag is introduced to the network on a new, previously unused phase, and so introduces a new entity to the network. These entities are used in the output format to represent the individual nodes of the parse tree.

The parent-child relationships are output by the parser as described in Section 4.1.5 of the previous chapter. The parser is provided with three kinds of relationship: ‘parent’, ‘sibling’ and ‘grandparent’. These relationships are indicated by setting the appropriate output unit in the phase of the node to which they refer. The ‘parent’ output is used to indicate the parent node of the current input word-tag. For each word-tag, there are two cases to consider. Firstly, its parent is an existing node in the tree. For example, the word-tags “NN” and “JJ” in Figure 5.1. Secondly, the parent is a new node and is therefore introduced by the current word-tag. In which case, the parent of the word-tag is the node introduced by the current word-tag. For example, the word-tags “NP”, “VVD”, “AT” and “VBD” in Figure 5.1. When a new node has been introduced into the parse tree, its relationship to other nodes must also be indicated. There are again two cases. The first occurs when, as with the first word-tag in the sentence, the parent of the new node (‘N’) is a node not yet introduced, the ‘S’ in this case. This relationship can only be given when the word-tag which introduces the ‘S’ node appears on the input. Once this word-tag, “VVD”, is input, then its relationship to the ‘N’ node will be output. This relationship is a ‘sibling’ relationship, and is shown as a dashed line in the figure. The second case occurs when the parent of the new node already exists in the sentence. The relationship of the current word-tag to that existing node is a ‘grandparent’ relationship, and is indicated at the same time as the new node is introduced. An example of a grandparent relationship is the dotted-and-dashed line in Figure 5.1 between the “VBD” word-tag and the ‘S’ node; the word-tag “VBD” has introduced a new ‘F’ node, and the parent of this node is the earlier ‘S’ node.

The dotted and dashed links in Figure 5.1 from the word-tags to nodes in the parse tree indicate which relationships are output by the network when that word-tag is input. These relationships are identified by making the relevant output unit active in the time period in which the word-tag is input. The relevant output unit is active in phase with the node to which the current word-tag is related. For example, the sibling relationship between the word-tag “VVD” and the ‘N’ node is indicated in the time period when “VVD” is input by making the S (sibling) output active in phase with the ‘N’ node. Table 5.3 lists the ‘parent’ (P), ‘sibling’ (S) and ‘grandparent’ (G) relationships relevant to each word-tag in the sentence from Figure 5.1; the table shows the time period and phase for the

Time Period	Pulsing Inputs Phase						Non-pulsing Inputs	Outputs Phase					
	1	2	3	4	5	6		1	2	3	4	5	6
1	NP						NP	P					
2		VVD					VVD	S	P				
3			AT				AT			P			
4				NN			NN			P			
5					VBD		VBD		G	S		P	
6						JJ	JJ					P	

Table 5.3: GPS outputs for the sample structure shown in Figure 5.1.

input of each word-tag, and similarly the relationships output for that word-tag.

These three outputs, the GPS representation, form the target output for the experiments in parsing natural language. The representation can be extended by adding output units for the node labels; the label for the node is indicated in the time and phase that the node is created. This representation is important as it provides a representation of arbitrary parse trees, not just for SSNs, but for any TSVB network, and provides a model for recursive structured representations when applying SSNs to other domains.

5.1.4 Conversion of the SUSANNE corpus

This section deals with the process by which the SUSANNE corpus is converted into the word-tag–GPS representation used in the experiments. The process may be summarised in the following steps:

- extract the word-tag and parse tree information (fields 3 and 6) from the raw corpus,
- remove certain categories of 'word' and parse-tree information,
- collapse some nodes of the parse tree, so that the one constituent per word-tag requirement is met,
- identify all constituents in the parse tree and their head words, i.e. work out the Parent relationships,
- identify each node's right children where known, i.e. the Grandparent relationships,
- identify each node's left children where known, i.e. the Sibling relationships,
- finally, convert into a binary representation.

For convenience, these steps are divided into a three-stage procedure, and described separately in detail. In summary, the first stage takes the raw SUSANNE data and outputs a sequence of the 3 letter (Lancaster set) word-tags with an indication of their immediate parent and grandparent nodes where possible. This stage is automated, and is achieved in a single forward pass of the data. This stage leaves out the sibling relationships, as these require consideration of earlier parts of a sentence. Instead, the program identifies these problem areas with a '?' sign in the sibling (S) output.

The second stage replaces each '?' sign with the appropriate phase of the left-child, identifying which previous constituent in the sentence the word attaches to. This step also requires a change

Word-tag	G	P	S	Phase	Parsed nodes
<>	.	0	.	0	S0
AT	0	1	.	1	S0 N1
NP	1	2	.	2	S0 N1 N2
NNL	.	2	.	3	S0 N1 N2
JJ	.	1	.	4	S0 N1
NN	.	1	.	5	S0 N1
VVD	.	0	.	6	S0
NPD	0	7	.	7	S0 N7
AT	8	8	?	8	S0 F8 N8
NN	.	8	.	9	S0 F8 N8
IO	8	10	.	10	S0 F8 N8 P10
NP	11	11	?	11	S0 F8 N8 P10 N11 G11 N11
GG	.	11	.	12	S0 F8 N8 P10 N11 G11
JJ	.	11	.	13	S0 F8 N8 P10 N11
JJ	.	11	.	14	S0 F8 N8 P10 N11
NN	.	11	.	15	S0 F8 N8 P10 N11
VVD	.	8	.	16	S0 F8
YIL	.	8	.	17	S0 F8
AT	8	18	.	18	S0 F8 N18
NN	.	18	.	19	S0 F8 N18
YIR	.	18	.	20	S0 F8 N18
CST	18	21	.	21	S0 F8 N18 F21
DD	21	22	.	22	S0 F8 N18 F21 N22
NN	.	22	.	23	S0 F8 N18 F21 N22
VVD	.	21	.	24	S0 F8 N18 F21
NNL	21	25	.	25	S0 F8 N18 F21 N25
.					

Table 5.4: First stage in conversion process.

of phase number of some of the intermediate nodes, and, in some cases, a merging of nodes. This process was performed manually to ensure that all relevant cases were handled correctly.

The third and final stage is again performed automatically by a program, and converts the word-tags, constituent labels and numbers for the various output relationships into the binary format required for training the SSN.

Stage one – extracting initial information

The SUSANNE database incorporates a wealth of detail which is not required for experiments in learning to output syntactic parse trees. Table 5.4 shows the result of the first stage in the conversion of the SUSANNE sample sentence from Table 5.1 into the desired format. This stage is performed by the computer, and explained in this section.

The first column gives the word-tag for each line in the sentence. The word-tag information is the Lancaster-set part of the SUSANNE word-tag, i.e. the first two or three capital letters of the SUSANNE word-tag. Note that not every line in the SUSANNE database is converted. Some lines refer to information about headings, and these are deleted. Also, the SUSANNE scheme can

Word-tag	G	P	S	Phase	Parsed nodes
<>	.	.	.	0	# remove attachment to <>
AT	.	1	.	1	S6 N1
NP	1	2	.	2	S6 N1 N2
NNL	.	2	.	3	S6 N1 N2
JJ	.	1	.	4	S6 N1
NN	.	1	.	5	S6 N1
VVD	.	6	1	6	S6 # make VVD head of S0
NPD	6	7	.	7	S6 N7
AT	.	8	.	8	S6 F16 N8 # introduce F8 on 16
NN	.	8	.	9	S6 F16 N8
IO	8	10	.	10	S6 F16 N8 P10
NP	.	11	.	11	S6 F16 N8 P10 N13 G12 N11
					# introduce first N11 on 13, G11 on 12
GG	.	12	11	12	S6 F16 N8 P10 N13 G12
JJ	10	13	12	13	S6 F16 N8 P10 N13
JJ	.	13	.	14	S6 F16 N8 P10 N13
NN	.	13	.	15	S6 F16 N8 P10 N13
VVD	6	16	8	16	S6 F16
YIL	.	16	.	17	S6 F16
AT	16	18	.	18	S6 F16 N18
NN	.	18	.	19	S6 F16 N18
YIR	.	18	.	20	S6 F16 N18
CST	18	21	.	21	S6 F16 N18 F21
DD	21	22	.	22	S6 F16 N18 F21 N22
NN	.	22	.	23	S6 F16 N18 F21 N22
VVD	.	21	.	24	S6 F16 N18 F21
NNL	21	25	.	25	S6 F16 N18 F21 N25
.					

Table 5.5: Second stage in conversion process.

produce a ghost marker in certain instances. This marker is used to convey semantic information in the parse tree about long-distance dependencies, and so is also ignored. Punctuation is retained within the sentence, although the various end-of-sentence markers (exclamation or question marks, ellipsis, etc) are all converted into full-stops ('.'). The start of the sentence is indicated with a <> symbol. The deletions made do not affect the constituent boundaries, and so leave the syntactic parse information unchanged.

The next three columns give the grandparent, parent and sibling relationships for each word-tag. The numbers in each column refer to the phase of the node to which the word-tag is related. The column 'Phase' gives the phase number for any node introduced by that word-tag. The GPS outputs are defined as described above, and indicate the structure of the evolving parse tree in terms of the relationships between the current word-tag and the rest of the parse tree. For example, the start symbol '<>' introduces the 'S0' node in phase 0, hence the P output is active in phase 0, and no activity appears on the G and S outputs. The conversion program recognises from the parse tree information held in the SUSANNE database when nodes are introduced. If a new node is introduced with the current word-tag, then the P output must be active in phase with the current word-tag. The G output is active if a new node is introduced, and its parent is a previous word-tag. For example, the second word-tag 'NP', introduces the new node 'N2' and is attached to the node 'N1' from phase 1, hence activity appears on the P output in phase 2 and the G output in phase 1. The conversion program considers each line of the SUSANNE corpus in succession, and thus cannot handle sibling relationships. Where such a relationship is indicated, i.e. the grandparent of a new node cannot be identified, the program marks the S output with a '?' sign, and continues. Such a case can be seen in phase 8: both a new noun-phrase and a new relative clause begin at this point. The head word for the latter will be its verb, which has yet to appear, and this verb will be the parent of the noun-phrase begun in phase 8. The problem is identified because two new nodes have been introduced, each with the new phase number.

The final column gives the parse tree information. This information is a list of the nodes relating to each word-tag in the sentence. The nodes are described with a letter and a number: the letter is taken from the SUSANNE database, and is the first letter in the Formtag, the number is that of the phase in which that node was introduced. The right-most node indicates which constituent is the word-tag's immediate parent, and this node's letter defines the constituent label output for this word-tag. For example, the first word-tag, 'AT' has as its parent a node labelled 'N1', indicating that it is a noun-phrase, and was introduced in phase 1 (along with the 'AT' word-tag). This 'N1' has the 'S' node, the root node for the sentence, as its parent, and this was introduced in phase 0, with the start symbol.

Some of the parse tree constituent markers have been removed. These are principally the meta-sentence markers, such as '[O.O]', which are irrelevant to a sentence level parser. Also, nodes indicating verb phrases have been combined with nodes indicating sentence or relative clauses. This change is the first motivated by the need to collapse some of the nodes within the tree in order to obtain a parse tree which only contains 'one node per word-tag'.

To summarise: this stage has performed much of the hard work in extracting the information from the SUSANNE database and putting it into the word-tag–GPS representation to be used in the experiments. Remaining are the sibling relationships, which are added in the second stage.

Stage two – adding the sibling information

The second stage in the conversion procedure completes the process of the previous stage, by considering each of the '?' marks in the S output. Consider the first '?' mark in Table 5.4. This occurs with the word-tag 'AT' in phase 8 and, according to the SUSANNE database, two new constituents begin at this point. The first is the noun-phrase of which the 'AT' forms a part, and the second is the relative clause which this noun-phrase begins. Because both the noun-phrase and relative clause are introduced at this point, the first stage has placed 'S0 F8 N8' in the parsed nodes column. However, when constructing the GPS outputs, the first stage has run into a problem: looking at the top-most node, 'N8', it determines that the current word-tag's Parent is the 'N8' node, and so the P output is made active in phase 8. Because this is the same as the current node, the word-tag has therefore introduced a new node, which means the parent of this new node must also be given. This node is the node 'F8'. Unfortunately, because this is also a new node, introduced in phase 8, it is not suitable as a grandparent of the current word-tag. The first stage has therefore given phase 8 for the G output, and marked the S output with a '?' mark.

