Accepted for publication, IEEE Transactions on Knowledge and Data Engineering.

Natural Language Grammatical Inference with Recurrent Neural Networks

Steve Lawrence, C. Lee Giles,* Sandiway Fong
{lawrence,giles,sandiway}@research.nj.nec.com

NEC Research Institute
4 Independence Way
Princeton, NJ 08540

Abstract

This paper examines the inductive inference of a complex grammar with neural networks – specifically, the task considered is that of training a network to classify natural language sentences as grammatical or ungrammatical, thereby exhibiting the same kind of discriminatory power provided by the Principles and Parameters linguistic framework, or Government-and-Binding theory. Neural networks are trained, without the division into learned vs. innate components assumed by Chomsky, in an attempt to produce the same judgments as native speakers on sharply grammatical/ungrammatical data. How a recurrent neural network could possess linguistic capability, and the properties of various common recurrent neural network architectures are discussed. The problem exhibits training behavior which is often not present with smaller grammars, and training was initially difficult. However, after implementing several techniques aimed at improving the convergence of the gradient descent backpropagation-through-time training algorithm, significant learning was possible. It was found that certain architectures are better able to learn an appropriate grammar. The operation of the networks and their training is analyzed. Finally, the extraction of rules in the form of deterministic finite state automata is investigated.

Keywords: recurrent neural networks, natural language processing, grammatical inference, government-and-binding theory, gradient descent, simulated annealing, principles-and-parameters framework, automata extraction.

* Also with the Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742.

1 Introduction

This paper considers the task of classifying natural language sentences as grammatical or ungrammatical. We attempt to train neural networks, without the bifurcation into learned vs. innate components assumed by Chomsky, to produce the same judgments as native speakers on sharply grammatical/ungrammatical data. Only recurrent neural networks are investigated for computational reasons. Computationally, recurrent neural networks are more powerful than feedforward networks and some recurrent architectures have been shown to be at least Turing equivalent [53, 54]. We investigate the properties of various popular recurrent neural network architectures, in particular Elman, Narendra & Parthasarathy (N&P) and Williams & Zipser (W&Z) recurrent networks, and also Frasconi-Gori-Soda (FGS) locally recurrent networks. We find that both Elman and W&Z recurrent neural networks are able to learn an appropriate grammar after implementing techniques for improving the convergence of the gradient descent based backpropagation-through-time training algorithm. We analyze the operation of the networks and investigate a rule approximation of what the recurrent network has learned – specifically, the extraction of rules in the form of deterministic finite state automata.

Previous work [38] has compared neural networks with other machine learning paradigms on this problem – this work focuses on recurrent neural networks, investigates additional networks, analyzes the operation of the networks and the training algorithm, and investigates rule extraction.

This paper is organized as follows: section 2 provides the motivation for the task attempted. Section 3 provides a brief introduction to formal grammars and grammatical inference and describes the data. Section 4 lists the recurrent neural network models investigated and provides details of the data encoding for the networks. Section 5 presents the results of investigation into various training heuristics, and investigation of training with simulated annealing. Section 6 presents the main results and simulation details, and investigates the operation of the networks. The extraction of rules in the form of deterministic finite state automata is investigated in section 7, and section 8 presents a discussion of the results and conclusions.

2 Motivation

2.1 Representational Power

Natural language has traditionally been handled using symbolic computation and recursive processes. The most successful stochastic language models have been based on finite-state descriptions such as *n*-grams or hidden Markov models. However, finite-state models cannot represent hierarchical structures as found in natural language[1] [48]. In the past few years several recurrent neural network architectures have emerged

[1]The inside-outside re-estimation algorithm is an extension of hidden Markov models intended to be useful for learning hierarchical systems. The algorithm is currently only practical for relatively small grammars [48].

which have been used for grammatical inference [9, 21, 19, 20, 68]. Recurrent neural networks have been used for several smaller natural language problems, e.g. papers using the Elman network for natural language tasks include: [1, 12, 24, 58, 59]. Neural network models have been shown to be able to account for a variety of phenomena in phonology [23, 61, 62, 18, 22], morphology [51, 41, 40] and role assignment [42, 58]. Induction of simpler grammars has been addressed often – e.g. [64, 65, 19] on learning Tomita languages [60]. The task considered here differs from these in that the grammar is more complex. The recurrent neural networks investigated in this paper constitute complex, dynamical systems – it has been shown that recurrent networks have the representational power required for hierarchical solutions [13], and that they are Turing equivalent.

2.2 Language and Its Acquisition

Certainly one of the most important questions for the study of human language is: How do people unfailingly manage to acquire such a complex rule system? A system so complex that it has resisted the efforts of linguists to date to adequately describe in a formal system [8]. A couple of examples of the kind of knowledge native speakers often take for granted are provided in this section.

For instance, any native speaker of English knows that the adjective *eager* obligatorily takes a complementizer *for* with a sentential complement that contains an overt subject, but that the verb *believe* cannot. Moreover, *eager* may take a sentential complement with a non-overt, i.e. an implied or understood, subject, but *believe* cannot[2]:

*I am eager John to be here	I believe John to be here
I am eager for John to be here	*I believe for John to be here
I am eager to be here	*I believe to be here

Such grammaticality judgments are sometimes subtle but unarguably form part of the native speaker's language competence. In other cases, judgment falls not on acceptability but on other aspects of language competence such as interpretation. Consider the reference of the embedded subject of the predicate *to talk to* in the following examples:

John is too stubborn for Mary to talk to
John is too stubborn to talk to
John is too stubborn to talk to Bill

In the first sentence, it is clear that *Mary* is the subject of the embedded predicate. As every native speaker knows, there is a strong contrast in the co-reference options for the understood subject in the second and

[2] As is conventional, an asterisk is used to indicate ungrammaticality.

third sentences despite their surface similarity. In the third sentence, *John* must be the implied subject of the predicate *to talk to*. By contrast, *John* is understood as the object of the predicate in the second sentence, the subject here having arbitrary reference; in other words, the sentence can be read as *John is too stubborn for some arbitrary person to talk to John*. The point to emphasize here is that the language faculty has impressive discriminatory power, in the sense that a single word, as seen in the examples above, can result in sharp differences in acceptability or alter the interpretation of a sentence considerably. Furthermore, the judgments shown above are robust in the sense that virtually all native speakers agree with the data.

In the light of such examples and the fact that such contrasts crop up not just in English but in other languages (for example, the *stubborn* contrast also holds in Dutch), some linguists (chiefly Chomsky [7]) have hypothesized that it is only reasonable that such knowledge is only partially acquired: the lack of variation found across speakers, and indeed, languages for certain classes of data suggests that there exists a fixed component of the language system. In other words, there is an innate component of the language faculty of the human mind that governs language processing. All languages obey these so-called universal principles. Since languages do differ with regard to things like subject-object-verb order, these principles are subject to parameters encoding systematic variations found in particular languages. Under the innateness hypothesis, only the language parameters plus the language-specific lexicon are acquired by the speaker; in particular, the principles are not learned. Based on these assumptions, the study of these language-independent principles has become known as the Principles-and-Parameters framework, or Government-and-Binding (GB) theory.

This paper investigates whether a neural network can be made to exhibit the same kind of discriminatory power on the sort of data GB-linguists have examined. More precisely, the goal is to train a neural network from scratch, i.e. without the division into learned vs. innate components assumed by Chomsky, to produce the same judgments as native speakers on the grammatical/ungrammatical pairs of the sort discussed above. Instead of using innate knowledge, positive and negative examples are used (a second argument for innateness is that it is not possible to learn the grammar without negative examples).

3 Data

We first provide a brief introduction to formal grammars, grammatical inference, and natural language; for a thorough introduction see Harrison [25] and Fu [17]. We then detail the dataset which we have used in our experiments.

3.1 Formal Grammars and Grammatical Inference

Briefly, a grammar G is a four tuple $\{N, T, P, S\}$, where N and T are sets of terminals and nonterminals comprising the alphabet of the grammar, P is a set of production rules, and S is the start symbol. For every

grammar there exists a language L, a set of strings of the terminal symbols, that the grammar generates or recognizes. There also exist automata that recognize and generate the grammar. Grammatical inference is concerned mainly with the procedures that can be used to infer the syntactic or production rules of an unknown grammar G based on a finite set of strings I from $L(G)$, the language generated by G, and possibly also on a finite set of strings from the complement of $L(G)$ [17]. This paper considers replacing the inference algorithm with a neural network and the grammar is that of the English language. The simple grammar used by Elman [13] shown in table 1 contains some of the structures in the complete English grammar: e.g. agreement, verb argument structure, interactions with relative clauses, and recursion.

S	$\rightarrow$	NP VP "."
NP	$\rightarrow$	PropN | N | N RC
VP	$\rightarrow$	V (NP)
RC	$\rightarrow$	who NP VP | who VP (NP)
N	$\rightarrow$	boy | girl | cat...
PropN	$\rightarrow$	John | Mary
V	$\rightarrow$	chase | feed | see...

Table 1. A simple grammar encompassing a subset of the English language (from [13]). NP = noun phrase, VP = verb phrase, PropN = proper noun, RC = relative clause, V = verb, N = noun, and S = the full sentence.

