Proceedings of the Alife IV Workshop; Cambridge, MA; 1994.

Innate Biases and Critical Periods: Combining Evolution and Learning in the Acquisition of Syntax

John Batali
Department of Cognitive Science
University of California at San Diego
La Jolla, CA 92093-0515
`batali@cogsci.ucsd.edu`

Abstract

Recurrent neural networks can be trained to recognize strings generated by context-free grammars, but the ability of the networks to do so depends on their having an appropriate set of initial connection weights. Simulations of evolution were performed on populations of simple recurrent networks where the selection criterion was the ability of the networks to recognize strings generated by grammars. The networks evolved sets of initial weights from which they could reliably learn to recognize the strings.

In order to recognize if a string was generated by a given context-free grammar, it is necessary to use a stack or counter to keep track of the depth of embedding in the string. The networks that evolved in our simulations are able to use the values passed along their recurrent connections for this purpose. Furthermore, populations of networks can evolve a bias towards learning the underlying regularities in a class of related languages.

These results suggest a new explanation for the "critical period" effects observed in the acquisition of language and other cognitive faculties. Instead of being the result of an exogenous maturational process, the degraded acquisition ability may be the result of the values of innately specified initial weights diverging in response to training on spurious input.

1 Introduction

One of the most popular, and controversial, explanations of the ability of children to learn their native languages, is that humans are born with innate biases to be able to recognize and produce certain kinds of linguistic structures (Chomsky, 1987). Human languages are apparently too complex, and the evidence available to the language-learner is too sparse, for general-purpose learning mechanisms to accurately acquire linguistic competence (Gold, 1967; Wexler & Culicover, 1980).

It has been suggested that first language acquisition is the result of the maturation of some kind of language-specific acquisition device which subsequently disappears when the child gets older (Lennenberg, 1967). One line of evidence for this hypothesis is the apparent existence of a "critical period" for language acquisition (Newport, 1990). Children not exposed to a language before the age of five or so are often not able to learn language as well as those exposed to language while younger.

We are motivated by such issues (and the attendant controversies) to explore the interactions between evolution and learning in the acquisition of syntax. The proposals for "innate biases" and a "language acquisition device" are very vague, and, at best, promissory notes to be cashed out when and if we understand better how language is processed in the brain, and therefore what innate biases could be like, and what sorts of acquisition devices might be used. And of course if a bias or an acquisition device is innate, it must have evolved somehow.

Artificial neural connectionist networks constitute a concrete (and somewhat biologically plausible) model of computation and learning. However the precise limits of their computational power have yet to be determined. Fodor & Pylyshyn (1988) argue that neural networks cannot represent recursively structured representations in a way that allows for the sorts of mental inferences that people can perform. One response to this argument is that neural networks have been shown to be capable, in principle, of performing any computation that a universal Turing machine can perform (Pollack, 1987; Siegelmann, 1993). However the practical question remains, for any given task, whether a network can be *trained* perform the task upon exposure to examples.

General purpose neural network training algorithms, for example weight updating by backpropagation of error (Rumelhart, Hinton and Williams, 1986), are useful for a very wide range of problems, but whether or not a given network and learning algorithm will converge on a solution for a given episode of training is crucially dependent on the initial connection weights of the network. Small differences in the initial weights can determine whether or not the training algorithm will converge on a solution, how quickly convergence occurs, and the specific properties of the solutions that are found (Kolen & Pollack, 1990).

Such considerations suggest a specific version of an innateness hypothesis: The initial values of the connection weights of a neural network are innate, the result of evolutionary processes where the selection criteria are based

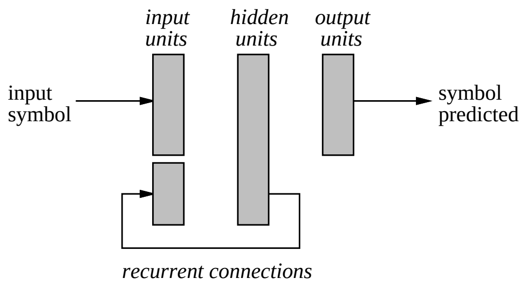

Figure 1: A simple recurrent network for recognizing strings generated by a grammar. Symbols from the string are presented sequentially to the input units. The network is trained to predict the next symbol in the string. Each unit in a layer has connections to each of the units in the next layer; these connections are not shown. Recurrent connections from hidden units feed back to the input layer. The specific numbers of input units, hidden units, output units, and recurrent connections varied in the different experiments.

on the ultimate performance of the networks after their connection weights are modified by training. The innate initial weights assure that the networks will efficiently and reliably find good solutions to the tasks they face.

Since the innate biases of the networks are realized as a set of specific values of initial connection weights, those biases are degraded if the weights of the networks diverge from those that evolved. Such divergence can be the result of training on spurious input. This suggests an explanation of some critical period effects that does not require any appeal to maturational processes or special-purpose language acquisition devices.

2 Networks & Grammars

Recurrent neural networks (as shown in figure 1) have been shown to be capable of recognizing and generating strings satisfying simple grammars (Jordan, 1986; Servan-Schreiber, Cleeremans and McClelland, 1988; Elman, 1990a). A typical training regimen (and the one used in the experiments reported here) consists of presenting the symbols from a string generated by the grammar sequentially to the input units of the network. Activation is then fed forward through the network. The activations of the output units are taken as a prediction of the next symbol in the string. The prediction is compared with the actual symbol that appears next, and the error between the predicted and correct value is backpropagated through the network to adjust the connection weights between units.