The question to be resolved here is, what to do about the 'F8' node? The solution adopted here is to make the phase for the 'F' node the phase in which its head word is introduced, because 'F8' is the node describing the relative clause constituent. The head word for a relative clause is its main verb, which only appears as the 16th word-tag, 'VVD'. (Visually, it is evident that the node 'F8' is the immediate parent only for this word-tag; this is only so because the verb-phrase descriptors were removed in the first conversion stage.)

Therefore, in this second stage, the entire 'F8' node is replaced by an 'F16' node, i.e. it is introduced by the word-tag 'VVD'. The noun-phrase introduced by the word-tag 'AT' in phase 8 therefore has 'F16' as its parent, which is indicated as a sibling relationship between the word-tag 'VVD' and the noun-phrase introduced by the word-tag 'AT' in phase 8. This change is carried out in Table 5.5, where the comment '# introduce F8 on 16' refers to the changes made to the GPS outputs in Table 5.4.

Further, because the root node for the sentence is the main verb, a similar change is made to the 'S0' node: the main verb in phase 6 introduces the node 'S'. This makes for a similarity in structure between the sentence and relative clauses.

There are several other places where such changes arise. The most frequent of these are: deeply embedded noun, prepositional, determiner and genitive phrases.

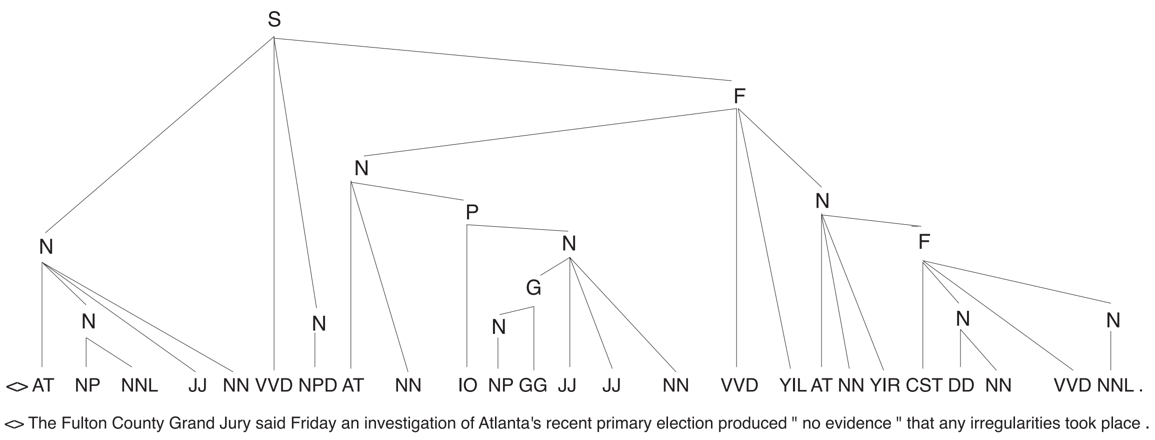
Before this stage can be made an automatic one, implemented in a computer program, all these possible changes must be identified and the appropriate responses determined. However, the basic technique for making the changes is a fairly straightforward one. Therefore, for the purposes of this thesis, it was found most appropriate to rely on a scanning by eye of the whole database, manually replacing the '?' marks and clause heads as detailed above. This process also uncovered other features of the SUSANNE scheme which detracted from the homogeneity of a sentence based parser, for example, the presence of long lists (100 words or more), which could be (and were) converted into short sentences.

Finally, from this second stage, the data can be presented to the networks, after converting the word-tags and GPS relationships into a sequence of binary numbers.

Stage three – converting into binary

The third stage is a simple procedure, converting the 'human-readable' representation generated by the first two stages into a set of binary values. The word-tags are dealt with as three separate inputs, one input per character in the tag. The three characters are then input to the network as three separate 1-in-n basis vectors.

For the outputs, values must be given for each output unit in every phase of that word-tag's time period. Thus, as each word-tag introduces an additional phase to the network, the amount of target output required for each input word-tag grows quadratically. For the GPS output units, the phase numbers given at the end of stage two indicate which phase of the current time period that output unit is active in. For all other phases in the current time period that output unit will be inactive, i.e. have a target activation of 0.



S
F
N
P
N
N
G
F
N
N
N
N
N
N
<> AT NP NNL JJ NN VVD NPD AT NN IO NP GG JJ JJ NN VVD YIL AT NN YIR CST DD NN VVD NNL .
<> The Fulton County Grand Jury said Friday an investigation of Atlanta's recent primary election produced " no evidence " that any irregularities took place .

Figure 5.2: The target parse tree for the example sentence from the SUSANNE corpus used in training the SSN parser.

The output label is also indicated as a 1-in-n vector, with 15 possible output labels, as shown in Table 5.2. The label information is only relevant to the currently introduced phase, and so the activation for the label output units are given once, indicating the target outputs for the label in the currently introduced phase.

Figure 5.2 illustrates the parse tree formed from the example sentence.

It remains to be considered how the GPS relationships output by a SSN, which define an output parse tree for the sentence, will be compared to a target parse tree.

5.1.5 Evaluation of the parser's output

The performance of the SSN with respect to its training data is computed on each pass of the backpropagation training algorithm as the difference between the computed activation and the target activation on each of the output units. However, this performance is only useful for training the weights within the network. For evaluating the output structure or comparing the performance of the SSN with other parsing techniques, such as the PCFG, a performance measure specific to the output parse tree is required. In Section 3.3, the *precision* and *recall* performance of a parser with respect to its target corpus were defined; these are the standard quantities used for comparing target and output parse trees in statistical parsing. These quantities are defined by comparing the constituents output by the parser with the constituents in the corpus.

In order to interpret the SSN's output as a parse tree, the first step is to convert the sequence of real-valued numbers on the GPS output units computed in response to each word-tag into a set of well-defined GPS relationships. The second step is to turn these GPS relationships into a set of constituents for the current sentence. Constituents are formed from the GPS outputs merely by thresholding the activation on each output unit and using this information, as described for constructing the target parse, to specify the presence of new nodes and the relationships between the nodes in the parse tree.

This process can be illustrated by considering an example sentence. Table 5.6 gives the input, target and actual output generated by a Type B network on an example sentence; the phases listed under 'Actual output' are those in which the designated output unit had an activation exceeding 0.6. This value is used as a threshold simply to ascertain which units are producing a significant amount of output; these units will then be compared to determine the relationships output by the network. It is necessary to threshold before making a comparison because units may output small activation values, and it is not desirable for such a unit to be selected as providing an informative output. From

		Target Output				Actual Output			
Time Period	Wordtag	G	P	S	Label	G	P	S	Label
0	<Start>								
1	PPI		1		N		1		N
2	VBM		2	1	S		0,2	1	S
3	JJ	2	3		J		2,3		N
4	TO	3	4		T		2,4		T
5	VV		4		T		2,4		T
6	APP	4	6		N	4	6		N
7	JJ		6		N		6		N
8	NN		6		N		6		N
9	II	4	9		P		9		P
10	PPH		9		P		9,10		N
11	YIR		2		V				F

Table 5.6: GPS target and actual outputs for a sample sentence

these thresholded output activations, information must be extracted about the nodes introduced by each word-tag and the relation of each new node to the rest of the tree. This is extracted in the following manner.

For each word-tag, its parent non-terminal node is identified by the parent output unit active in its time period; as activity may appear in several phases, the one with the greatest activity is chosen, or else the most recent one. The parent of that non-terminal node is then identified as the winner of the following competition: first, the phase in which the grandparent output has the maximum activation in the current time period is identified; then the time period in which the sibling output has the maximum activation in the current new phase is identified; then, of these two numbers, the one with the highest activation or else the most recent is selected.

Applying this procedure to the outputs given in Table 5.6 creates the set of relationships shown in Table 5.7. These relationships are then converted into a set of constituents. This is achieved in the following manner. First, the active nodes are identified, i.e. those in which the word-tag's parent is its own phase, so indicating that a new node has been introduced to the parse tree. These active nodes are the heads of constituents, and the constituent is initialised with the word-tag which introduced it. Second, the word-tags whose parent is one of the active nodes are added to their parent constituents. Third, the grandparent relations are considered to determine which other constituents each constituent is a member of; when a constituent is a member of another constituent, then every word-tag in that constituent is a member of the other. This process leads to the sets of constituents illustrated in Table 5.8.

Evaluation proceeds by counting the number of constituents output by the parser, the number of constituents in the target parse and the number of constituents which are identical in the target and output sets. In this example, 6 constituents are in the output and target sets, and 3 are identical in both. Thus the precision, the percentage of the output constituents which are correct, and recall, the percentage of the correct constituents which are output, for this sentence are both 50%.

There are also more sophisticated approaches to extracting and comparing the constituents. For example, punctuation is treated like extra word-tags, but it is unrealistic to penalise a parse if a comma is located in one constituent instead of an adjacent one. One approach ignores the commas when computing precision/recall figures. Also, one word-tag constituents may also arise. This extra information may, in many cases, be irrelevant. e.g. `[[CC ] VV ... ]` and `[CC VV`

		Target Output			Actual Output		
Time Period	Wordtag	Parent	Grandparent	Label	Parent	Grandparent	Label
0	<Start>						
1	PPI	1	2	N	1	2	N
2	VBM	2		S	2		S
3	JJ	3	2	J	3		N
4	TO	4	3	T	4		T
5	VV	4		T	4		T
6	APP	6	4	N	6	4	N
7	JJ	6		N	6		N
8	NN	6		N	6		N
9	II	9	4	P	9		P
10	PPH	9		P	9		N
11	YIR	2		V			F

Table 5.7: Parent-child relationships for the non-terminal nodes in a sample sentence

. . .] . No such techniques are used in the experiments reported in this thesis, which therefore uses a 'worst case' computation of the precision/recall figures.

5.2 Experimental Results

This section describes results from a series of experiments performed with Simple Synchrony Networks (SSNs). The SSNs are trained to parse samples of real natural language taken from the corpus described in the previous section. The experiments are performed using the STM-SSN model, i.e. the SSNs used in the previous chapter have been augmented with the Short-Term Memory queue. This has the effect of dramatically reducing training times, by reducing the number of phases the network must compute over. Various sizes of STM queue are tested, to determine its effect on learning. Throughout, results are quoted both in terms of the precise number of relationships correctly output by the parser, and in terms of the standard precision/recall measures for evaluating the degree of fit between an output parse tree and its target.

5.2.1 Experiments with the STM-SSN network

The experiments all use the same set of data for selection of training, cross-validation and test sets, taken from one of the SUSANNE genres. Genre A for press reportage was selected and sentences allocated into the three data sets in the ratio 4:1:1. Sentences of less than 15 word-tags were selected from the training set, forming a set of 265 sentences containing 2834 word-tags, an average sentence length of 10.7. Similarly, a cross-validation set was selected, containing 38 sentences of 418 word-tags, average sentence length of 11.0, and a test set with 34 sentences, containing 346 words, with an average sentence length of 10.2.