In the Chomsky hierarchy of phrase structured grammars, the simplest grammar and its associated automata are regular grammars and finite-state-automata (FSA). However, it has been firmly established [6] that the syntactic structures of natural language cannot be parsimoniously described by regular languages. Certain phenomena (e.g. center embedding) are more compactly described by context-free grammars which are recognized by push-down automata, while others (e.g. crossed-serial dependencies and agreement) are better described by context-sensitive grammars which are recognized by linear bounded automata [50].

3.2 Data

The data used in this work consists of 552 English positive and negative examples taken from an introductory GB-linguistics textbook by Lasnik and Uriagereka [37]. Most of these examples are organized into minimal pairs like the example *I am eager for John to win/*I am eager John to win* above. The minimal nature of the changes involved suggests that the dataset may represent an especially difficult task for the models. Due to the small sample size, the raw data, namely words, were first converted (using an existing parser) into the major syntactic categories assumed under GB-theory. Table 2 summarizes the parts of speech that were used.

The part-of-speech tagging represents the sole grammatical information supplied to the models about particular sentences in addition to the grammaticality status. An important refinement that was implemented

Category	Examples
Nouns (N)	*John*, *book* and *destruction*
Verbs (V)	*hit*, *be* and *sleep*
Adjectives (A)	*eager*, *old* and *happy*
Prepositions (P)	*without* and *from*
Complementizer (C)	*that* or *for* as in *I thought* that ... or *I am eager* for ...
Determiner (D)	*the* or *each* as in the *man* or each *man*
Adverb (Adv)	*sincerely* or *why* as in *I* sincerely *believe* ... or Why *did John want* ...
Marker (Mrkr)	possessive *'s*, *of*, or *to* as in *John*'s *mother*, *the destruction* of ..., or *I want* to *help* ...

Table 2. Parts of speech

was to include sub-categorization information for the major predicates, namely nouns, verbs, adjectives and prepositions. Experiments showed that adding sub-categorization to the bare category information improved the performance of the models. For example, an intransitive verb such as *sleep* would be placed into a different class from the obligatorily transitive verb *hit*. Similarly, verbs that take sentential complements or double objects such as *seem*, *give* or *persuade* would be representative of other classes[3]. Fleshing out the sub-categorization requirements along these lines for lexical items in the training set resulted in 9 classes for verbs, 4 for nouns and adjectives, and 2 for prepositions. Examples of the input data are shown in table 3.

Sentence	Encoding	Grammatical Status
I am eager for John to be here	n4 v2 a2 c n4 v2 adv	1
	n4 v2 a2 c n4 p1 v2 adv	1
I am eager John to be here	n4 v2 a2 n4 v2 adv	0
	n4 v2 a2 n4 p1 v2 adv	0
I am eager to be here	n4 v2 a2 v2 adv	1
	n4 v2 a2 p1 v2 adv	1

Table 3. Examples of the part-of-speech tagging

Tagging was done in a completely context-free manner. Obviously, a word, e.g. *to*, may be part of more than one part-of-speech. The tagging resulted in several contradictory and duplicated sentences. Various methods

[3]Following classical GB theory, these classes are synthesized from the theta-grids of individual predicates via the Canonical Structural Realization (CSR) mechanism of Pesetsky [49].

were tested to deal with these cases, however they were removed altogether for the results reported here. In addition, the number of positive and negative examples was equalized (by randomly removing examples from the higher frequency class) in all training and test sets in order to reduce any effects due to differing *a priori* class probabilities (when the number of samples per class varies between classes there may be a bias towards predicting the more common class [3, 2]).

4 Neural Network Models and Data Encoding

The following architectures were investigated. Architectures 1 to 3 are topological restrictions of 4 when the number of hidden nodes is equal and in this sense may not have the representational capability of model 4. It is expected that the Frasconi-Gori-Soda (FGS) architecture will be unable to perform the task and it has been included primarily as a control case.

1. *Frasconi-Gori-Soda (FGS) locally recurrent networks* [16]. A multilayer perceptron augmented with local feedback around each hidden node. The local-output version has been used. The FGS network has also been studied by [43] – the network is called FGS in this paper in line with [63].

2. *Narendra and Parthasarathy [44].* A recurrent network with feedback connections from each output node to all hidden nodes. The N&P network architecture has also been studied by Jordan [33, 34] – the network is called N&P in this paper in line with [30].

3. *Elman [13].* A recurrent network with feedback from each hidden node to all hidden nodes. When training the Elman network backpropagation-through-time is used rather than the truncated version used by Elman, i.e. in this paper "Elman network" refers to the architecture used by Elman but not the training algorithm.

4. *Williams and Zipser [67].* A recurrent network where all nodes are connected to all other nodes.

Diagrams of these architectures are shown in figures 1 to 4.

For input to the neural networks, the data was encoded into a fixed length window made up of segments containing eight separate inputs, corresponding to the classifications noun, verb, adjective, etc. Sub-categories of the classes were linearly encoded into each input in a manner demonstrated by the specific values for the noun input: Not a noun = 0, noun class 1 = 0.5, noun class 2 = 0.667, noun class 3 = 0.833, noun class 4 = 1. The linear order was defined according to the similarity between the various sub-categories[4]. Two outputs were used in the neural networks, corresponding to grammatical and ungrammatical classifications. The data was input to the neural networks with a window which is passed over the sentence in temporal

[4] A fixed length window made up of segments containing 23 separate inputs, corresponding to the classifications noun class 1, noun class 2, verb class 1, etc. was also tested but proved inferior.

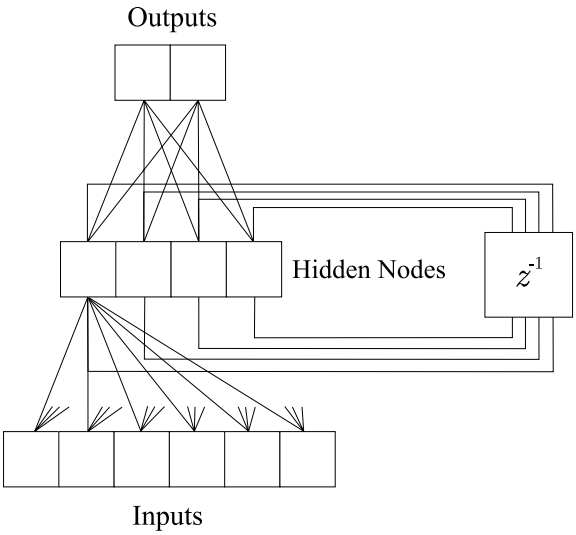

Figure 1. A Frasconi-Gori-Soda locally recurrent network. Not all connections are shown fully.

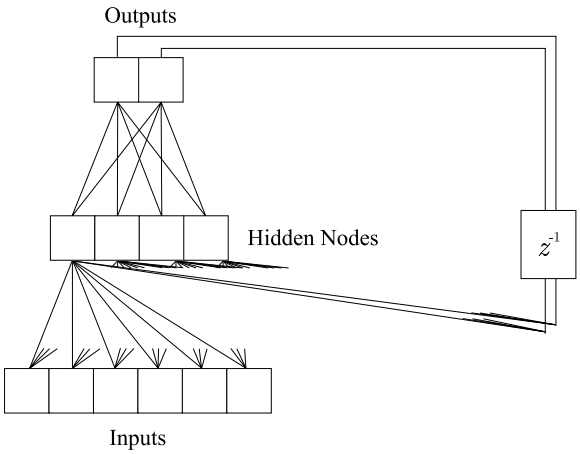

Figure 2. A Narendra & Parthasarathy recurrent network. Not all connections are shown fully.

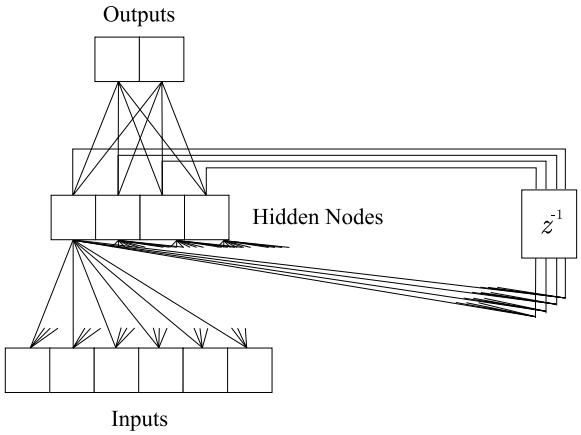

Figure 3. An Elman recurrent network. Not all connections are shown fully.

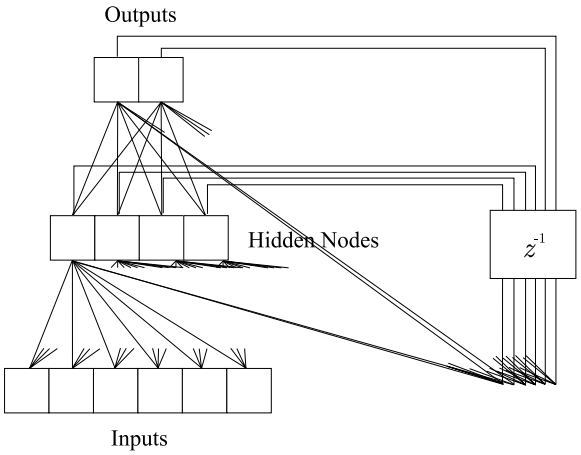

Figure 4. A Williams & Zipser fully recurrent network. Not all connections are shown fully.

order from the beginning to the end of the sentence (see figure 5). The size of the window was variable from one word to the length of the longest sentence. We note that the case where the input window is small is of greater interest – the larger the input window, the greater the capability of a network to correctly classify the training data without forming a grammar. For example, if the input window is equal to the longest sentence, then the network does not have to store any information – it can simply map the inputs directly to the classification. However, if the input window is relatively small, then the network must learn to store information. As will be shown later these networks implement a grammar, and a deterministic finite state automaton which recognizes this grammar can be extracted from the network. Thus, we are most interested in the small input window case, where the networks are required to form a grammar in order to perform well.