The three SSN types A, B and C were all tested. Twelve networks were trained from each type, consisting of four sizes of network (between 20 and 100 units in each layer), each size was tested with three different STM lengths (3, 6 and 10). Each network was trained on the training set for 100 epochs, using a constant learning rate η of 0.05. The Appendix contains results from testing each network on the cross validation set at intervals of 20 epochs throughout training.

```
Target constituents
Word  0  1  2  3  4  5  6  7  8  9 10 11  Label
  0   0  0  0  0  0  0  0  0  0  0  0  0
  1   0  1  0  0  0  0  0  0  0  0  0  0   N
  2   0  1  1  1  1  1  1  1  1  1  1  1   S
  3   0  0  0  1  1  1  1  1  1  1  1  0   J
  4   0  0  0  0  1  1  1  1  1  1  1  0   T
  5   0  0  0  0  0  0  0  0  0  0  0  0
  6   0  0  0  0  0  0  1  1  1  0  0  0   N
  7   0  0  0  0  0  0  0  0  0  0  0  0
  8   0  0  0  0  0  0  0  0  0  0  0  0
  9   0  0  0  0  0  0  0  0  0  1  1  0   P
 10   0  0  0  0  0  0  0  0  0  0  0  0
 11   0  0  0  0  0  0  0  0  0  0  0  0

Output constituents
Word  0  1  2  3  4  5  6  7  8  9 10 11  Label
  0   0  0  0  0  0  0  0  0  0  0  0  0
  1   0  1  0  0  0  0  0  0  0  0  0  0   N
  2   0  1  1  0  0  0  0  0  0  0  0  0   S
  3   0  0  0  1  0  0  0  0  0  0  0  0   N
  4   0  0  0  0  1  1  1  1  1  0  0  0   T
  5   0  0  0  0  0  0  0  0  0  0  0  0
  6   0  0  0  0  0  0  1  1  1  0  0  0   N
  7   0  0  0  0  0  0  0  0  0  0  0  0
  8   0  0  0  0  0  0  0  0  0  0  0  0
  9   0  0  0  0  0  0  0  0  0  1  1  0   P
 10   0  0  0  0  0  0  0  0  0  0  0  0
 11   0  0  0  0  0  0  0  0  0  0  0  0

Correct constituents: 3
Number in target    : 6 (Recall:    50.0%)
Number in output    : 6 (Precision: 50.0%)
```

Table 5.8: Target and output constituents for a sample sentence; a 1 in column x of row y indicates that word-tag x is a member of the constituent introduced by word-tag y.

Test	Sentences	Precision	Recall	G	P	S	Label
Type A : STM of 10, 100 units							
Train	1/265 (0%)	34.7	31.3	35.6	65.9	20.8	89.3
Cross	0/38 (0%)	31.8	29.7	30.5	65.2	18.7	84.7
Type B : STM of 3, 80 units in each layer							
Train	63/265 (24%)	69.9	68.7	74.1	94.3	73.0	97.5
Cross	10/38 (26%)	71.5	71.2	74.8	95.6	70.7	96.4
Test	9/34 (25%)	71.2	70.8	82.0	94.6	64.6	98.3
Type B : STM of 6, 80 units in each layer							
Train	62/265 (24%)	66.0	64.4	75.2	92.3	84.6	97.5
Cross	12/38 (32%)	65.3	64.2	82.4	92.6	85.3	96.2
Test	8/34 (25%)	67.4	65.2	83.0	93.3	83.1	98.3
Type C : STM of 3, 80 units in each layer							
Train	64/265 (24%)	72.6	71.7	70.8	95.7	72.1	98.4
Cross	9/38 (24%)	74.4	73.8	71.8	97.3	70.7	97.8
Test	13/34 (38%)	80.3	80.3	85.0	96.0	66.1	99.1
Type C : STM of 6, 80 units in each layer							
Train	56/265 (15%)	65.5	63.7	68.1	92.8	82.3	97.1
Cross	8/38 (21%)	63.6	61.1	68.7	91.5	81.3	95.5
Test	10/34 (29%)	71.4	70.2	76.0	93.9	84.7	98.3

Table 5.9: Comparison of network types in learning to parse

The best example of each network type was chosen for comparison. Table 5.9 gives figures for five networks. For each network, the performance on the three datasets (training, cross-validation and test) are given under three categories: the number of correct sentences, a measure of the number of correct constituents (precision and recall) and the percentage of correct responses on each output unit.

Considering the figures in the table, the type A networks are not particularly successful. Results for the best performing type A network are given. The type B and C networks were much more successful. For each type, the results from two networks are given: the first having the best precision/recall measure, and the second having better results on the individual outputs (i.e. G, P, S and label). Both the type B and C networks produce similar ranges of performance: around 25% of sentences are correct and between 70-80% is scored in average precision/recall by the better networks. In particular, the percentage correct for the constituent labels and the P output exceed 90%. Also notable is that the percentage results of the networks are similar across the three datasets, indicating that the network has learnt a robust mapping from input sentences to output parse trees. This level of generalisation (around 80% average precision/recall) is similar to that achieved by PCFG parsers [48], although for a fair comparison identical experiments must be performed with each algorithm: this point is returned to in the conclusion.

5.2.2 Analysis of results

The basic experimental results above have provided both detailed values of the performance of the network with specific output relationships as well as their combined performance in terms of the constituent-level measures of precision and recall. Here, these results are broken down and compared to see the progress of learning of the networks over time, a comparison of the effect of the STM queue and a table of the actual dependencies present in the data itself.

Effects of STM length

The effects of STM length can be seen by plotting the performance of one type of network with varying sizes of STM. This is done in Figure 5.3, in which the performance of a type C network with 80 units in every hidden layer is shown for the three sizes of STM, i.e. 3 (from Table A.7), 6 (from Table A.8) and 10 (from Table A.9). The separate graphs show the constituent-level performance of the network, in terms of average precision/recall, and the performance of the separate output units, grandparent, parent, sibling and constituent label (this latter, though a group of units, is treated as a single output). The graphs demonstrate that, at the constituent-level, the shorter STM lengths perform better. However, the longer STM lengths can achieve greater accuracy, in particular with respect to the sibling output. This is to be expected, as the longer lengths have more information and also a greater likelihood of containing the phase referred to by the specific output.

Dependency lengths in data set

One of the concerns of connectionist language learning has been the length of dependency which the SRN can learn [21]. It is of interest to provide an analysis of the data set used in these experiments to see what lengths of dependency occur, particularly with regard to the impact the finite length of STM might have on the network's ability to learn.

Table 5.10 contains an analysis of the lengths of each dependency contained in a sample of the training corpus of sentences of length less than 30 words. This corpus contains 13,472 word-tags in 716 sentences, with an average length of 18.83. The length of a dependency is the number of words between the current word and that indicated by each output. The table lists separately the lengths for each of the output units, with the final two columns providing a total number and percentage for that dependency length across the whole corpus. The surprising result of this analysis is that most of the dependencies (almost 70%) relate to the current word or its predecessor. There is a sharp tailing off of frequency as longer dependencies are considered.

Of course, with the STM queue, the network can only process a limited number of words at any one time, and so the length of dependency which the network can handle is altered. However, because the STM holds phases for the relevant constituents, the length of any dependency is not restricted to the length of the STM queue. With a STM, the length of dependency will be the number of places down the queue which each phase has progressed before being required. So, in Table 5.11, a similar analysis to that above is performed, but this time, instead of counting the length as across phases, the length of each dependency is counted as the position which that phase occupies in the STM. Thus, if a phase is in the third position of the STM when referred to, then the length of the dependency will be given as three. This table shows that the dependency lengths possess a similar range to that in the previous one, although there is a greater concentration in the shortest lengths, as might be expected. The limited number of longer dependencies still extend to the same length, and this is due to isolated words or punctuation symbols which are referred to only the once during each sentence.

5.2.3 Conclusion of experiments

From the technical point of view, the impact of the STM has had considerable effect, reducing training times by at least an order of magnitude. As discussed above, the actual lengths of dependencies encountered by the network are not changed much by the addition of a STM. The experiments show that longer STMs achieve better performance on the specific outputs of the network, however the shorter STM still yields the best level of constituent accuracy. This is because the shorter STM

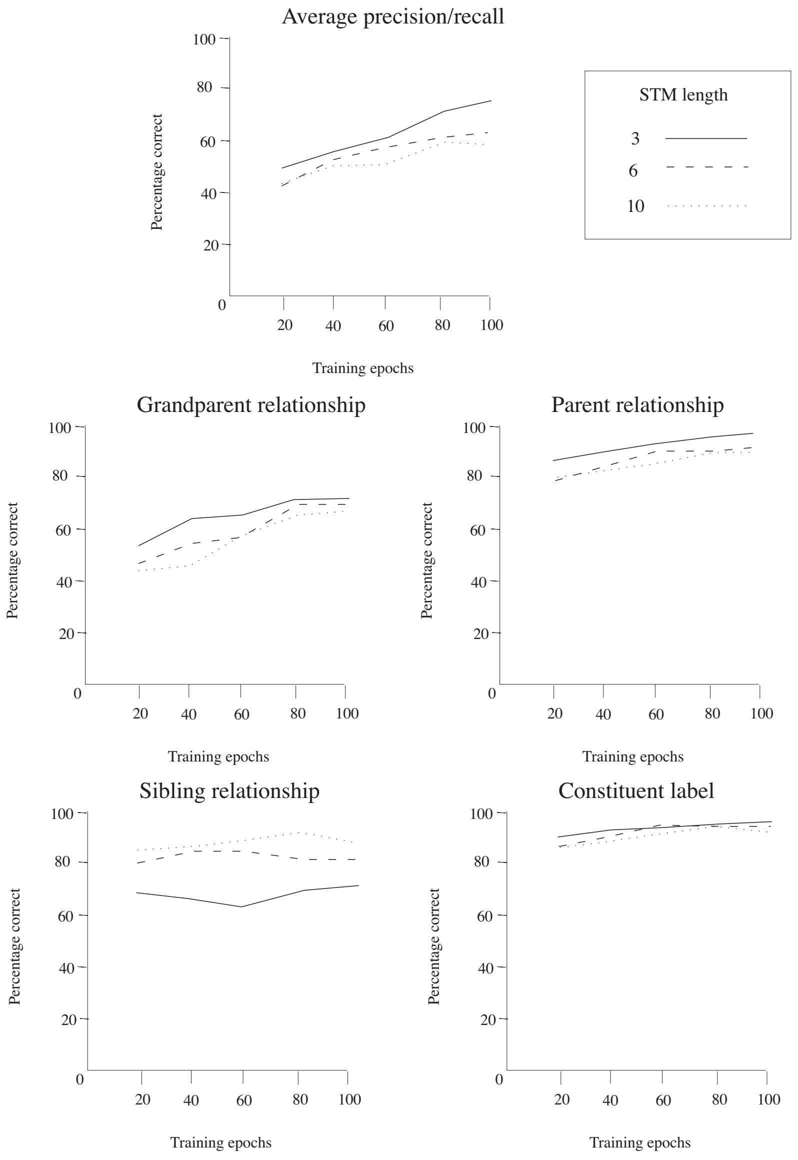


Figure 5.3: Comparison of the effects of STM length on a type C network with 80 units in every hidden layer.

Length	G	P	S	total	%
0	0	7354	0	7354	(38.0%)
1	2268	3134	763	6165	(31.8%)
2	1018	1037	502	2557	(13.2%)
3	568	350	211	1129	(5.8%)
4	351	173	106	630	(3.3%)
5	225	101	78	404	(2.1%)
6	157	60	54	271	(1.4%)
7	115	45	40	200	(1.0%)
8	79	30	34	143	(0.7%)
9	52	19	23	94	(0.5%)
10	36	15	24	75	(0.4%)
11	36	17	16	69	(0.4%)
12	27	17	19	63	(0.3%)
13	21	10	19	50	(0.3%)
14	16	6	16	38	(0.2%)
15	5	8	10	23	(0.1%)
16	6	8	11	25	(0.1%)
17	6	7	10	23	(0.1%)
18	4	4	6	14	(0.1%)
19	2	6	5	13	(0.1%)
20	1	3	8	12	(0.1%)
21	4	1	2	7	(0.0%)
22	0	1	4	5	(0.0%)
23	0	4	3	7	(0.0%)
24	0	1	3	4	(0.0%)
25	0	0	1	1	(0.0%)
Means	2.7	0.8	3.3	1.6	

Table 5.10: Dependencies by type and length in the training corpus with less than 30 words per sentence.

Dependency Length	G	P	S	total	%
0	0	7354	0	7354	(38.0%)
1	2614	3011	1128	6753	(34.9%)
2	1394	1513	310	3217	(16.6%)
3	397	252	150	799	(4.1%)
4	284	127	78	489	(2.5%)
5	135	42	68	245	(1.3%)
6	62	26	48	136	(0.7%)
7	45	16	38	99	(0.5%)
8	15	10	30	55	(0.3%)
9	16	12	21	49	(0.3%)
10	6	9	17	32	(0.2%)
11	10	9	11	30	(0.2%)
12	6	9	12	27	(0.1%)
13	5	8	8	21	(0.1%)
14	1	4	10	15	(0.1%)
15	3	2	7	12	(0.0%)
16	1	4	4	9	(0.0%)
17	0	1	8	9	(0.0%)
18	2	4	4	10	(0.0%)
19	0	1	2	3	(0.0%)
20	1	0	4	5	(0.0%)
21	0	1	3	4	(0.0%)
22	0	0	2	2	(0.0%)
23	0	0	2	2	(0.0%)
24	0	0	2	2	(0.0%)
25	0	0	1	1	(0.0%)
Means	2.0	0.7	2.7	1.2	

Table 5.11: Dependencies by type and length across STM in a training corpus with less than 30 words per sentence.

must contain fewer phases for determining the relationships between word-tags. This means the inaccurate phases are less likely to feature in the competition for determining which relationships to use in building up the output parse tree, and so the resultant tree is more likely to be correct.

However, more importantly, the experiments demonstrate how a connectionist network can successfully learn to generate parse trees for sentences drawn from a corpus of naturally occurring text. This is a standard task in computational language learning using statistical methods. Because the same performance measures (precision/recall) can be applied to the output of the SSN as with a typical statistical method, such as the Probabilistic Context-Free Grammar (PCFG), direct comparisons can be made between the two approaches. For instance, the simple PCFG can achieve around 72% average precision/recall [48] on parsing from sequences of word-tags, and the SSN in the above experiments achieved 80% average precision/recall when trained and tested on sentences with fewer than 15 words. However, this is not a fair comparison, as the corpora representations, sizes and contents are dissimilar.

In an extension to the work here, Henderson [39] has presented a slight variant of the basic SSN model and compared its performance directly with that of PCFGs on identical corpora. In those results, the PCFG, due to the restricted size of the training set, was only able to parse half the test sentences, with a precision/recall figure of 54%/29%. In comparison, the SSN was able to parse all the sentences and yielded a performance of 65%/65%. Even when counting only the parsed sentences, the PCFG only had a performance of 54%/58%, compared to the SSN's performance of 68%/67% on that subset. The variations introduced by Henderson [39] to the SSN mostly affect the input layer. In this thesis, the pulsing inputs to the SSN receive input only for the newly introduced phases, requiring the network to remember the input from previous time periods. In [39], the pulsing input from the previous period is carried forward in its particular phase. An additional pulsing input unit is then used to distinguish the newly introduced phase from the others. Because of this change in input representation, the results in [39] have been achieved with a type A SSN (and no STM was used). These results indicate that the SSN is generalising from its training data in a superior fashion to the PCFG. [39] attributes this to the combination of the SSNs' ability to generalise across constituents and the ability of connectionist networks to learn their own internal representations.

(Preliminary results from the experiments in this section appeared in [40, 55], and a full description appears in [56].)

5.3 Conclusions

This chapter has described a set of experiments training the Simple Synchrony Network (SSN) to parse samples of natural language taken from a corpus of naturally occurring text. The importance of these experiments for connectionist language learning is that the SSN, tested in the previous chapter on toy grammars, has been shown to work additionally on sentences drawn from naturally occuring text. The SSN has achieved a consistently good performance on this task, both in terms of its specific outputs and with the constituent-level evaluation familiar from the statistical community. The simplicity of the SSN means that these results are explicable entirely by the introduction of a representation which generalises according to systematicity [37] into standard connectionist networks. That is, incorporating the appropriate generic generalisation principle into connectionist networks has produced a true AI based approach to NLP. The approach handles real natural language, outputs the structured parse tree representation, and also generalises well from its training data. This makes the SSN the first serious connectionist alternative to standard statistical approaches to natural language parsing: a claim which has been further supported by the direct comparison in [39].

Chapter 6

Evaluation and Conclusions

This thesis has developed a new class of connectionist architecture, the Simple Synchrony Network (SSN), and presented experimental results achieved by the SSN in learning to parse toy grammars and samples of real natural language. The importance of the SSN as a new machine learning algorithm is discussed in this chapter by relating the theoretical and empirical results of the previous chapters to earlier approaches in the literature. Finally, this chapter describes some areas for future work and concludes.

6.1 Basic Results

This thesis has developed a new class of connectionist architecture, the Simple Synchrony Network (SSN). The SSN combines two specific extensions of standard feed-forward connectionist networks into a single architecture: Simple Recurrent Networks (SRNs), which use context units to preserve activations from previous time periods, and so learn about patterns across time; and Temporal Synchrony Variable Binding (TSVB), which uses a division of each time period into independent phases to represent multiple entities.

This combination of SRNs and TSVB, implemented as shown in Sections 4.1.1 and 4.1.2, enables a natural definition to be made of a range of trainable TSVB networks. From results achieved in exploring this range of networks, this thesis has developed two ideas for achieving the most effective learning from TSVB networks. The first idea is a restriction to the range of possible networks. The variety of networks arises because TSVB networks contain more than one type of unit, pulsing and non-pulsing. However, there are interactions between these units which affect the efficiency of learning. In particular, it was shown that links between pulsing and non-pulsing units lead to problems in generalising across increasing numbers of entities; the SSN is defined as a TSVB network without such links.

The second idea addresses a computational inefficiency with this definition of SSNs, which is the assumption that all phases are retained within the network over the processing time of the current sentence. This is theoretically unnecessary, as shown by limitations on human short-term memory [16, 70], and computationally undesirable, due to the protracted training times it entails. Therefore, a further important contribution of this thesis is a mechanism for ensuring that only the relevant constituents within a sentence are processed. This model is known as the STM-SSN. Because much of the discussion in this chapter concerns the theoretical abilities of the network, only the SSN is referred to throughout; the STM is an extension to the SSN, which enables more efficient training and processing by removing superfluous phases.

The SSN has been argued to be an appropriate connectionist architecture for natural language processing on the grounds that, firstly, the SSN can output structured representations, such as parse trees, and secondly, the SSN can generalise information learned across multiple entities, such as constituents within the parse tree. Further, because the SSN is essentially an SRN with added TSVB, the proven ability of the SRN to learn about patterns across time has been retained and augmented.

These properties have been tested experimentally by training the SSN on two toy grammars. In addition, the SSN has also been shown to learn to parse samples of real natural language. The ability of the SSN to learn both from toy grammars and from naturally occurring sentences makes it a significant new architecture for connectionist language learning. The specific features of the SSN are not, however, language specific, and so may be applied to a wider range of tasks requiring the output of similarly structured information.

6.2 Evaluation and Comparisons

In the introductory chapter it was argued that a machine learning algorithm may be judged either in terms of *results* or in terms of *principles*. This suggests that evaluating the SSN requires, respectively, a quantitative and a qualitative comparison with earlier work. This section considers these two issues in turn. In addition, although the principles behind the design of the SSN were motivated by specifics of the task of learning to parse, the very fact that the application area is language permits some comparisons to be drawn with areas in the cognitive literature: this is particularly true in the case of the STM, and is discussed in the third subsection below.

6.2.1 Quantitative comparisons

The value of quantitative comparisons of machine learning algorithms is purely to determine which has the better performance, based on some metric. To be effective, the comparisons must naturally be obtained from the same task and employ the same metric. In this sense, two sets of experiments with the SSN may be directly compared with results achieved in earlier work. The first of these is the recursive grammar, used by Hadley and Hayward [33] to verify that their Hebbian connectionist network could learn to generalise across syntactic constituents (see Section 2.3.4).

The recursive grammar experiments investigate the ability of the network to generalise across structure. For instance, when learning that a word appearing in the subject position of a sentence is a noun, this information should be generalised to the same word appearing in further syntactic positions, such as the object of a relative clause. As described in Section 4.2.2, the SSN's performance on this task is very good, demonstrating an ability to generalise across structure. This performance may be directly compared with that of the Hebbian connectionist network [33], which demonstrated the same ability to generalise across structure. This comparison tells us that there is nothing to choose between the two models on an empirical level on this particular test set. As discussed below, more qualitative considerations make the SSN more attractive, in terms of its potential for learning about more complex grammars. This potential is tested in the experiments with the Susanne corpus, described in Chapter 5, which forms the second set of experiments comparable with other work.

The experiments with the Susanne corpus are important in the literature on connectionist language learning because they use a corpus of naturally occurring text as a source of sentences. The target parse tree of the SSN is the parse tree information contained within the Susanne corpus, subject to a few structural limitations. One of the strengths of the SSN is its ability to incrementally output a parse tree as the sentence is input. To do this, the SSN uses a representation specifying the structural relationships between the current input word and the earlier portions of the parse tree. This representation is the GPS representation introduced in Section 5.1.3. Because the output of the SSN specifies a parse tree for the input sentence, it is possible to use standard measures for the precision and recall of constituents when compared with the target parse tree. This means that the SSN's performance may be compared directly with that of other parsers, such as the simple PCFG [48], as described in Section 3.3. However, although the best SSN achieves 80% average precision/recall on its test set, this result cannot simply be compared to the 72% achieved with simple PCFGs [48], because the corpora used differ in representation, size and contents.

In an extension to the work here, Henderson [39] has introduced a slight variant of the basic SSN model and compared its performance directly with that of PCFGs on identical corpora. In those results, the PCFG, due to the restricted size of the training set, was only able to parse half the test sentences, with a precision/recall figure of 54%/29%. In comparison, the SSN was able to parse all the sentences and yielded a performance of 65%/65%. Even when counting only the parsed sentences, the PCFG only had a performance of 54%/58%, compared to the SSN's performance of 68%/67% on that subset. These results show that the SSN compares well on an empirical level with a well-established, though basic, parser.

From the empirical evidence presented in this thesis, it is evident that the SSN is an effective model for learning to parse. This may be explained by the presence of Temporal Synchrony Variable Binding which provides an inherent ability to generalise across output structures. The effectiveness of the SSN is demonstrated by specific results with toy grammars and a corpus of naturally occurring text. The important point to note is that the SSN transfers its ability smoothly from the toy grammars to the natural text. This contrasts with other connectionist approaches to language learning whose performance on toy grammars has so far not been shown to scale up beyond certain toy grammars [43, 69]. This ability can be explained by the SSN's more effective use of its internal representation to support the parsing process, as described below.

6.2.2 Qualitative comparisons

This section compares three sets of earlier work with the SSN. The first is that of the holistic parser, which can output a sequential representation of a parse tree. The second is the range of alternative connectionist approaches to learning about language or handling structure. And thirdly, there are non-connectionist approaches to learning to parse.

Comparison with holistic parsers

The closest connectionist approach to the SSN in terms of parsing abilities is that known as holistic parsing, as discussed in Section 3.4.5 of Chapter 3. Holistic parsers, such as the Confluent Preorder Parser (CPP) [43], rely on the ability of a recurrent network to represent complex information within a distributed representation. This ability is applied to parsing by encoding a sequential representation of a parse tree (in preorder form) into the fixed-width internal representation of the network.

Two limitations have been seen in this approach. The first is that holistic parsing has only been demonstrated for limited forms of parse tree, e.g. the CPP requires parse trees to be of fixed valency so that the preorder encoding is unambiguous. Secondly, such parsers have not been demonstrated to learn beyond certain toy grammars. This latter fact has been commented on by [44, 69], and described as a fundamental limitation in the distributed representation itself, which can only store a certain amount of information reliably.

The SSN parser, in Chapter 5, has been shown to overcome these two limitations. First, the output representation of the SSN is not limited to fixed valency parse trees, but can produce a number of complex hierarchical representations. In particular, the SSN has demonstrated a capability in outputting parse trees from a standard (and only slightly modified) corpus of naturally occurring text. Second, the fact that the SSN does learn effectively from such a corpus of natural language, as shown here and in [39], demonstrates that the SSN has circumvented some of the capacity limitations apparent in the holistic approach.

The advantages of the SSN are due to the way it produces an output parse tree. The holistic parser builds up a distributed representation from its input sentence, which is then unfolded into a parse tree by a separate mechanism. This technique requires the holistic representation to encode *all* the information within the sentence required for constructing the parse tree; in some forms of holistic parser, additional information is also encoded to support various transformations [8, 14]. The SSN adopts almost the opposite approach to producing its parse tree. The SSN employs temporal synchrony, and uses the phases within each time period to represent the syntactic constituents within a parse tree. Because the structural information is held across phases, and the information about words and constituent labels is held within the activations on particular phases, the structural information about the parse tree is held independently from the information about individual constituents. This enables the SSN to generalise across syntactic constituents, as discussed in Section 2.3.5. But in addition, it means that the structural and constituent information need not be output at the same time. This idea enables the SSN parser to incrementally output parse trees: for each input word, the parser outputs the label for any newly introduced constituent, and every relation which that word has with previous constituents.