5 Gradient Descent and Simulated Annealing Learning

Backpropagation-through-time [66][5] has been used to train the globally recurrent networks[6], and the gradient descent algorithm described by the authors [16] was used for the FGS network. The standard gradient descent algorithms were found to be impractical for this problem[7]. The techniques described below for improving convergence were investigated. Due to the dependence on the initial parameters, a number of simulations were performed with different initial weights and training set/test set combinations. However, due to the computational complexity of the task[8], it was not possible to perform as many simulations as

[5]Backpropagation-through-time extends backpropagation to include temporal aspects and arbitrary connection topologies by considering an equivalent feedforward network created by unfolding the recurrent network in time.

[6]Real-time [67] recurrent learning (RTRL) was also tested but did not show any significant convergence for the present problem.

[7]Without modifying the standard gradient descent algorithms it was only possible to train networks which operated on a large temporal input window. These networks were not forced to model the grammar, they only memorized and interpolated between the training data.

[8]Each individual simulation in this section took an average of two hours to complete on a Sun Sparc 10 server.

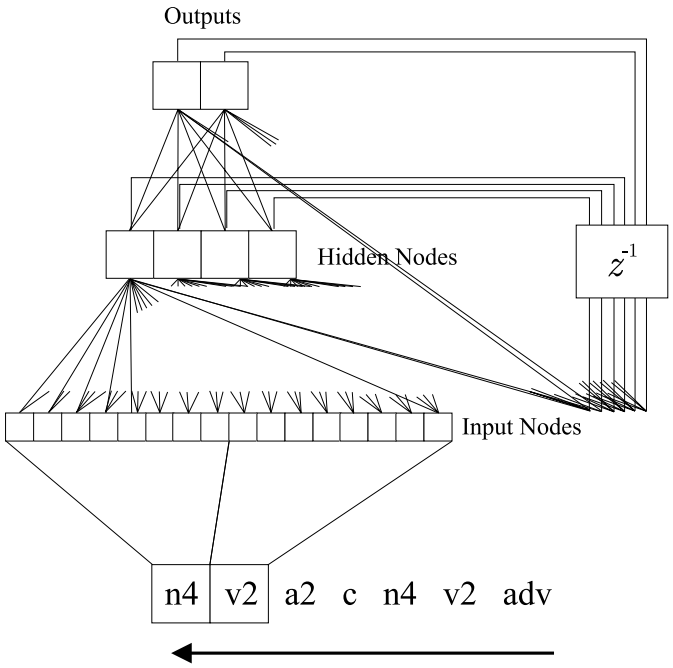

Figure 5. Depiction of how the neural network inputs come from an input window on the sentence. The window moves from the beginning to the end of the sentence.

desired. The standard deviation of the NMSE values is included to help assess the significance of the results. Table 4 shows some results for using and not using the techniques listed below. Except where noted, the results in this section are for Elman networks using: two word inputs, 10 hidden nodes, the quadratic cost function, the logistic sigmoid function, sigmoid output activations, one hidden layer, the learning rate schedule shown below, an initial learning rate of 0.2, the weight initialization strategy discussed below, and 1 million stochastic updates (target values are only provided at the end of each sentence).

Standard	NMSE	Std. Dev.	Variation	NMSE	Std. Dev.
Update: batch	0.931	0.0036	**Update: stochastic**	0.366	0.035
Learning rate: constant	0.742	0.154	**Learning rate: schedule**	0.394	0.035
Activation: logistic	0.387	0.023	Activation: tanh	0.405	0.14
Sectioning: no	0.367	0.011	Sectioning: yes	0.573	0.051
Cost function: quadratic	0.470	0.078	Cost function: entropy	0.651	0.0046

Table 4. Comparisons of using and not using various convergence techniques. All other parameters are constant in each case: Elman networks using: two word inputs (i.e. a sliding window of the current and previous word), 10 hidden nodes, the quadratic cost function, the logistic activation function, sigmoid output activations, one hidden layer, a learning rate schedule with an initial learning rate of 0.2, the weight initialization strategy discussed below, and 1 million stochastic updates. Each NMSE result represents the average of four simulations. The standard deviation value given is the standard deviation of the four individual results.

1. *Detection of Significant Error Increases.* If the NMSE increases significantly during training then network weights are restored from a previous epoch and are perturbed to prevent updating to the same point. This technique was found to increase robustness of the algorithm when using learning rates large enough to help avoid problems due to local minima and "flat spots" on the error surface, particularly in the case of the Williams & Zipser network.

2. *Target Outputs.* Targets outputs were 0.1 and 0.9 using the logistic activation function and -0.8 and 0.8 using the `tanh` activation function. This helps avoid saturating the sigmoid function. If targets were set to the asymptotes of the sigmoid this would tend to: a) drive the weights to infinity, b) cause outlier data to produce very large gradients due to the large weights, and c) produce binary outputs even when incorrect – leading to decreased reliability of the confidence measure.

3. *Stochastic Versus Batch Update.* In stochastic update, parameters are updated after each pattern presentation, whereas in true gradient descent (often called "batch" updating) gradients are accumulated over the complete training set. Batch update attempts to follow the true gradient, whereas a stochastic path is followed using stochastic update.

   Stochastic update is often much quicker than batch update, especially with large, redundant datasets [39]. Additionally, the stochastic path may help the network to escape from local minima. However, the error can jump around without converging unless the learning rate is reduced, most second order methods do not work well with stochastic update, and stochastic update is harder to parallelize than batch [39]. Batch update provides guaranteed convergence (to local minima) and works better with second order techniques. However it can be very slow, and may converge to very poor local minima.

   In the results reported, the training times were equalized by reducing the number of updates for the batch case (for an equal number of weight updates batch update would otherwise be much slower). Batch update often converges quicker using a higher learning rate than the optimal rate used for stochastic update[9], hence altering the learning rate for the batch case was investigated. However, significant convergence was not obtained as shown in table 4.

4. *Weight Initialization.* Random weights are initialized with the goal of ensuring that the sigmoids do not start out in saturation but are not very small (corresponding to a flat part of the error surface) [26]. In addition, several (20) sets of random weights are tested and the set which provides the best performance on the training data is chosen. In our experiments on the current problem, it was found that these techniques do not make a significant difference.

5. *Learning Rate Schedules.* Relatively high learning rates are typically used in order to help avoid slow convergence and local minima. However, a constant learning rate results in significant parameter and performance fluctuation during the entire training cycle such that the performance of the network can

[9] Stochastic update does not generally tolerate as high a learning rate as batch update due to the stochastic nature of the updates.

alter significantly from the beginning to the end of the final epoch. Moody and Darken have proposed "search then converge" learning rate schedules of the form [10, 11]:

$$\eta(t) = \frac{\eta_0}{1 + \frac{t}{\tau}} \tag{1}$$

where $\eta(t)$ is the learning rate at time t, η_0 is the initial learning rate, and τ is a constant.

We have found that the learning rate during the final epoch still results in considerable parameter fluctuation,[10] and hence we have added an additional term to further reduce the learning rate over the final epochs (our specific learning rate schedule can be found in a later section). We have found the use of learning rate schedules to improve performance considerably as shown in table 4.

6. *Activation Function.* Symmetric sigmoid functions (e.g. $\tanh$) often improve convergence over the standard logistic function. For our particular problem we found that the difference was minor and that the logistic function resulted in better performance as shown in table 4.

7. *Cost Function.* The relative entropy cost function [4, 29, 57, 26, 27] has received particular attention and has a natural interpretation in terms of learning probabilities [36]. We investigated using both quadratic and relative entropy cost functions:

   Definition 1 The quadratic cost function is defined as

   $$E = \frac{1}{2}\sum_k (y_k - d_k)^2 \tag{2}$$

   □

   Definition 2 The relative entropy cost function is defined as

   $$E = \sum_k \left[\frac{1}{2}(1 + y_k)\log\frac{1 + y_k}{1 + d_k} + \frac{1}{2}(1 - y_k)\log\frac{1 - y_k}{1 - d_k}\right] \tag{3}$$

   □

   where y and d correspond to the actual and desired output values, k ranges over the outputs (and also the patterns for batch update). We found the quadratic cost function to provide better performance as shown in table 4. A possible reason for this is that the use of the entropy cost function leads to an increased variance of weight updates and therefore decreased robustness in parameter updating.

8. *Sectioning of the Training Data.* We investigated dividing the training data into subsets. Initially, only one of these subsets was used for training. After 100% correct classification was obtained or a pre-specified time limit expired, an additional subset was added to the "working" set. This continued until the working set contained the entire training set. The data was ordered in terms of sentence length with

[10] NMSE results which are obtained over an epoch involving stochastic update can be misleading. We have been surprised to find quite significant difference in these on-line NMSE calculations compared to a static calculation even if the algorithm appears to have converged.

the shortest sentences first. This enabled the networks to focus on the simpler data first. Elman suggests that the initial training constrains later training in a useful way [13]. However, for our problem, the use of sectioning has consistently decreased performance as shown in table 4.