This manner of outputting parse trees by the SSN illustrates a combined usage of local and distributed representations. First, the internal units of the SSN use distributed representations to encode information about the entire sentence (on the non-pulsing units) and individual constituents (on the entire sentence). But the constituents are represented on independent phases within the time period, and so are, in a sense, locally represented. Hence the output of the parse tree can rely on the distributed internal representation for computation to ensure robust application of its learnt knowledge, but rely on the local use of time to clarify the relations between constituents. This contrasts with the internal representation used by the holistic parser, which is entirely distributed, both in terms of its description of individual constituents and their inter-relations. This representational technique is entirely due to the use of pulsing and non-pulsing units within the SSN; the pulsing units enable the SSN to output structured information, as discussed in Section 4.1.5, and the non-pulsing units interact with the pulsing units to ensure sufficient information is brought to bear when outputting the desired relations.

This incremental approach of the SSN to outputting a parse tree has a number of benefits in terms of the resources required of the network when parsing. First, the parser need not memorise what it has done before, because that information has already been output. Second, the parser is able to apply all its resources to outputting the relations between the current word and the previous constituents. Finally, the fact that not all previous constituents are likely to be required in practise means that the parser can safely forget about some of them for processing the remainder of a sentence. These three factors imply that the burden placed on the internal representation of the SSN parser is less than that of the holistic parser, and hence enable the SSN to focus its resources more on the task of parsing than on the task of memorising. These facts also suggest that the SSN may provide similar advantages in other domains requiring the output of similarly structured representations.

Comparison with other connectionist models

Apart from the holistic parsers, a number of other connectionist approaches to language learning have been discussed within this thesis. The holistic parser itself is an extension of the simpler recurrent networks, which have been popular in connectionist language learning due to their ability to learn about sequences over time. In particular, the Simple Recurrent Network (SRN) [18, 19] has been used extensively, although its successes have been confined to two basic tasks: learning to predict the next word in a sentence [19, 21, 81] or to assess whether a sentence is grammatical or not [60, 62]. Each of these only requires the SRN to output a single piece of information, either the predicted word for each output, or else to indicate the grammaticality after the whole sentence has been input. Neither of these tasks is structured in the sense of producing a representation such as a parse tree. The SSN itself is an extension of the SRN, and provides the SRN with the ability to output a hierarchically structured representation. As discussed above, the difference between the SSN's extension of SRNs and that of holistic parsers is that the SSN incrementally outputs its representation, using temporal synchrony to represent the relations between output constituents. This enables the SSN to generalise across syntactic constituents, which has been argued to play an important role in language learning [24, 30, 31].

The use of TSVB is not the only approach to including such generalisations within connectionist networks. An alternative approach to representing multiple entities within a connectionist network is that of tensor product variable binding [98], which uses a spatial distribution of units within the network instead of the temporal distribution used by TSVB. This form of representation is used by Hadley and Hayward [32, 33], as discussed in Section 2.3.4, in which a specific network architecture is shown to generalise across syntactic constituents in the context of a specific toy grammar. However, the network requires a specific internal structure to ensure that the appropriate generalisations are made, and this structure is not independently motivated. At present, the network only applies to extracting the case-role triple (agent,action,patient) from simple sentences, and extending this to more complex information would require extensive additions to the network. Given that the SSN in Section 4.2.2 has shown an equivalent ability to generalise across syntactic constituents, it is evident that the SSN and Hadley and Hayward's network provide for equivalent generalisation abilities, but the SSN is a more parsimonious architecture; this parsimony is borne out in Chapter 5 where essentially the same network is applied to naturally occurring sentences.

A second approach to generalising across syntactic constituents is that adopted by the Subsymbolic Parser [68], discussed in Section 3.4.6. With this network, three separate mechanisms guide the parsing process: a parser, a segmenter and a stack. SPEC is limited in its output to simple (agent,action,patient) triples, and does not provide any indication of their structural inter-relation. SPEC demonstrates generalisation across syntactic constituents by forming the same triples irrespective of the larger syntactic context of that part of the sentence. This is achieved through the segmenter and stack mechanisms which respectively determine where the breaks are between phrases in the sentence, and hold information about the state of the network across phrases. SPEC's output representation clearly is not comparable with that possible from the SSN, and in addition [69] points out that SPEC's internal representations suffer capacity limitations rather like those found in holistic parsers [44].

One further approach to extending recurrent networks to handle structured representations is to use Generalised Recursive Neurons (GRNs) [95], as discussed in Section 2.3.3. Rather as the standard recurrent network can be unfolded across the temporal structure of the input, a network of GRNs can be unfolded across the representational structure of the input. However, this approach is most suited to classifying structured input representations [25], or transforming the information at the nodes of equivalently structured input and output representations [26]. This appears to limit the applicability of the approach to an application such as parsing, in which a sequential input structure must be transformed into a hierarchical output structure: no such application has currently been demonstrated.

Comparison with other parsers

Chapter 3 introduced two other classes of parser, a symbolic-style technique known as PARSIFAL [63], and the statistical algorithm, Probabilistic Context-Free Grammars (PCFGs) [9, 48].

PARSIFAL is an example of a classical parser based on an incremental processing of the input sentence to create an output parse tree. This incremental processing uses a restricted form of look-ahead to substitute for search when parsing, so that it can consider the next couple of words when deciding how to parse the current word. The SSN parser adopts a similar incremental output for processing sentences as with PARSIFAL, although it cannot look-ahead at present. What is interesting is that the SSN learns its internal weights in order to generate the desired parse trees, whereas PARSIFAL must be preprogrammed with an adequate ruleset.

PCFGs are a relatively simple statistical algorithm for learning to parse from samples of natural language. They offer a useful benchmark for assessing the performance of the SSN parser, as is done in [39]. What is interesting about the PCFG is that the same general principles within the SSN are also present within the PCFG: the PCFG outputs structured information (the parse trees), learns about sequences of words (sentences), and uses a compositional grammar to inherently generalise learned information across syntactic constituents. The SSN combines TSVB with SRNs to provide a general-purpose architecture formed from these principles. The PCFG instead incorporates these principles into a specialist architecture for learning about natural language. The SSN is therefore interesting within the broader context of computational linguistics in that it achieves good results in a field previously dominated by specialised algorithms such as the PCFG.

6.2.3 Linguistic issues

The SSN developed in this thesis has been motivated by engineering concerns: how best to design a connectionist parser. However, because these concerns were largely motivated by the target application of parsing, it is reasonable to consider whether the success of the SSN implies any intrinsic linguistic value to these issues. There are two ideas that may be discussed in this context. The first is the type of representation, or parse tree, which the SSN parser can output, and the second is the role of the STM in reducing the resource requirements of the SSN parser.

Parse tree representations

The type of representation output by the connectionist models discussed in this thesis have differed in many respects with the amount of information output and the degree of structure contained within that information. At the simpler level are the basic tagger-type networks, such as the Simple Recurrent Network, which can output a label for each word based on its context, but not the relationship between each label and any others. More complex are those which output information grouped into the particular actions described within the sentence. These use a case-role representation, such as the (agent, action, patient) triples found within the SPEC [68] network and the Hebbian network of Hadley and Hayward [32, 33]. This representation expresses the semantic relationship between a number of entities within the sentence, and, with the Hebbian network, includes the further relationship between these triples contained in sentences with relative clauses.

However, neither of these capture the full generality of the structure contained within a parse tree. A parse tree can include information about the relationships between almost any words within a sentence. Both holistic parsers [44] and the SSN parser described in this thesis can output such representations. However, each has their own limitations. With the holistic parser, this limitation is contained in the requirement that the parse tree be encoded into a sequential description. For example, the Confluent Preorder Parser [43], a specific type of holistic parser, uses a preorder encoding of the parse tree which requires the parse tree to be of fixed valency.

The SSN parser can output parse trees of arbitrary valency, but is restricted in its current input/output format to parse trees whose number of internal nodes does not exceed that of the number of input words. This limitation restricts the complexity of the trees which the SSN can output, but is still adequate to output the parse trees from a standard corpus with only a few modifications necessary. An extension to the SSN input/output format is described in the next section which may overcome this limitation, and so enable the SSN to directly represent more complex parse trees.

A further extension to the SSN parser would improve the information within the labels output for each node within the parse tree. At present, these labels are taken from a standard set of category labels, as used within the SUSANNE corpus. In principle, these labels could be extended to include information output by more complex grammars, such as the slash categories within the Generalised Phrase Structure Grammar [29].

Bounded resource effects

The SSN, as defined in Chapter 4, can in principal use as many phases as it requires for solving a given problem. This of course means that the amount of computational time required by the SSN will grow in direct proportion to the number of phases. For instance, with the experiments in this thesis, each word on the input introduces one additional phase to the SSN, and so the total number of phases computed over during an entire sentence is $O(n^2)$, where n is the number of words within the sentence.

Considerations of the language-specific requirements of the task of learning to parse tell us that not all these phases may be required. The basic SSN retains all its introduced phases because, in theory, any later input may require an output to signal a relation to any earlier input. With the language experiments, each phase identifies a separate constituent within the sentence. The theoretical possibility that any input may be related to any other input has the practical consequence that all grammatical sentences should be treated equally. This, however, is manifestly not the case. For example, the sentence ‘the dog chased the cat that bit the rat that died’ is easily understood, but the equivalent ‘the rat that the cat that the dog chased bit died’ is not. The difference between these two sentences is that, in the former, each noun phrase may be forgotten as soon as the following verb has been seen. In the latter, all the noun phrases must be retained in memory so that their relationships to the later verbs can be determined.

The standard explanation for this phenomenon is in the form of a memory capacity limit [16, 70]; as the sentence is processed, there are limits on how many constituents may be retained for modification by later parts of the sentence. Sentences which violate these limits, such as the second one above, will be hard or impossible to understand. Within the context of the SSN parser, a memory limit on the number of constituents corresponds directly to a memory limit on the number of phases which the SSN can use for outputting relations between later words and earlier constituents. This limit is implemented in the SSN in the form of the STM.

The main advantage of the STM for the SSN is in reducing resource requirements for learning to parse. By having a fixed number of phases, the time requirements for the SSN are now only a fixed proportion of the length of the input sentence, i.e. $O(n)$. The fact that the STM is valuable in this specific task of learning to parse is a feature of natural language. However, the STM will be valuable, as an efficiency device, for any domain where the maximum length of the dependency which the SSN should output is significantly less than the maximum length of the total sequence which is being processed.

Section 5.2 contained a discussion of the dependency lengths within a sample of the SUSANNE corpus. There it was found that around 90% of the dependencies would fit within a STM of 3 items. This agrees extremely well with the predictions of four items by Cowan [16] and of seven items by Miller [70]. It must be remembered though that the syntactic parser would only form a part of a complete sentence analyser. A more complete system would additionally take into account the semantics of the situation and a number of default assumptions. Such additional factors account for the remaining 10% of the dependencies which extended almost up to the total length of the sentences. For example, the STM for active constituents only relates to the *short*-term memory of a cognitive system; there is an additional *long*-term memory, which would likely store more important aspects of the sentence, and provide default values for some of the words or particularly punctuation.

An interesting extension of the current work would be to embed the SSN parser within such a more comprehensive system; however, the potential of the SSN as a cognitive model is limited, unless a more plausible learning mechanism to backpropagation can be adopted.

6.3 Further Work

The results of this thesis may be used as a basis for further work in three broad areas: further applications, theoretical extensions and refinements for connectionist language learning.

Further applications One of the strengths of the SSN is that its architecture has been designed to be as general as possible for tasks requiring the output of structured representations. Therefore the good results achieved in one such task, natural language, should encourage use of the SSN in similar tasks. For example, work with protein structure uses grammatical models similar to those used for natural language [5] and the SSN may be readily applied to such domains.

The existence of a trainable architecture for TSVB connectionist networks also has implications for connectionist implementations of cognitive models. For example, the ACT model of Anderson [3] employs production-rules as the representation most likely to correspond with human cognition. The interaction of these production-rules is implemented as the spreading of activation between competing sets of neurons; an implementation intended to correspond with biological neurons. However, this implementation mechanism has a weakness in that production-rules support the use of variables, whereas the standard definition of neurons does not. Specifically, production-rules rely on an abstraction from specific pieces of data for their generality, and this generality is achieved by variabilising each chunk. Thus, "the variable is the critical element in achieving production-rule generality. Oddly, it is proper treatment of variables that is causing some of the greatest difficulties for connectionist theories of mind." [93].

The production-rule model of cognition relies upon two elements of learning. The first is a declarative form of learning, whereby the production-rule is formed. The second is a fine-tuning of this rule for specific applications. Within the neural implementation of ACT proposed by Anderson [3] it is noted that the problem is not the representation of the rule as neurons, but the problem of multiple instantiations of that rule. These multiple instantiations can be achieved if the neurons employ TSVB. Further, the training of these rules can be achieved using the extended form of Backpropagation Through Time developed in this thesis.

Extensions One area where the model is weak is the representation of relations, that is functions over tuples of variables. For example, one classic application of AI techniques is in performing actions on a particular world. The blocks-world example is one such, using a description of the world in terms of blocks, *block(x)*, and relations between them, *on(x, y)*. The task being to output operations such as *put_on(x, y)*. Could TSVB networks as described in this thesis be extended to this problem?

The extension of connectionist networks to using pulsing units involves the addition of a single dimension of data output to each unit. This process can be extended by the addition of further dimensions of output. The question is one of interpretation, and the existence of any constraints between the dimensions so constructed.

For the purpose of representing relations, doubly-pulsing units may be used, i.e. units which have two sets of additional dimensions. The phases on each of the two dimensions would be interpreted as representing the *same* set of variables, i.e. for a phase representing x in one dimension, there is a phase representing x in the second dimension. A constraint must accordingly be placed on the weights so that weight changes learned for phase x in one dimension are also applied to the weight changes learned for phase x in the second dimension. The relation *on(x,y)* would then be represented by having the unit pulse in the phase representing x in one dimension, and, during that phase, in the secondary phase representing y in the second dimension. It is an open question as to whether these relations may be trained from experience, so that the network, for instance, can learn to perform a task such as "construct-tower".

Refinements The basic aim in this thesis was to construct the 'cleanest' implementation of trainable TSVB networks possible. Hence, the use of different training options and the impact of different error functions was not investigated. This leads to the beneficial conclusion that the TSVB networks described in this thesis possess the abilities demonstrated in the experiments with natural language as inherent features of their design. More sophisticated error functions for training, more suitable output representations and different choices of network architecture can only improve on the results obtained here.

More language parsing The experiments in natural language parsing described in this thesis could be improved in a number of ways. First, it is important to work with larger corpora. For instance, statistical parsers are trained using databases up to a thousand times larger than those used here. It can also be expected that larger training sets will improve the performance of the parser, due to a greater exposure to all the possible parses. (This expectation is not without precedent: Eisner [17] reports that a parser trained with a corpus of 4000 words had twice the error of the same parser trained with a corpus of 25,000 words.) A further improvement would be to use a large corpus and training with the actual words as input to the network. Ideally, both of these extended experiments would be done with a corpus also used in training and testing a PCFG, so that a direct comparison may be made.

Finally, as discussed in Section 5.1.2, the limitation of the SSN's basic output representation to providing only one new constituent per input word enforced some changes in the parse trees within the training corpus. It is important to develop some further incremental representations of parse trees, or altered versions of the SSN, which would enable the SSN to output exactly the target parse tree supplied in the corpus. Two possibilities for this are as follows:

- Allow the SSN to use one phase for more than one constituent. For instance, two new consituents could be introduced in one phase by activating the P and G outputs within that phase: the G output implying that the new phase is also the grandparent of the current word, and hence the parent of the new constituent referred to by the P output. An additional 'great-grandparent' output relation would also be required, for specifying an earlier phase as the parent node for the parent of the two new constituents. This scheme has two difficulties. Most importantly, the phase numbers can no longer be used to unambiguously indicate an earlier constituent, although possibly the constituent labels could be used for this. In addition, it would be difficult to provide a label for the two new constituents, as previously the label outputs always referred to the unique new constituent.

- A second approach is to provide the SSN with an unused phase (this could be provided in the input, i.e. a specific input unit for the unused phase, active in the relevant phase). A word which introduced two new constituents would then activate the P output in its own phase, the G output in the unused phase, and use a 'great-grandparent' output unit to indicate any earlier phase as the parent node of the one introduced by the G output. Note that this approach avoids the labelling problems of the previous one, because each new constituent is in its own unique phase.

Such approaches can evidently be extended for any finite number of constituents which might be introduced at any given time. Analysis of different corpora would be needed for determining the precise number required.

6.4 Conclusion

This thesis has defined a new connectionist architecture, the Simple Synchrony Network, and empirically verified its ability to generalise in specific ways across structured representations. As is implicit in the suggestions for further work above, the theoretical and experimental results of this thesis extend beyond language learning. It is apparent that the SSN architectures and short-term memory mechanisms are not specifically adapted to natural language. Thus their ability to learn about and manipulate structured information is a very general one. Taking the specific input-output representations used in this thesis as a model, the SSN may be applied to a range of other domains which also use structured information. The SSN therefore extends the range of applications for which connectionist networks may be used.

Appendix A

Experimental results

This Appendix contains tabulated results for the experiments on the shorter training set described in Chapter 5. Four network sizes were trained for each type of Simple Synchrony Network, and three lengths of short-term memory (STM) queue for each network size. Each network was trained for 100 epochs, and evaluated after every 20 epochs against the short cross validation set which contains sentences of less than 15 word-tags. These tables contain the results of these evaluations, each table contains the results from the four network sizes of a particular network type and STM length.

Type A Networks: STM of 3							
Epochs	/38	Precision	Recall	G	P	S	Label
25 units in hidden layer							
20	0	17.9	14.8	9.2	49.9	0.0	84.2
40	0	28.5	23.1	9.2	58.1	0.0	84.0
60	0	16.7	13.5	8.4	49.0	2.7	84.2
80	0	20.8	14.4	9.2	42.5	2.7	84.2
100	0	19.7	13.5	7.6	41.4	0.0	84.4
50 units in hidden layer							
20	0	26.1	22.7	6.9	60.3	2.7	84.4
40	0	26.3	22.7	6.9	61.6	2.7	84.0
60	0	25.0	21.4	17.6	61.4	2.7	84.0
80	0	28.0	22.2	16.0	56.7	2.7	84.2
100	0	29.7	22.3	13.7	55.6	16.0	84.0
75 units in hidden layer							
20	0	27.6	23.6	22.9	61.1	18.7	84.4
40	0	26.4	22.7	18.3	60.8	17.3	84.7
60	0	28.9	24.5	22.9	60.8	18.7	84.9
80	0	29.4	23.1	18.3	57.3	18.7	84.7
100	0	30.3	23.6	22.9	56.7	18.7	84.4
100 units in hidden layer							
20	0	25.7	21.4	16.8	60.3	18.7	84.4
40	0	27.7	23.1	17.6	61.4	18.7	84.4
60	0	28.2	24.0	22.9	61.6	18.7	84.9
80	0	27.1	22.7	17.6	61.1	18.7	84.7
100	0	29.7	24.9	31.3	60.3	18.7	84.7

Table A.1: SSN Type A: Cross validation results for STM length 3.

Type A Networks: STM of 6							
Epochs	/38	Precision	Recall	G	P	S	Label
25 units in hidden layer							
20	0	27.6	21.8	0.8	57.8	0.0	84.7
40	0	27.7	22.7	3.8	60.0	4.0	84.7
60	0	27.5	22.7	3.8	60.8	4.0	84.7
80	0	27.8	22.7	3.8	60.3	4.0	84.7
100	0	29.3	24.0	8.4	60.5	4.0	84.7
50 units in hidden layer							
20	0	27.8	25.8	9.9	63.8	16.0	84.7
40	0	28.6	26.6	13.7	64.7	17.3	84.9
60	0	28.6	26.6	13.7	64.7	17.3	84.9
80	0	28.6	26.6	17.6	64.7	18.7	84.9
100	0	28.6	26.6	17.6	64.7	18.7	84.9
75 units in hidden layer							
20	0	28.5	26.6	16.0	64.7	17.3	84.9
40	0	28.6	26.6	17.6	64.7	17.3	84.9
60	0	28.6	26.6	17.6	64.7	18.7	84.9
80	0	28.5	26.6	16.8	64.7	18.7	84.9
100	0	28.5	26.6	26.0	64.7	18.7	84.9
100 units in hidden layer							
20	0	30.5	25.3	32.1	60.3	18.7	84.4
40	0	30.5	25.3	33.6	60.8	18.7	84.7
60	0	30.4	25.3	33.6	61.4	18.7	84.7
80	0	30.4	28.4	33.6	65.2	18.7	84.4
100	0	30.4	28.4	33.6	65.2	18.7	84.4

Table A.2: SSN Type A: Cross validation results for STM length 6.

Type A Networks: STM of 10							
Epochs	/38	Precision	Recall	G	P	S	Label
25 units in hidden layer							
20	0	18.2	14.0	0.8	47.7	0.0	84.7
40	0	17.9	14.4	0.8	49.6	0.0	84.7
60	0	17.2	13.5	0.8	48.8	0.0	84.7
80	0	17.9	14.4	0.8	49.9	0.0	84.7
100	0	17.9	14.4	0.8	49.9	0.0	84.7
50 units in hidden layer							
20	0	27.9	23.1	16.0	60.3	17.3	84.9
40	0	27.9	23.1	16.0	60.3	17.3	84.9
60	0	28.6	26.6	14.5	64.1	17.3	84.9
80	0	28.6	26.6	16.8	64.1	17.3	84.9
100	0	28.6	26.6	16.8	64.1	17.3	84.9
75 units in hidden layer							
20	0	28.6	26.6	16.8	64.4	17.3	84.9
40	0	28.6	26.6	17.6	64.9	17.3	84.9
60	0	28.5	26.6	17.6	65.2	18.7	84.9
80	0	28.5	26.6	17.6	65.2	17.3	84.9
100	0	28.5	26.6	17.6	65.2	18.7	84.9
100 units in hidden layer							
20	0	30.5	25.3	27.5	60.3	18.7	84.7
40	0	31.0	28.8	25.2	64.7	18.7	84.9
60	0	30.5	25.3	30.5	60.8	18.7	84.7
80	0	31.8	29.7	30.5	65.2	18.7	84.7
100	0	31.8	29.7	30.5	65.2	18.7	84.7

Table A.3: SSN Type A: Cross validation results for STM length 10.

Type B Networks: STM of 3							
Epochs	/38	Precision	Recall	G	P	S	Label
20 units in each hidden layer							
20	4	45.1	42.4	37.4	79.5	61.3	86.6
40	6	48.8	46.3	54.2	81.6	60.0	88.8
60	5	50.2	48.5	50.4	83.6	56.0	90.9
80	5	50.5	48.0	55.7	83.8	56.0	92.1
100	5	51.4	48.5	56.5	82.5	58.7	90.4
40 units in each hidden layer							
20	5	47.7	45.4	43.5	81.4	56.0	87.6
40	6	54.8	52.4	52.7	89.3	57.3	90.9
60	5	62.8	59.0	68.7	87.9	64.0	93.3
80	6	61.1	59.0	70.2	91.0	61.3	94.5
100	7	67.7	64.2	74.0	91.2	61.3	95.9
60 units in each hidden layer							
20	5	47.8	42.4	47.3	78.1	62.7	88.3
40	3	55.7	51.5	66.4	86.8	62.7	93.1
60	6	60.9	57.2	70.2	88.8	62.7	95.2
80	7	66.2	64.2	68.7	93.2	65.3	94.5
100	6	60.9	58.5	69.5	89.3	61.3	94.5
80 units in each hidden layer							
20	5	50.5	45.9	57.3	81.4	65.3	89.7
40	6	58.0	52.0	67.2	84.4	64.0	92.3
60	4	65.1	61.1	63.4	89.6	65.3	95.2
80	8	67.4	64.2	70.2	92.1	65.3	96.9
100	10	71.5	71.2	74.8	95.6	70.7	96.4

Table A.4: SSN Type B: Cross validation results for STM length 3.