We have also investigated the use of simulated annealing. Simulated annealing is a global optimization method [32, 35]. When minimizing a function, any downhill step is accepted and the process repeats from this new point. An uphill step may also be accepted. It is therefore possible to escape from local minima. As the optimization process proceeds, the length of the steps declines and the algorithm converges on the global optimum. Simulated annealing makes very few assumptions regarding the function to be optimized, and is therefore quite robust with respect to non-quadratic error surfaces.

Previous work has shown the use of simulated annealing for finding the parameters of a recurrent network model to improve performance [56]. For comparison with the gradient descent based algorithms the use of simulated annealing has been investigated in order to train exactly the same Elman network as has been successfully trained to 100% correct training set classification using backpropagation-through-time (details are in section 6). No significant results were obtained from these trials[11]. The use of simulated annealing has not been found to improve performance as in Simard et al. [56]. However, their problem was the parity problem using networks with only four hidden units whereas the networks considered in this paper have many more parameters.

This result provides an interesting comparison to the gradient descent backpropagation-through-time (BPTT) method. BPTT makes the implicit assumption that the error surface is amenable to gradient descent optimization, and this assumption can be a major problem in practice. However, although difficulty is encountered with BPTT, the method is significantly more successful than simulated annealing (which makes few assumptions) for this problem.

6 Experimental Results

Results for the four neural network architectures are given in this section. The results are based on multiple training/test set partitions and multiple random seeds. In addition, a set of Japanese control data was used as a test set (we do not consider training models with the Japanese data because we do not have a large enough dataset). Japanese and English are at the opposite ends of the spectrum with regard to word order. That is, Japanese sentence patterns are very different from English. In particular, Japanese sentences are typically SOV (subject-object-verb) with the verb more or less fixed and the other arguments more or less available to freely permute. English data is of course SVO and argument permutation is generally not available. For example, the canonical Japanese word order is simply ungrammatical in English. Hence, it would be extremely surprising if an English-trained model accepts Japanese, i.e. it is expected that a network trained

[11] The adaptive simulated annealing code by Lester Ingber [31, 32] was used.

on English will not generalize to Japanese data. This is what we find – all models resulted in no significant generalization on the Japanese data (50% error on average).

Five simulations were performed for each architecture. Each simulation took approximately four hours on a Sun Sparc 10 server. Table 5 summarizes the results obtained with the various networks. In order to make the number of weights in each architecture approximately equal we have used only single word inputs for the W&Z model but two word inputs for the others. This reduction in dimensionality for the W&Z network improved performance. The networks contained 20 hidden units. Full simulation details are given in section 6.

The goal was to train a network using only a small temporal input window. Initially, this could not be done. With the addition of the techniques described earlier it was possible to train Elman networks with sequences of the last two words as input to give 100% correct (99.6% averaged over 5 trials) classification on the training data. Generalization on the test data resulted in 74.2% correct classification on average. This is better than the performance obtained using any of the other networks, however it is still quite low. The data is quite sparse and it is expected that increased generalization performance will be obtained as the amount of data is increased, as well as increased difficulty in training. Additionally, the dataset has been hand-designed by GB linguists to cover a range of grammatical structures and it is likely that the separation into the training and test sets creates a test set which contains many grammatical structures that are not covered in the training set. The Williams & Zipser network also performed reasonably well with 71.3% correct classification of the test set. Note that the test set performance was not observed to drop significantly after extended training, indicating that the use of a validation set to control possible overfitting would not alter performance significantly.

TRAIN	Classification	Std. dev.
Elman	**99.6%**	0.84
FGS	67.1%	1.22
N&P	75.2%	1.41
W&Z	**91.7%**	2.26

ENGLISH TEST	Classification	Std. dev.
Elman	**74.2%**	3.82
FGS	59.0%	1.52
N&P	60.6%	0.97
W&Z	**71.3%**	0.75

Table 5. Results of the network architecture comparison. The classification values reported are an average of five individual simulations, and the standard deviation value is the standard deviation of the five individual results.

Complete details on a sample Elman network are as follows (other networks differ only in topology except

for the W&Z for which better results were obtained using an input window of only one word): the network contained three layers including the input layer. The hidden layer contained 20 nodes. Each hidden layer node had a recurrent connection to all other hidden layer nodes. The network was trained for a total of 1 million stochastic updates. All inputs were within the range zero to one. All target outputs were either 0.1 or 0.9. Bias inputs were used. The best of 20 random weight sets was chosen based on training set performance. Weights were initialized as shown in Haykin [26] where weights are initialized on a node by node basis as uniformly distributed random numbers in the range $(-2.4/F_i, 2.4/F_i)$ where F_i is the fan-in of neuron i. The logistic output activation function was used. The quadratic cost function was used. The search then converge learning rate schedule used was $\eta = \frac{\eta_0}{\frac{n}{N/2} + \frac{c_1}{\max\left(1, (c_1 - \frac{\max(0, c_1(n - c_2 N))}{(1-c_2)N}\right)}}$ where η = learning rate, η_0 = initial learning rate, N = total training epochs, n = current training epoch, $c_1 = 50$, $c_2 = 0.65$. The training set consisted of 373 non-contradictory examples as described earlier. The English test set consisted of 100 non-contradictory samples and the Japanese test set consisted of 119 non-contradictory samples.

We now take a closer look at the operation of the networks. The error during training for a sample of each network architecture is shown in figure 6. The error at each point in the graphs is the NMSE over the complete training set. Note the nature of the Williams & Zipser learning curve and the utility of detecting and correcting for significant error increases[12].

Figure 7 shows an approximation of the "complexity" of the error surface based on the first derivatives of the error criterion with respect to each weight $C = \frac{\sum_i \frac{\partial E}{\partial w_i}}{N_w}$ where i sums over all weights in the network and N_w is the total number of weights. This value has been plotted after each epoch during training. Note the more complex nature of the plot for the Williams & Zipser network.

Figures 8 to 11 show sample plots of the error surface for the various networks. The error surface has many dimensions making visualization difficult. We plot sample views showing the variation of the error for two dimensions, and note that these plots are indicative only – quantitative conclusions should be drawn from the test error and not from these plots. Each plot shown in the figures is with respect to only two randomly chosen dimensions. In each case, the center of the plot corresponds to the values of the parameters after training. Taken together, the plots provide an approximate indication of the nature of the error surface for the different network types. The FGS network error surface appears to be the smoothest, however the results indicate that the solutions found do not perform very well, indicating that the minima found are poor compared to the global optimum and/or that the network is not capable of implementing a mapping with low error. The Williams & Zipser fully connected network has greater representational capability than the Elman architecture (in the sense that it can perform a greater variety of computations with the same number of hidden units). However, comparing the Elman and W&Z network error surface plots, it can be observed that the W&Z network has a greater percentage of flat spots. These graphs are not conclusive (because they only show two dimensions and are only plotted around one point in the weight space), however they back up

[12]The learning curve for the Williams & Zipser network can be made smoother by reducing the learning rate but this tends to promote convergence to poorer local minima.