Type B Networks: STM of 6							
Epochs	/38	Precision	Recall	G	P	S	Label
20 units in each hidden layer							
20	5	47.2	44.5	46.6	77.8	64.0	86.1
40	5	50.9	47.6	47.3	77.5	64.0	87.1
60	6	51.4	48.5	47.3	81.1	72.0	90.0
80	6	50.0	47.6	44.3	82.7	70.7	91.9
100	5	45.0	49.0	38.9	80.3	76.0	91.4
40 units in each hidden layer							
20	5	45.6	42.8	42.0	79.5	81.3	88.3
40	6	49.8	45.4	55.7	80.3	81.3	92.1
60	7	49.5	46.3	55.0	84.1	78.7	92.1
80	9	57.1	54.1	57.3	87.9	76.0	94.5
100	8	61.1	57.6	58.8	89.3	78.7	95.9
60 units in each hidden layer							
20	6	48.0	46.7	44.3	82.2	84.0	86.8
40	7	59.1	55.5	56.5	86.6	84.0	90.7
60	9	60.0	55.0	60.3	86.8	81.3	90.9
80	8	59.5	55.9	71.8	86.6	82.7	91.9
100	7	61.4	57.6	62.6	89.6	82.7	91.6
80 units in each hidden layer							
20	6	49.5	44.5	46.6	76.2	81.3	86.4
40	7	57.6	54.6	49.6	80.3	74.7	90.9
60	8	62.3	59.8	64.1	92.3	81.3	94.5
80	10	62.4	60.3	67.9	93.7	85.3	94.7
100	12	65.3	64.2	82.4	92.6	85.3	96.2

Table A.5: SSN Type B: Cross validation results for STM length 6.

Type B Networks: STM of 10							
Epochs	/38	Precision	Recall	G	P	S	Label
20 units in each hidden layer							
20	6	46.6	45.0	40.5	80.5	81.3	86.4
40	6	49.8	47.6	45.0	81.6	74.7	89.5
60	6	48.4	46.3	42.0	82.7	70.7	90.2
80	5	44.6	41.5	45.0	80.0	62.7	89.7
100	7	50.0	49.3	44.3	83.3	69.3	88.8
40 units in each hidden layer							
20	6	44.2	41.9	48.9	78.1	74.7	85.2
40	6	50.9	47.6	51.1	82.7	77.3	89.2
60	6	53.7	51.1	52.7	86.3	74.7	93.1
80	8	53.3	54.7	56.5	87.4	78.7	94.7
100	9	57.9	55.9	62.6	89.3	84.0	95.5
60 units in each hidden layer							
20	5	46.4	42.8	49.6	77.0	76.0	86.4
40	5	54.1	52.0	45.0	89.0	82.7	90.7
60	7	55.9	52.0	50.4	87.9	80.0	91.6
80	7	54.7	53.7	53.4	88.2	86.7	93.3
100	9	61.2	58.5	61.1	90.1	89.3	94.3
80 units in each hidden layer							
20	6	46.1	43.7	42.0	81.1	81.3	88.3
40	7	48.3	44.1	60.3	87.7	78.7	90.4
60	5	47.2	45.0	48.1	78.1	80.0	85.2
80	10	60.8	59.0	63.4	92.3	74.7	95.7
100	11	66.4	65.5	65.6	92.9	85.3	97.4

Table A.6: SSN Type B: Cross validation results for STM length 10.

Type C Networks: STM of 3							
Epochs	/38	Precision	Recall	G	P	S	Label
20 units in each hidden layer							
20	5	43.6	41.9	45.8	78.4	56.0	86.6
40	4	50.5	45.9	56.5	80.5	50.7	88.5
60	4	54.5	50.7	56.5	84.7	54.7	91.4
80	5	57.0	55.0	56.5	84.7	54.7	91.4
100	5	61.0	58.1	61.8	89.9	60.0	94.3
40 units in each hidden layer							
20	5	48.6	47.2	51.1	83.3	64.0	88.0
40	6	53.6	52.0	63.4	86.8	64.0	90.0
60	7	61.6	58.1	65.6	88.5	65.3	93.5
80	6	58.3	55.5	63.4	87.9	62.7	94.7
100	9	64.6	67.6	73.3	89.6	66.7	96.4
60 units in each hidden layer							
20	5	50.0	47.6	45.8	83.8	60.0	87.3
40	6	57.8	56.3	64.1	90.4	62.7	92.3
60	6	62.1	60.7	67.2	94.0	64.0	94.7
80	9	66.7	65.5	71.0	94.2	62.7	96.4
100	9	68.6	67.7	74.8	95.3	62.7	97.4
80 units in each hidden layer							
20	5	52.8	50.2	55.7	87.1	68.0	90.2
40	5	58.1	55.0	62.6	90.7	65.3	93.3
60	6	61.2	60.7	64.1	93.4	62.7	94.0
80	9	71.7	70.7	71.0	95.6	69.3	96.7
100	9	74.4	73.8	71.8	97.3	70.7	97.8

Table A.7: SSN Type C: Cross validation results for STM length 3.

Type C Networks: STM of 6							
Epochs	/38	Precision	Recall	G	P	S	Label
20 units in each hidden layer							
20	6	45.6	43.2	42.7	78.1	70.7	87.3
40	6	46.8	44.1	40.5	80.5	64.0	86.1
60	6	49.3	45.9	48.1	83.6	72.0	87.1
80	5	47.4	44.5	45.0	81.9	70.7	87.1
100	5	48.2	46.3	43.5	81.9	72.0	86.6
40 units in each hidden layer							
20	6	45.9	44.5	45.8	80.0	72.0	85.6
40	6	47.7	44.5	50.4	79.5	74.7	88.3
60	7	48.2	45.9	56.5	82.2	74.7	91.1
80	9	55.5	52.8	62.6	85.5	78.7	92.1
100	10	57.5	54.1	58.0	86.0	82.7	93.5
60 units in each hidden layer							
20	6	48.6	44.1	51.1	78.1	82.7	88.0
40	6	59.1	55.5	62.6	86.6	84.0	90.9
60	8	61.1	57.6	67.2	89.9	86.7	93.1
80	8	58.6	56.8	62.6	88.2	85.3	93.3
100	8	59.9	58.1	77.9	91.5	89.3	94.3
80 units in each hidden layer							
20	4	45.1	41.9	49.6	78.4	80.0	87.1
40	5	57.5	53.7	56.5	85.8	85.3	91.9
60	5	58.1	55.0	57.3	89.0	85.3	94.7
80	7	61.6	58.1	68.7	89.0	81.3	95.0
100	8	63.6	61.1	68.7	91.5	81.3	95.5

Table A.8: SSN Type C: Cross validation results for STM length 6.

Type C Networks: STM of 10							
Epochs	/38	Precision	Recall	G	P	S	Label
20 units in each hidden layer							
20	6	42.5	40.6	33.6	76.2	77.3	85.9
40	5	44.0	41.5	46.6	77.3	70.7	86.1
60	6	49.1	46.3	43.5	80.5	74.7	88.8
80	6	49.5	45.9	46.6	79.7	74.7	87.8
100	5	44.8	41.0	47.3	78.4	78.7	88.8
40 units in each hidden layer							
20	6	45.1	40.2	43.5	75.1	77.3	86.1
40	6	45.5	43.7	51.1	79.5	82.7	89.2
60	5	41.3	39.3	45.0	77.3	81.3	85.4
80	6	50.5	48.0	48.1	83.0	81.3	90.7
100	7	50.9	47.6	45.8	81.9	73.3	91.9
60 units in each hidden layer							
20	5	49.8	47.6	49.6	85.5	80.0	86.6
40	7	48.8	45.4	54.2	83.8	74.7	82.5
60	8	53.6	52.4	62.6	84.7	73.3	90.9
80	9	56.0	53.3	60.3	85.8	80.0	92.3
100	11	63.3	61.1	67.2	91.8	92.0	93.1
80 units in each hidden layer							
20	6	45.4	43.2	45.8	80.0	85.3	87.1
40	5	54.2	50.2	47.3	84.9	86.7	90.9
60	7	54.8	52.0	58.0	86.8	89.3	92.8
80	10	60.9	57.2	65.6	89.0	92.0	95.0
100	10	59.2	56.3	67.9	89.6	86.7	93.5

Table A.9: SSN Type C: Cross validation results for STM length 10.

Bibliography

[1] K. Aizawa. Exhibiting versus explaining systematicity : a reply to Hadley and Hayward. *Minds and Machines*, 7:39–55, 1997.

[2] J. Allen. *Natural Language Understanding*. Benjamin/Cummings, California, 1987.

[3] J. A. Anderson. *Rules of the Mind*. Lawrence Erlbaum, 1993.

[4] A. D. Baddeley. *Working Memory*. New York: Oxford University Press, 1986.

[5] P. Baldi and S. Brunah. *Bioinformatics*. MIT Press: Adaptive Computation and Machine Learning Series, 1998.

[6] R. Callan and D. Palmer-Brown. An analytical technique for fast and reliable derivation of connectionist symbol structure representations. *Connection Science*, 9(2):139–159, 1997.

[7] D. Chalmers. Syntactic transformations on distributed representations. *Connection Science*, 2:53–62, 1990.

[8] D. Chalmers. Syntactic transformations of distributed representations. In N. Sharkey, editor, *Connectionist natural language processing*, pages 46–55. Boston:Kluwer, 1992.

[9] E. Charniak. *Statistical Language Learning*. Cambridge MA:MIT Press, 1993.

[10] E. Charniak. Tree-bank grammars. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 1031–1036, Menlo Park, AAAI Press/MIT Press, 1996.

[11] E. Charniak. Statistical techniques for natural language parsing. *AI Magazine*, 18:33–43, 1997.

[12] N. Chomsky. *Aspects of the Theory of Syntax*. MIT Press, Cambridge, MA, 1965.

[13] N. Chomsky. *Lectures on Government and Binding*. Foris Publications, Dordrecht, Holland, 1981.

[14] L. Chrisman. Learning recursive distributed representations for holistic computation. *Connection Science*, 3:345–366, 1991.

[15] G. W. Cottrell and M. K. Fleming. Face recognition using unsupervised feature extraction. In *Proceedings of the International Neural Network Conference*, pages 322–325, Paris, France, 1990.

[16] N. Cowan. The magical number 4 in short-term memory: A reconsideration of mental storage capacity. *Behavioral and Brain Sciences*, 24(1), in press.

[17] J. M. Eisner. An empirical comparison of probability models for dependency grammar. Technical report, University of Philadelphia, USA, 1997.

[18] J. L. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.

[19] J. L. Elman. Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning*, 7:195–225, 1991.

[20] J. L. Elman. Grammatical structure and distributed representations. In S. Davis, editor, *Connectionism: Theory and Practice*. Oxford University Press, 1992, 1992.

[21] J. L. Elman. Learning and development in neural networks: the importance of starting small. *Cognition*, 48:71–99, 1993.

[22] S. Fahlman and C. Lebiere. The CASCADE-CORRELATION learning architecture. Technical Report CMU-CS-90-100, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, 1990.

[23] J. A. Fodor and B. McLaughlin. Connectionism and the problem of systematicity: why Smolensky's solution doesn't work. *Cognition*, 35:183–204, 1990.

[24] J. A. Fodor and Z. W. Pylyshyn. Connectionism and cognitive architecture: a critical analysis. *Cognition*, 28:3–71, 1988.

[25] P. Frasconi, M. Gori, and A. Sperduti. On the efficient classification of data structures by neural networks. In *International Joint Conference on Artificial Intelligence*, 1997.

[26] P. Frasconi, M. Gori, and A. Sperduti. A general framework for adaptive processing of data structures. *IEEE Transactions on Neural Networks*, 9:768–786, 1998.

[27] L. Frazier and J. D. Fodor. The sausage machine: a new two-stage parsing model. *Cognition*, 6:291–325, 1978.

[28] R. Garside, G. Leech, and G. Sampson, editors. *The Computational Analysis of English: a corpus-based approach*. Longman Group UK Limited, 1987.

[29] G. Gazdar, E. Klein, G. Pullum, and S. Ivan. *Generalized Phrase Structure Grammar*. Blackwell, Oxford, UK, 1985.

[30] R. F. Hadley. Systematicity in connectionist language learning. *Mind and Language*, 9:247–72, 1994.

[31] R. F. Hadley. Systematicity revisited: reply to Christiansen and Chater and Niklasson and van Gelder. *Mind and Language*, 9:431–44, 1994.