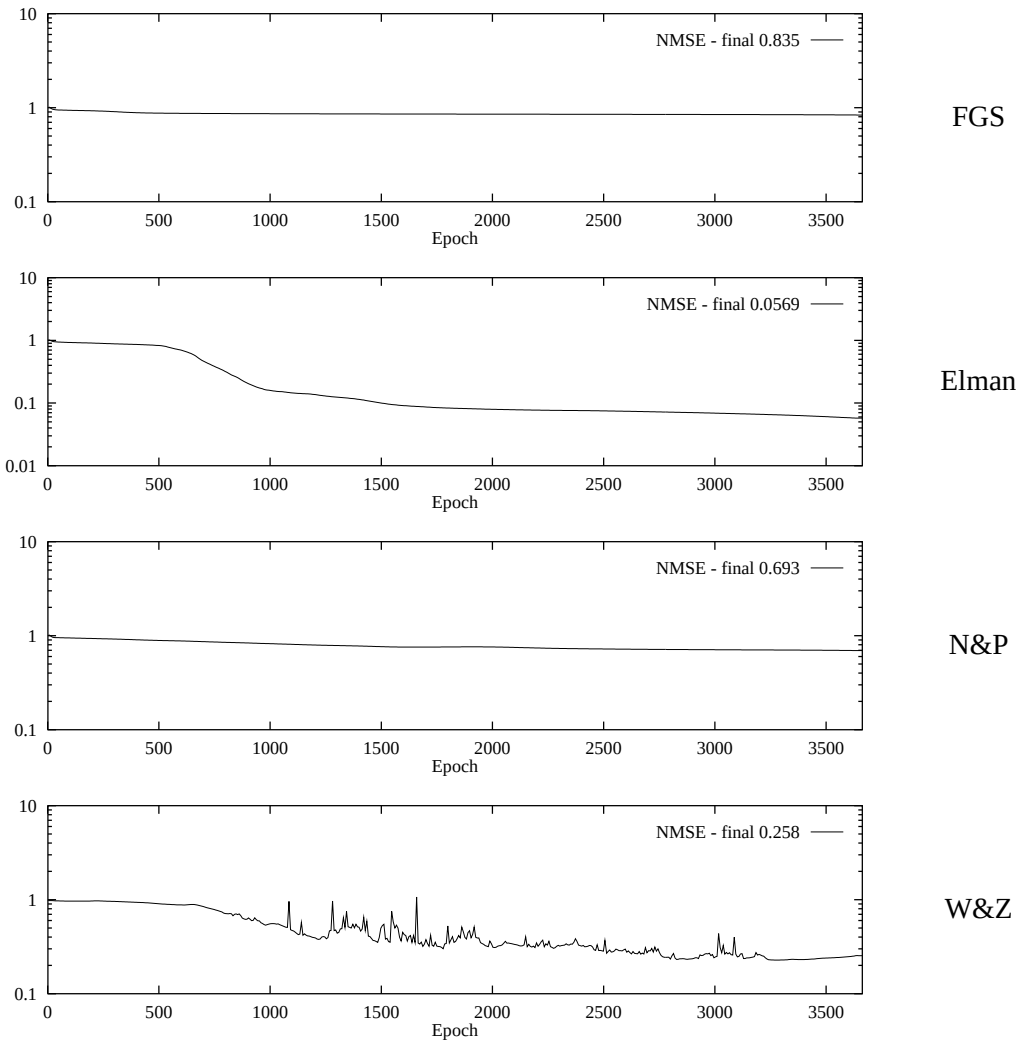

Figure 6. Average NMSE (log scale) over the training set during training. Top to bottom: Frasconi-Gori-Soda, Elman, Narendra & Parthasarathy and Williams & Zipser.

the hypothesis that the W&Z network performs worse because the error surface presents greater difficulty to the training method.

7 Automata Extraction

The extraction of symbolic knowledge from trained neural networks allows the exchange of information between connectionist and symbolic knowledge representations and has been of great interest for understanding what the neural network is actually doing [52]. In addition symbolic knowledge can be inserted into recurrent neural networks and even refined after training [15, 47, 45].

The ordered triple of a discrete Markov process ($\{$state; input $\rightarrow$ next-state$\}$) can be extracted from a RNN

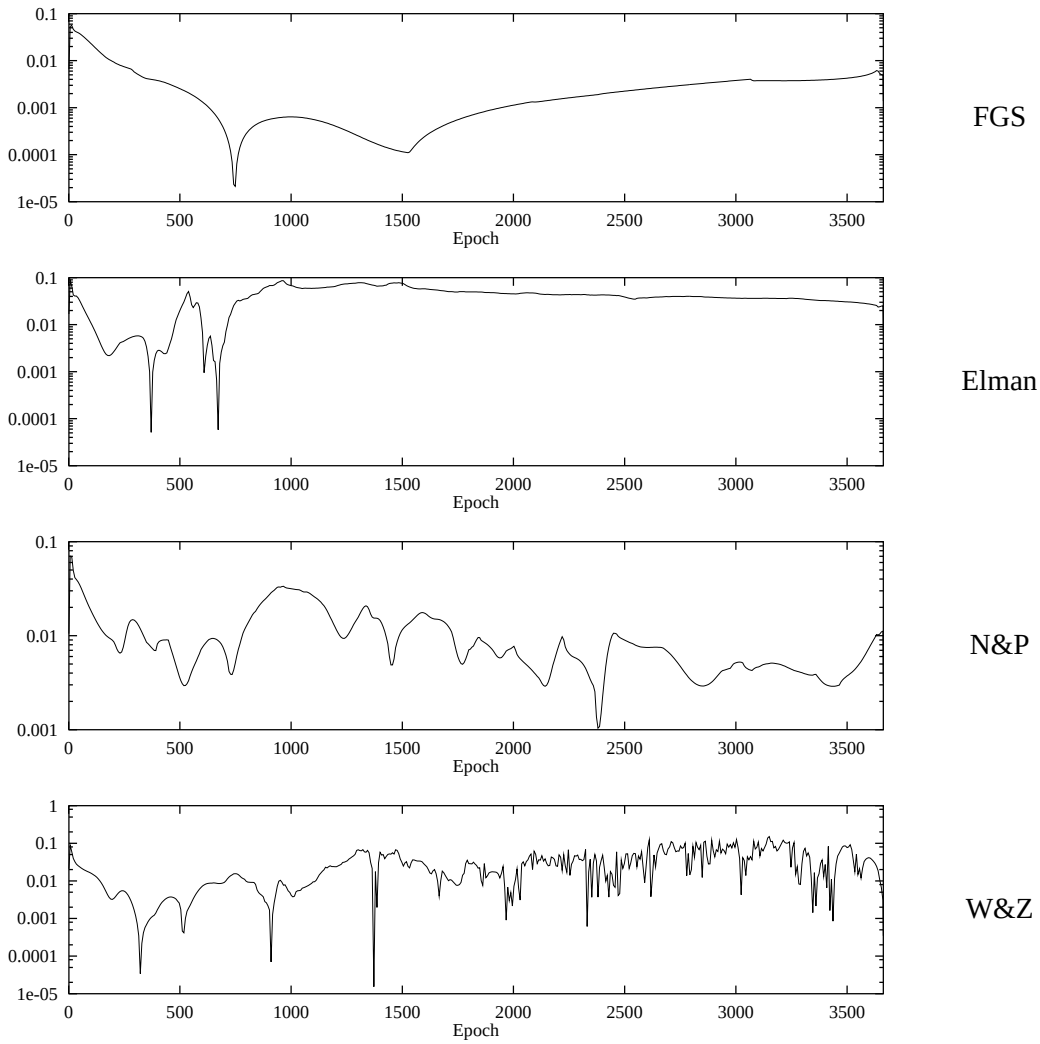

Figure 7. Approximate "complexity" of the error surface during training. Top to bottom: Frasconi-Gori-Soda, Elman, Narendra & Parthasarathy and Williams & Zipser.

and used to form an equivalent deterministic finite state automata (DFA). This can be done by clustering the activation values of the recurrent state neurons [46]. The automata extracted with this process can only recognize regular grammars[13].

However, natural language [6] cannot be parsimoniously described by regular languages - certain phenomena (e.g. center embedding) are more compactly described by context-free grammars, while others (e.g. crossed-serial dependencies and agreement) are better described by context-sensitive grammars. Hence, the networks may be implementing more parsimonious versions of the grammar which we are unable to extract with this technique.

[13]A regular grammar G is a 4-tuple $G = \{S, N, T, P\}$ where S is the start symbol, N and T are non-terminal and terminal symbols, respectively, and P represents productions of the form $A \rightarrow a$ or $A \rightarrow aB$ where $A, B \in N$ and $a \in T$.

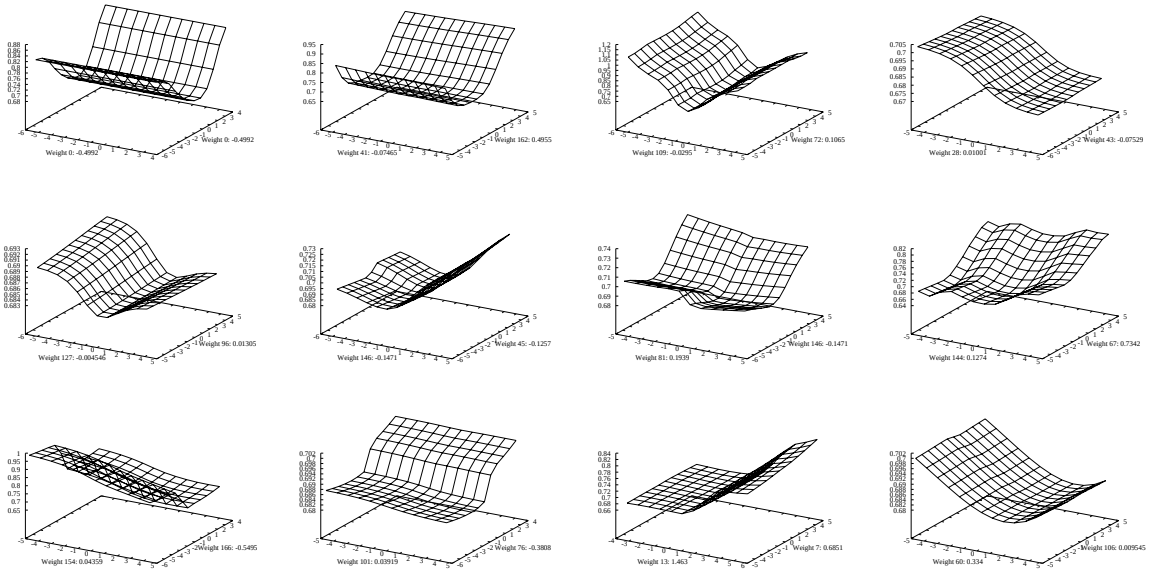

Figure 8. Error surface plots for the FGS network. Each plot is with respect to two randomly chosen dimensions. In each case, the center of the plot corresponds to the values of the parameters after training.

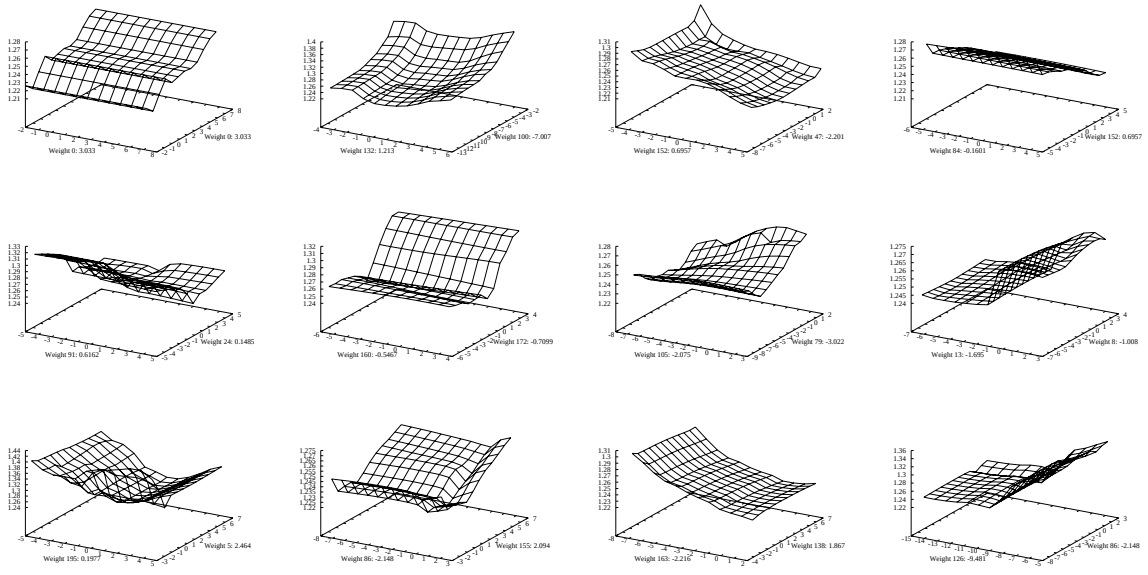

Figure 9. Error surface plots for the N&P network. Each plot is with respect to two randomly chosen dimensions. In each case, the center of the plot corresponds to the values of the parameters after training.

The algorithm we use for automata extraction (from [19]) works as follows: after the network is trained (or even during training), we apply a procedure for extracting what the network has learned — *i.e.*, the network's current conception of what DFA it has learned. The DFA extraction process includes the following steps: 1) clustering of the recurrent network activation space, $\mathbf{S}$, to form DFA states, 2) constructing a transition

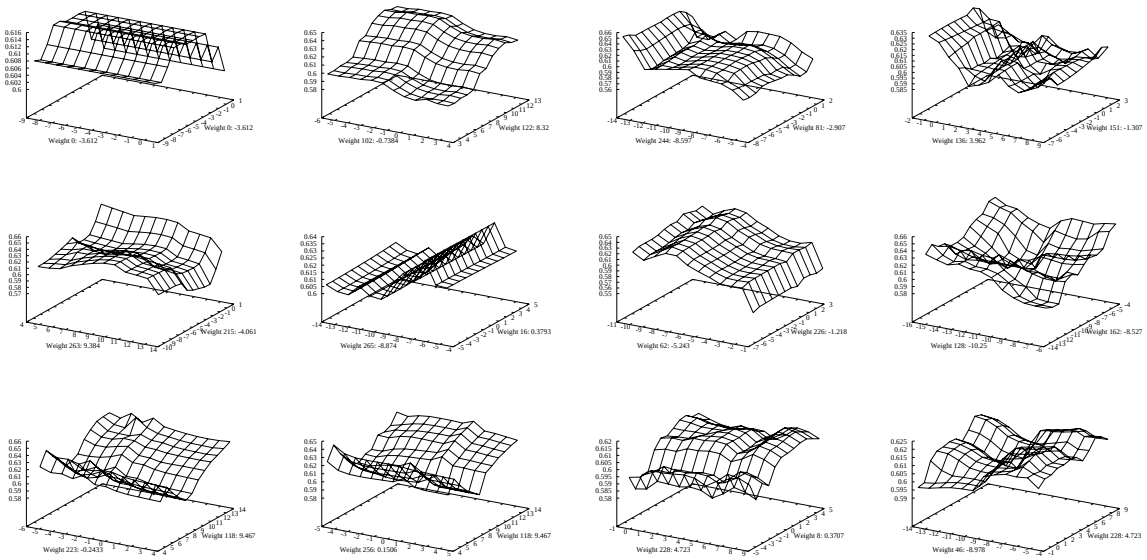

Figure 10. Error surface plots for the Elman network. Each plot is with respect to two randomly chosen dimensions. In each case, the center of the plot corresponds to the values of the parameters after training.

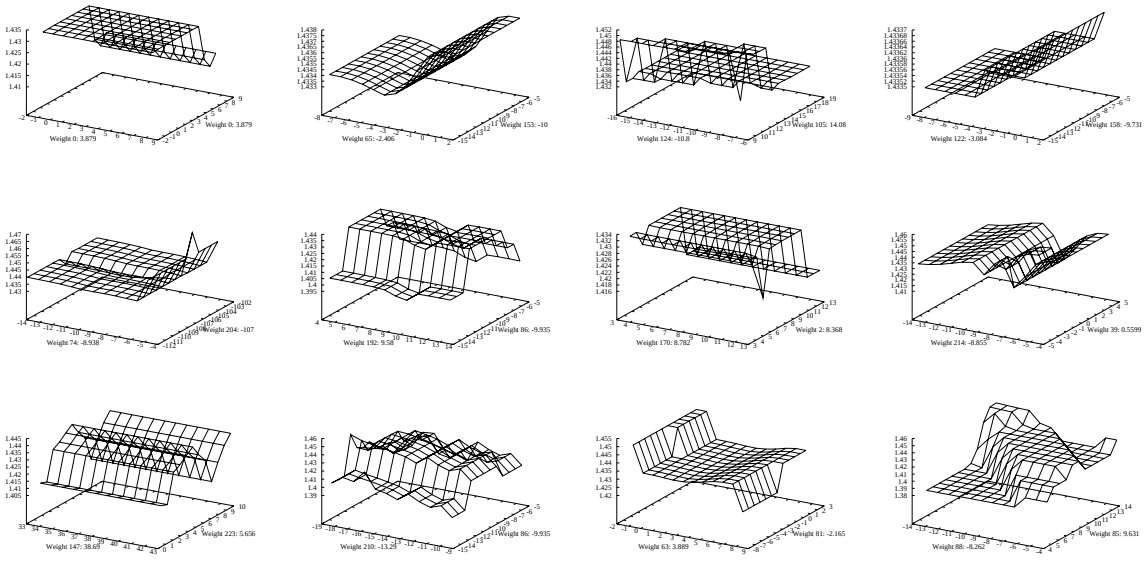

Figure 11. Error surface plots for the W&Z network. Each plot is with respect to two randomly chosen dimensions. In each case, the center of the plot corresponds to the values of the parameters after training.

diagram by connecting these states together with the alphabet labelled arcs, 3) putting these transitions together to make the full digraph – forming loops, and 4) reducing the digraph to a minimal representation. The hypothesis is that during training, the network begins to partition (or quantize) its state space into fairly well-separated, distinct regions or clusters, which represent corresponding states in some finite state

automaton (recently, it has been proved that arbitrary DFAs can be stably encoded into recurrent neural networks [45]). One simple way of finding these clusters is to divide each neuron's range into q partitions of equal width. Thus for N hidden neurons, there exist q^N possible *partition states*. The DFA is constructed by generating a state transition diagram, i.e. associating an input symbol with the *partition state* it just left and the *partition state* it activates. The initial *partition state*, or start state of the DFA, is determined from the initial value of $\mathbf{S}^{(t=0)}$. If the next input symbol maps to the same *partition state* value, we assume that a loop is formed. Otherwise, a new state in the DFA is formed. The DFA thus constructed may contain a maximum of q^N states; in practice it is usually much less, since not all *partition states* are reached by $\mathbf{S}^{(t)}$. Eventually this process must terminate since there are only a finite number of partitions available; and, in practice, many of the partitions are never reached. The *derived* DFA can then be reduced to its minimal DFA using standard minimization algorithms [28]. It should be noted that this DFA extraction method may be applied to any discrete-time recurrent net, regardless of order or hidden layers. Recently, the extraction process has been proven to converge and extract any DFA learned or encoded by the neural network [5].

The extracted DFAs depend on the quantization level, q. We extracted DFAs using values of q starting from 3 and used standard minimization techniques to compare the resulting automata [28]. We passed the training and test data sets through the extracted DFAs. We found that the extracted automata correctly classified 95% of the training data and 60% of the test data for $q = 7$. Smaller values of q produced DFAs with lower performance and larger values of q did not produce significantly better performance. A sample extracted automata can be seen in figure 12. It is difficult to interpret the extracted automata and a topic for future research is analysis of the extracted automata with a view to aiding interpretation. Additionally, an important open question is how well the extracted automata approximate the grammar implemented by the recurrent network which may not be a regular grammar.

Automata extraction may also be useful for improving the performance of the system via an iterative combination of rule extraction and rule insertion. Significant learning time improvements can be achieved by training networks with prior knowledge [46]. This may lead to the ability to train larger networks which encompass more of the target grammar.