[32] R. F. Hadley and M. B. Hayward. Strong semantic systematicity from unsupervised connectionist learning. In *Proceedings of the Seventeenth Conference of the Cognitive Science Society*, Pittsburgh, PA., 1995.

[33] R. F. Hadley and M. B. Hayward. Strong semantic systematicity from Hebbian connectionist learning. *Minds and Machines*, 7:1–37, 1997.

[34] D. J. Hand. *Construction and Assessment of Classification Rules*. John Wiley and Sons Ltd, 1997.

[35] J. B. Henderson. Structure unification grammar: A unifying framework for investigating language. Technical Report MS-CIS-90-94, University of Pennsylvania, Philadelphia, PA, 1990.

[36] J. B. Henderson. *Description Based Parsing in a Connectionist Network.* PhD thesis, University of Pennsylvania, 1994.

[37] J. B. Henderson. A connectionist architecture with inherent systematicity. In *Proceedings of the Eighteenth Conference of the Cognitive Science Society*, pages 574–579, La Jolla, CA, 1996.

[38] J. B. Henderson. Constituency, context, and connectionism in syntactic parsing. In M. Crocker, M. Pickering, and C. Clifton, editors, *Architectures and Mechanisms for Language Processing*, pages 189–209. Cambridge University Press, Cambridge UK, 2000.

[39] J. B. Henderson. A neural network parser that handles sparse data. In *Proceedings of the 6th International Workshop on Parsing Technologies*, pages 123–134, Trento, Italy, 2000.

[40] J. B. Henderson and P. C. R. Lane. A connectionist architecture for learning to parse. In *Proceedings of the 17th International Conference on Computational Linguistics and the 36th Annual Meeting of the Association for Computational Linguistics (COLING-ACL'98)*, pages 531–537, University of Montreal, Canada, 1998.

[41] J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the Theory of Neural Computation.* Addison-Wesley, 1991.

[42] E. K. S. Ho and L. W. Chan. Efficient connectionist representations of syntactic parse trees for grammatical inference. Technical Report TR-95-15, UCI, Irvine, 1995.

[43] E. K. S. Ho and L. W. Chan. Confluent preorder parsing of deterministic grammars. *Connection Science*, 9:269–293, 1997.

[44] E. K. S. Ho and L. W. Chan. How to design a connectionist holistic parser. *Neural Computation*, 11(8):1995–2016, 1999.

[45] K. Hornik, W. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[46] W. Y. Huang and R. P. Lippmann. Neural net and traditional classifiers. In Anderson, editor, *Neural Information Processing Systems*, pages 387–396, 1988.

[47] M. F. St. John and J. L. McClelland. Learning and applying contextual constraints in sentence comprehension. *Artificial Intelligence*, 46:217–257, 1990.

[48] M. Johnson. PCFG models of linguistic tree representations. *Computational Linguistics*, 24:613–632, 1998.

[49] M. Jordan. Attractor dynamics and parallelism in a connectionist sequential machine. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, pages 531–546, 1986.

[50] J. Kimball. Seven principles of surface structure parsing in natural language. *Cognition*, 2:15–47, 1976.

[51] T. Kohonen. Self-organization of very large document collections: State of the art. In Niklasson et al. [74], pages 65–74.

[52] M. Kubat, I. Bratko, and R.S. Michalski. A review of machine learning methods. In R. S. Michalski, I. Bratko, and M. Kubat, editors, *Machine Learning and Data Mining*, pages 3–70. John Wiley & Sons, Chicester, England, 1998.

[53] S. C. Kwasny and K. A. Faisal. Symbolic parsing via subsymbolic rules. In J. Dinsmore, editor, *The symbolic and connectionist paradigm: Closing the gap*, pages 209–236. Hillsdale, NJ: Erlbaum, 1992.

[54] P. C. R. Lane. Simple Synchrony Networks: Learning generalisations across syntactic constituents. In H. Prade, editor, *Proceedings of the Thirteenth European Conference in Artificial Intelligence*, pages 469–470, Brighton, UK, 1998. John Wiley & Sons, UK.

[55] P. C. R. Lane and J. B. Henderson. Simple Synchrony Networks : Learning to parse natural language with Temporal Synchrony Variable Binding. In Niklasson et al. [74], pages 615–620.

[56] P. C. R. Lane and J. B. Henderson. Incremental syntactic parsing of natural language corpora with Simple Synchrony Networks. *IEEE Transactions on Knowledge and Data Engineering*, in press.

[57] K. J. Lang, A. H. Waibel, and G. E. Hinton. A time-delay neural network architecture for isolated word recognition. *Neural Networks*, 3:33–43, 1990.

[58] R. W. Langacker. *Foundations of Cognitive Grammar : Theoretical Perspectives*, volume 1. Stanford: Stanford University Press, 1987.

[59] P. Langley. *Elements of Machine Learning*. Morgan Kaufmann, 1996.

[60] S. Lawrence, S. Fong, and C. L. Giles. Natural language grammatical inference: A comparison of recurrent neural networks and machine learning methods. In S. Wermter, E. Riloff, and G. Scheler, editors, *Connectionist, Statistical and Symbolic Approaches to Learning for Natural Language Processing*, Lecture Notes on Artificial Intelligence. Springer-Verlag, New York, 1996.

[61] S. Lawrence, C. L. Giles, and S. Fong. On the applicability of neural network and machine learning methodologies to natural language processing. Technical Report UMIACS-TR-95-64, Institute for Advanced Computer Studies, University of Maryland, USA, 1995.

[62] S. Lawrence, C. L. Giles, and S. Fong. Natural language grammatical inference with recurrent neural networks. *IEEE Transactions on Knowledge and Data Engineering*, in press.

[63] M. Marcus. *A theory of syntactic recognition for natural language*. Cambridge MA: MIT Press, 1980.

[64] M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz. Building a large annotated corpus of English: The Penn treebank. *Computational Linguistics*, 19:313–330, 1993.

[65] W. S. McCulloch and W. Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[66] K. McGarry, S. Wermter, and J. MacIntyre. Hybrid neural systems: From simple coupling to fully integrated neural networks. *Neural Computing Surveys*, 2:62–94, 1999.

[67] I. Melčuk. *Dependency Syntax: Theory and Practice*. SUNY Press, 1988.

[68] R. Miikkulainen. Subsymbolic case-role analysis of sentences with embedded clauses. *Cognitive Science*, 20:47–73, 1996.

[69] R. Miikkulainen. Natural language processing with subsymbolic neural networks. In A. Browne, editor, *Neural Network Perspectives on Cognition and Adaptive Robotics*, pages 120–139, 1997.

[70] G. A. Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review*, 63:81–97, 1956.

[71] M. Minsky. A framework for representing knowledge. In P. H. Winston, editor, *The Psychology of Computer Vision*, pages 221–277. New York, McGraw-Hill, 1975.

[72] M. Minsky and S. Papert. *Perceptrons*. MIT Press, Cambridge, MA, 1988.

[73] T. Mitchell. *Machine Learning*. McGraw Hill Companies Ltd, 1997.

[74] L. Niklasson, M. Boden, and T. Ziemke, editors. *Proceedings of the Eighth International Conference on Artificial Neural Networks*, Skövde, Sweden, 1998.

[75] B. A. Pearlmutter. Learning state space trajectories in recurrent neural networks. *Neural Computation*, 1:263–269, 1989.

[76] B. Pell. METAGAME: A new challenge for games and learning. In H. J. van der Herik and L. V. Allis, editors, *Heuristic Programming in Artificial Intelligence 3 – The Third Computer Olympiad*. Ellis Horwood, 1992.

[77] J. Pollack. Recursive distributed representations. *Artificial Intelligence*, 46:77–105, 1990.

[78] D. A. Pomerleau. Knowledge-based training of artificial neural networks for autonomous robot driving. In J. Connell and S. Mahadevan, editors, *Robot Learning*, pages 19–43. Boston:Kluwer Academic Publishers, 1993.

[79] J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[80] R. Reilly. Connectionist techniques for on-line parsing. *Network*, 3:37–45, 1992.

[81] R. G. Reilly. Enriched lexical representations, large corpora, and the performance of srns. In Niklasson et al. [74], pages 405–410.

[82] T. J. Reynolds, E. B. Pizzolato, and C. Antoniou. Multinet: A new connectionist architecture for speech recognition. In Niklasson et al. [74], pages 257–262.

[83] D. L. T. Rohde and D. C. Plaut. Simple recurrent networks and natural language: How important is starting small? In *Proceedings of the Nineteenth Annual Conference of the Cognitive Science Society*, pages 656–661. Hillsdale, NJ: Lawrence Erlbaum Associates, 1997.

[84] D. L. T. Rohde and D. C. Plaut. Language acquisition in the absence of explicit negative evidence: how important is starting small? *Cognition*, 72:67–109, 1999.

[85] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In Rumelhart et al. [86].

[86] D. E. Rumelhart, J. L. McClelland, and the PDP Research Group, editors. *Parallel Distributed Processing: Explorations in the microstructure of cognition*, volume 1. MIT Press: Cambridge MA, 1986.

[87] G. Sampson. *English for the Computer*. Oxford University Press, Oxford, UK, 1995.

[88] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3:211–229, 1959.

[89] T. J. Sejnowski and C. R. Rosenberg. Parallel networks that learn to pronounce English text. *Complex Systems*, 1:145–168, 1987.

[90] N. E. Sharkey and A. J. C. Sharkey. A modular design for connectionist parsing. In *Proceedings of the Twentieth Workshop on Language Technology 3: Connectionist and Natural Language Processing*, pages 87–96, 1992.

[91] L. Shastri. Exploiting temporal binding to learn relational rules within a connectionist network. Technical Report TR-97-003, International Computer Science Institute, Berkeley, CA, 1997.

[92] L. Shastri and V. Ajjanagadde. From simple associations to systematic reasoning: A connectionist representation of rules, variables, and dynamic bindings using temporal synchrony. *Behavioral and Brain Sciences*, 16:417–494, 1993.

[93] P. Smolensky. Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial Intelligence*, 46:159–216, 1990.

[94] A. Sperduti. Stability properties of labeling recursive auto-associative memory. *IEEE Transactions on Neural Networks*, 6:1452–1460, 1995.

[95] A. Sperduti and T. Starita. Supervised neural networks for classification of structures. *IEEE Transactions on Neural Networks*, 8:714–735, 1997.

[96] G. Z. Sun, C. L. Giles, H. H. Chen, and Y. C. Lee. The neural network pushdown automata: Model, stack and learning simulations. Technical Report UMIACS-TR-93-77 & CS-TR03118, College Park: University of Maryland, 1993.

[97] R. Sun. Schemas, logics, and neural assemblies. *Applied Intelligence*, 5:83–102, 1995.

[98] B. B. Tesar and P. Smolensky. Synchronous firing variable binding is a tensor product representation with temporal role vectors. In *Proceedings of the Sixteenth Conference of the Cognitive Science Society*, pages 870–875, 1994.

[99] C. von der Malsburg. The correlation theory of brain function. Technical Report 81-2, Max-Planck-Institute for Biophysical Chemistry, Gottingen, 1981.

[100] R. L. Watrous. GRADSIM:a connectionist network simulator using gradient optimization techniques. Technical Report LS92-02, Siemens Research Laboratory, Princeton, NJ, 1993.

[101] S. Wermter and R. Sun, editors. *Hybrid Neural Systems*. Springer, Heidelberg, New York, 2000.

[102] R. J. Williams and D. Zipser. Experimental analysis of the real-time recurrent learning algorithm. *Connection Science*, 1:87–111, 1989.

[103] R. J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1:270–280, 1989.

[104] R. J. Williams and D. Zipser. Gradient-based learning algorithms for recurrent networks and their computational complexity. In Y. Chauvin and D. E. Rumelhart, editors, *Backpropagation: Theory, architectures amd applications*, pages 433–486. Lawrence Erlbaum Associates, Inc., 1995.

[105] P. H. Winston. *Artificial Intelligence*. second edition, Addison-Wesley, 1984.

[106] W. A. Woods. Transition network grammars for natural language analysis. *Communications of the ACM*, 13:591–606, 1970.

[107] J. Zavrel and J. Venstra. The language environment and syntactic word-class acquisition. In *Proceedings of the Groningen Assembly on Language Acquisition*, Groningen, 1996.