8 Discussion

This paper has investigated the use of various recurrent neural network architectures (FGS, N&P, Elman and W&Z) for classifying natural language sentences as grammatical or ungrammatical, thereby exhibiting the same kind of discriminatory power provided by the Principles and Parameters linguistic framework, or Government-and-Binding theory. From best to worst performance, the architectures were: Elman, W&Z, N&P and FGS. It is not surprising that the Elman network outperforms the FGS and N&P networks. The computational power of Elman networks has been shown to be at least Turing equivalent [55], where the N&P networks have been shown to be Turing equivalent [54] but to within a linear slowdown. FGS networks

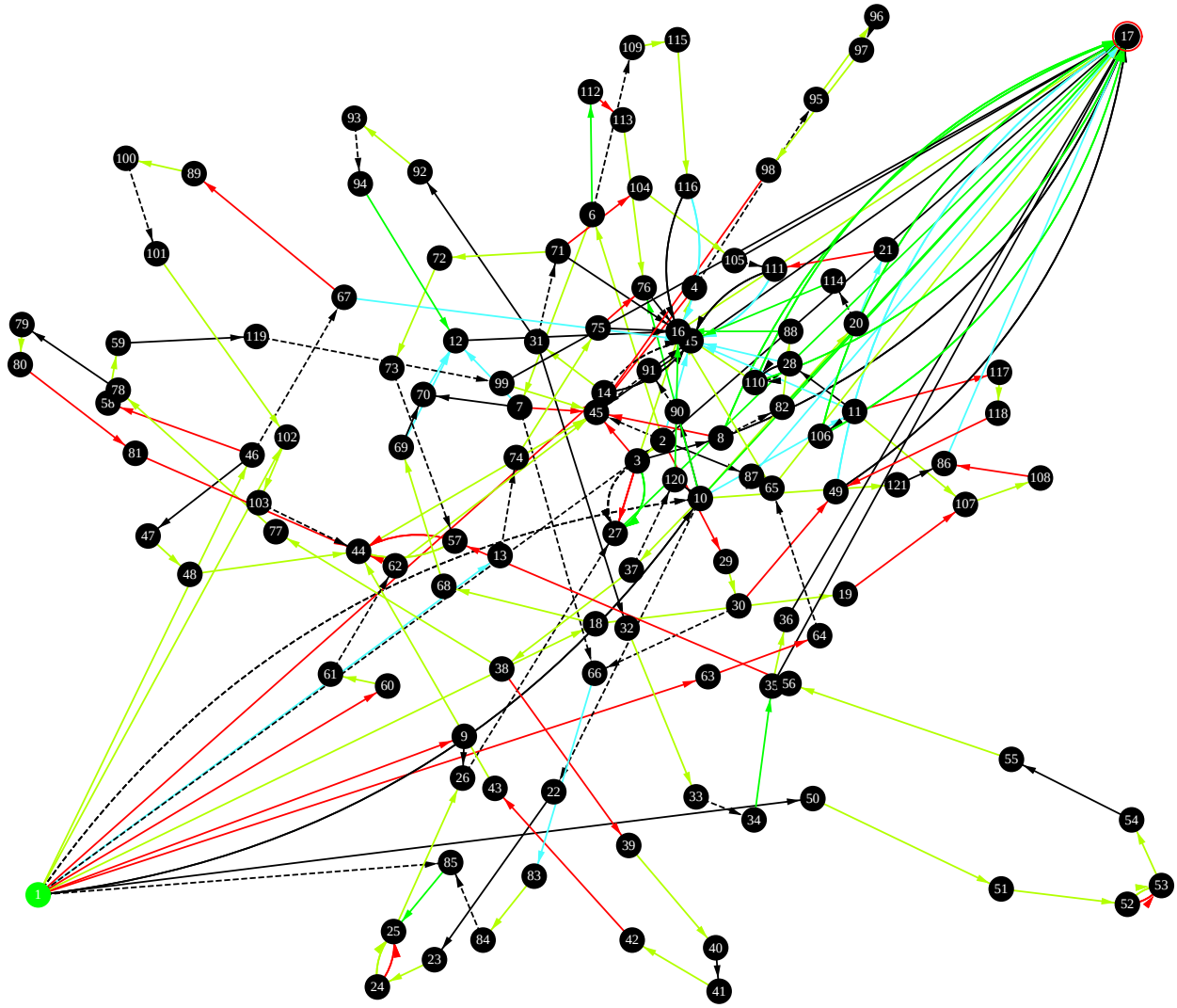

Figure 12. An automata extracted from an Elman network trained to perform the natural language task. The start state is state 1 at the bottom left and the accepting state is state 17 at the top right. All strings which do not reach the accepting state are rejected.

have recently been shown to be the most computationally limited [14]. Elman networks are just a special case of W&Z networks – the fact that the Elman and W&Z networks are the top performers is not surprising. However, theoretically why the Elman network outperformed the W&Z network is an open question. Our experimental results suggest that this is a training issue and not a representational issue. Backpropagation-through-time (BPTT) is an iterative algorithm that is not guaranteed to find the global minima of the cost function error surface. The error surface is different for the Elman and W&Z networks, and our results suggest that the error surface of the W&Z network is less suitable for the BPTT training algorithm used. However, all architectures do learn some representation of the grammar.

Are the networks learning the grammar? The hierarchy of architectures with increasing computational power (for a given number of hidden nodes) give an insight into whether the increased power is used to model the

more complex structures found in the grammar. The fact that the more powerful Elman and W&Z networks provided increased performance suggests that they were able to find structure in the data which it may not be possible to model with the FGS network. Additionally, investigation of the data suggests that 100% correct classification on the training data with only two word inputs would not be possible unless the networks were able to learn significant aspects of the grammar.

Another comparison of recurrent neural network architectures, that of Giles and Horne [30], compared various networks on randomly generated 6 and 64-state finite memory machines. The locally recurrent and Narendra & Parthasarathy networks proved as good or superior to more powerful networks like the Elman network, indicating that either the task did not require the increased power, or the vanilla backpropagation-through-time learning algorithm used was unable to exploit it.

This paper has shown that both Elman and W&Z recurrent neural networks are able to learn an appropriate grammar for discriminating between the sharply grammatical/ungrammatical pairs used by GB-linguists. However, generalization is limited by the amount and nature of the data available, and it is expected that increased difficulty will be encountered in training the models as more data is used. It is clear that there is considerable difficulty scaling the models considered here up to larger problems. We need to continue to address the convergence of the training algorithms, and believe that further improvement is possible by addressing the nature of parameter updating during gradient descent. However, a point must be reached after which improvement with gradient descent based algorithms requires consideration of the nature of the error surface. This is related to the input and output encodings (rarely are they chosen with the specific aim of controlling the error surface), the ability of parameter updates to modify network behavior without destroying previously learned information, and the method by which the network implements structures such as hierarchical and recursive relations.

Acknowledgments

This work has been partially supported by the Australian Telecommunications and Electronics Research Board (SL).

References

[1] Robert B. Allen. Sequential connectionist networks for answering simple questions about a microworld. In *5th Annual Proceedings of the Cognitive Science Society*, pages 489–495, 1983.

[2] Etienne Barnard and Elizabeth C. Botha. Back-propagation uses prior information efficiently. *IEEE Transactions on Neural Networks*, 4(5):794–802, September 1993.

[3] Etienne Barnard and David Casasent. A comparison between criterion functions for linear classifiers, with an application to neural nets. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(5):1030–1041, 1989.

[4] E.B. Baum and F. Wilczek. Supervised learning of probability distributions by neural networks. In D.Z. Anderson, editor, *Neural Information Processing Systems*, pages 52–61, New York, 1988. American Institute of Physics.

[5] M.P. Casey. The dynamics of discrete-time computation, with application to recurrent neural networks and finite state machine extraction. *Neural Computation*, 8(6):1135–1178, 1996.

[6] N.A. Chomsky. Three models for the description of language. *IRE Transactions on Information Theory*, IT-2:113–124, 1956.

[7] N.A. Chomsky. *Lectures on Government and Binding*. Foris Publications, 1981.

[8] N.A. Chomsky. *Knowledge of Language: Its Nature, Origin, and Use*. Prager, 1986.

[9] A. Cleeremans, D. Servan-Schreiber, and J.L. McClelland. Finite state automata and simple recurrent networks. *Neural Computation*, 1(3):372–381, 1989.

[10] C. Darken and J.E. Moody. Note on learning rate schedules for stochastic optimization. In R.P. Lippmann, J.E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems*, volume 3, pages 832–838. Morgan Kaufmann, San Mateo, CA, 1991.

[11] C. Darken and J.E. Moody. Towards faster stochastic gradient search. In *Neural Information Processing Systems 4*, pages 1009–1016. Morgan Kaufmann, 1992.

[12] J.L. Elman. Structured representations and connectionist models. In *6th Annual Proceedings of the Cognitive Science Society*, pages 17–25, 1984.

[13] J.L. Elman. Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning*, 7(2/3):195–226, 1991.

[14] P. Frasconi and M. Gori. Computational capabilities of local-feedback recurrent networks acting as finite-state machines. *IEEE Transactions on Neural Networks*, 7(6), 1996. 1521-1524.

[15] P. Frasconi, M. Gori, M. Maggini, and G. Soda. Unified integration of explicit rules and learning by example in recurrent networks. *IEEE Transactions on Knowledge and Data Engineering*, 7(2):340–346, 1995.

[16] P. Frasconi, M. Gori, and G. Soda. Local feedback multilayered networks. *Neural Computation*, 4(1):120–130, 1992.

[17] K.S. Fu. *Syntactic Pattern Recognition and Applications*. Prentice-Hall, Englewood Cliffs, N.J, 1982.

[18] M. Gasser and C. Lee. Networks that learn phonology. Technical report, Computer Science Department, Indiana University, 1990.

[19] C. Lee Giles, C.B. Miller, D. Chen, H.H. Chen, G.Z. Sun, and Y.C. Lee. Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation*, 4(3):393–405, 1992.

[20] C. Lee Giles, C.B. Miller, D. Chen, G.Z. Sun, H.H. Chen, and Y.C. Lee. Extracting and learning an unknown grammar with recurrent neural networks. In J.E. Moody, S.J. Hanson, and R.P Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 317–324, San Mateo, CA, 1992. Morgan Kaufmann Publishers.

[21] C. Lee Giles, G.Z. Sun, H.H. Chen, Y.C. Lee, and D. Chen. Higher order recurrent networks & grammatical inference. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 380–387, San Mateo, CA, 1990. Morgan Kaufmann Publishers.

[22] M. Hare. The role of similarity in Hungarian vowel harmony: A connectionist account. Technical Report CRL 9004, Centre for Research in Language, University of California, San Diego, 1990.

[23] M. Hare, D. Corina, and G.W. Cottrell. Connectionist perspective on prosodic structure. Technical Report CRL Newsletter Volume 3 Number 2, Centre for Research in Language, University of California, San Diego, 1989.

[24] Catherine L. Harris and J.L. Elman. Representing variable information with simple recurrent networks. In *6th Annual Proceedings of the Cognitive Science Society*, pages 635–642, 1984.

[25] M.H. Harrison. *Introduction to Formal Language Theory*. Addison-Wesley Publishing Company, Reading, MA, 1978.

[26] S. Haykin. *Neural Networks, A Comprehensive Foundation*. Macmillan, New York, NY, 1994.

[27] J.A. Hertz, A. Krogh, and R.G. Palmer. *Introduction to the Theory of Neural Computation*. Addison-Wesley Publishing Company, Redwood City, CA, 1991.

[28] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Publishing Company, Reading, MA, 1979.

[29] J. Hopfield. Learning algorithms and probability distributions in feed-forward and feed-back networks. *Proceedings of the National Academy of Science*, 84:8429–8433, 1987.

[30] B. G. Horne and C. Lee Giles. An experimental comparison of recurrent neural networks. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems 7*, pages 697–704. MIT Press, 1995.

[31] L. Ingber. Very fast simulated re-annealing. *Mathematical Computer Modelling*, 12:967–973, 1989.

[32] L. Ingber. Adaptive simulated annealing (ASA). Technical report, Lester Ingber Research, McLean, VA, 1993.

[33] M.I. Jordan. Attractor dynamics and parallelism in a connectionist sequential machine. In *Proceedings of the Ninth Annual conference of the Cognitive Science Society*, pages 531–546. Lawrence Erlbaum, 1986.

[34] M.I. Jordan. Serial order: A parallel distributed processing approach. Technical Report ICS Report 8604, Institute for Cognitive Science, University of California at San Diego, La Jolla, CA, May 1986.

[35] S. Kirkpatrick and G.B. Sorkin. Simulated annealing. In Michael A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*, pages 876–878. MIT Press, Cambridge, Massachusetts, 1995.

[36] S. Kullback. *Information Theory and Statistics*. Wiley, New York, 1959.

[37] H. Lasnik and J. Uriagereka. *A Course in GB Syntax: Lectures on Binding and Empty Categories*. MIT Press, Cambridge, MA, 1988.

[38] Steve Lawrence, Sandiway Fong, and C. Lee Giles. Natural language grammatical inference: A comparison of recurrent neural networks and machine learning methods. In Stefan Wermter, Ellen Riloff, and Gabriele Scheler, editors, *Symbolic, Connectionist, and Statistical Approaches to Learning for Natural Language Processing*, Lecture notes in AI, pages 33–47. Springer Verlag, New York, 1996.

[39] Y. Le Cun. Efficient learning and second order methods. Tutorial presented at Neural Information Processing Systems 5, 1993.

[40] L.R. Leerink and M. Jabri. Learning the past tense of English verbs using recurrent neural networks. In Peter Bartlett, Anthony Burkitt, and Robert Williamson, editors, *Australian Conference on Neural Networks*, pages 222–226. Australian National University, 1996.

[41] B. MacWhinney, J. Leinbach, R. Taraban, and J. McDonald. Language learning: cues or rules? *Journal of Memory and Language*, 28:255–277, 1989.

[42] R. Miikkulainen and M. Dyer. Encoding input/output representations in connectionist cognitive systems. In D. S. Touretzky, G. E. Hinton, and T. J. Sejnowski, editors, *Proceedings of the 1988 Connectionist Models Summer School*, pages 188–195, Los Altos, CA, 1989. Morgan Kaufmann.

[43] M.C. Mozer. A focused backpropagation algorithm for temporal pattern recognition. *Complex Systems*, 3(4):349–381, August 1989.

[44] K.S. Narendra and K. Parthasarathy. Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, 1(1):4–27, 1990.

[45] C.W. Omlin and C.L. Giles. Constructing deterministic finite-state automata in recurrent neural networks. *Journal of the ACM*, 45(6):937, 1996.

[46] C.W. Omlin and C.L. Giles. Extraction of rules from discrete-time recurrent neural networks. *Neural Networks*, 9(1):41–52, 1996.

[47] C.W. Omlin and C.L. Giles. Rule revision with recurrent neural networks. *IEEE Transactions on Knowledge and Data Engineering*, 8(1):183–188, 1996.

[48] F. Pereira and Y. Schabes. Inside-outside re-estimation from partially bracketed corpora. In *Proceedings of the 30th annual meeting of the ACL*, pages 128–135, Newark, 1992.

[49] D. M. Pesetsky. *Paths and Categories*. PhD thesis, MIT, 1982.

[50] J.B. Pollack. The induction of dynamical recognizers. *Machine Learning*, 7:227–252, 1991.

[51] D.E. Rumelhart and J.L. McClelland. On learning the past tenses of English verbs. In D.E. Rumelhart and J.L. McClelland, editors, *Parallel Distributed Processing*, volume 2, chapter 18, pages 216–271. MIT Press, Cambridge, 1986.

[52] J. W. Shavlik. Combining symbolic and neural learning. *Machine Learning*, 14(3):321–331, 1994.

[53] H.T. Siegelmann. Computation beyond the turing limit. *Science*, 268:545–548, 1995.

[54] H.T. Siegelmann, B.G. Horne, and C.L. Giles. Computational capabilities of recurrent NARX neural networks. *IEEE Trans. on Systems, Man and Cybernetics - Part B*, 27(2):208, 1997.

[55] H.T. Siegelmann and E.D. Sontag. On the computational power of neural nets. *Journal of Computer and System Sciences*, 50(1):132–150, 1995.

[56] P. Simard, M.B. Ottaway, and D.H. Ballard. Analysis of recurrent backpropagation. In D. Touretzky, G. Hinton, and T. Sejnowski, editors, *Proceedings of the 1988 Connectionist Models Summer School*, pages 103–112, San Mateo, 1989. (Pittsburg 1988), Morgan Kaufmann.

[57] S.A. Solla, E. Levin, and M. Fleisher. Accelerated learning in layered neural networks. *Complex Systems*, 2:625–639, 1988.

[58] M. F. St. John and J.L. McClelland. Learning and applying contextual constraints in sentence comprehension. *Artificial Intelligence*, 46:5–46, 1990.

[59] Andreas Stolcke. Learning feature-based semantics with simple recurrent networks. Technical Report TR-90-015, International Computer Science Institute, Berkeley, California, April 1990.

[60] M. Tomita. Dynamic construction of finite-state automata from examples using hill-climbing. In *Proceedings of the Fourth Annual Cognitive Science Conference*, pages 105–108, Ann Arbor, MI, 1982.

[61] D. S. Touretzky. Rules and maps in connectionist symbol processing. Technical Report CMU-CS-89-158, Carnegie Mellon University: Department of Computer Science, Pittsburgh, PA, 1989.

[62] D. S. Touretzky. Towards a connectionist phonology: The "many maps" approach to sequence manipulation. In *Proceedings of the 11th Annual Conference of the Cognitive Science Society*, pages 188–195, 1989.

[63] A.C. Tsoi and A.D. Back. Locally recurrent globally feedforward networks: A critical review of architectures. *IEEE Transactions on Neural Networks*, 5(2):229–239, 1994.

[64] R.L. Watrous and G.M. Kuhn. Induction of finite state languages using second-order recurrent networks. In J.E. Moody, S.J. Hanson, and R.P Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 309–316, San Mateo, CA, 1992. Morgan Kaufmann Publishers.

[65] R.L. Watrous and G.M. Kuhn. Induction of finite-state languages using second-order recurrent networks. *Neural Computation*, 4(3):406, 1992.

[66] R.J. Williams and J. Peng. An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural Computation*, 2(4):490–501, 1990.

[67] R.J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280, 1989.

[68] Z. Zeng, R.M. Goodman, and P. Smyth. Learning finite state machines with self-clustering recurrent networks. *Neural Computation*, 5(6):976–990, 1993.