One way of characterizing the complexity of formal languages is in terms of the kinds of automata that can recognize strings from the language (Hopcroft & Ullman, 1979). The "regular languages," for example, can be recognized by finite-state machines. Recurrent neural networks are quite capable of being trained to implement finite state machines; they can learn to use the values passed along their recurrent connections to represent the state (Cleeremans, Servan-Schreiber and McClelland, 1989). Hence recurrent neural networks can be trained to recognize strings from regular languages.

Human languages are more complex than regular languages (Chomsky, 1957). In particular, all human languages allow for recursive nesting of phrase structure. The simplest class of formal languages which can generate strings exhibiting recursive phrase structure is the class of "context-free" languages. Recognition of strings in a context-free language requires, in addition to storing the state the recognizer is in, the use of a "pushdown stack" to keep track of the depth of nesting in the string.

An example of a syntactic pattern which requires a context-free grammar to express is that of the "center-embedded" constructions in natural languages. In such a construction some constituent, for example a noun phrase, must be matched with another constituent, for example a verb phrase, across some intervening material, for example a relative clause, which might exhibit the same structure.

Recall Fodor & Pylyshyn's (1988) arguments, mentioned above, that networks are incapable of manipulating recursive constituent structure. Christiansen (1992) wonders, based on evidence from natural language corpora, and psycholinguistic experiments, whether it is really sensible to presume that human languages can exhibit very deep nested structures. He suggests, in fact, that the limitations of networks in processing embedded structures match closely with those observed in humans, thus the network's limitations *support* their appeal as models of human language processing. While this response is intriguing, if we choose to accept Fodor & Pylyshyn's challenge, and take the ability to recognize whether strings were generated by a context-free grammar as a minimal criterion for the ability to handle recursive constituent structure, then we can demonstrate that ability by showing that part of the network is implementing a pushdown stack or some computationally equivalent device.

The results of Gold (1967) show that if a general-purpose learning algorithm is shown strings from a formal language, only regular languages can be learned. Gold lists several ways out of this predicament, including the one we are exploring: that the learner possesses innate biases to learn specific classes of languages. With respect to learning context-free languages, this means that the network must possess an innate bias to learn to implement and manipulate something equivalent to a stack.

Investigations of the ability of neural networks to learn to recognize strings exhibiting recursive phrase structure have begun to show how networks can represent that structure. For example in (Elman, 1990b) and (Weckerly & Elman, 1992) networks trained to recognize strings containing nested relative clauses and center-embedded constructions, respectively, use different trajectories of the activation values of their hidden units to represent the depth of embedding in the strings. Pollack (1990, 1991) shows how neural networks can implement dynamical systems and suggests that some of the attractors of such systems can perform the functions of a pushdown stack. Giles, et al. (1990) and Sun, et al. (1990) describe networks equipped with an external stack, which can learn to manipulate the stack and recognize context-free languages.

Elman (1993) addresses the difficulty of training networks to recognize recursive phrase structure. His solution involves beginning training with short input strings and gradually increasing the lengths of the strings. This training schedule is successful because the embedded material (e.g., relative clauses), exhibits the same sequential pattern as that of the structure it is embedded within. Even with the use of special training routines, learning to recognize strings generated by a grammar that allows recursively nested structure is difficult, time-consuming, and not guaranteed to succeed in any particular training trial.

3 Simulated Evolution

A compelling intuition, first presented by Baldwin (1896), is that evolution and learning can work synergistically. Suppose we are considering a species of animals whose fitness depends on their behavior. If the behavior is encoded entirely genetically, evolution will have difficulty locating a solution of relatively high fitness if such solutions are very small regions in the space of possible solutions. An animal whose behavior is modifiable by learning, on the other hand, can locate a highly adaptive solution if evolution places it near, rather than precisely on, the solution. The combination of evolution and learning thus broadens the optima in the solution space and gives evolution more of a gradient to work with. Hinton & Nowlan (1987) and Belew (1989), have explored this idea in computational simulations.

Another potential interaction between evolution and learning is illustrated by Kolen & Pollack's (1990) demonstration of the extreme sensitivity of the back-propagation network training algorithm to the values of networks' initial connection weights. Belew, McInerney, and Schraudolph (1991) suggest that the initial weights for networks be located by a genetic algorithm, where the fitness measure is based on the network's ultimate performance on a task after training. Nolfi, Elman and Parisi (1990) describe a population of simple creatures, controlled by networks whose initial weights are selected by a genetic algorithm, that evolve the ability to learn to behave appropriately for their environment.

In our experiments, the networks initial weights were specified directly by a sequence of real numbers. The initial weights of the first generation were uniformly distributed between -1 and $+1$. After each of the networks in the population was trained, its fitness was assessed. The top third of the networks survived unchanged into the next generation, with their connection weights reset to the initial values they had before training. Each of these networks was also used to create two offspring. For each offspring, a copy of the parent's initial weights was copied to the offspring. The offspring's initial weights were then modified by adding a random vector to the initial weight vector. The values at each position of the random mutation vector were normally distributed with a standard deviation of 0.05. The new offspring networks and the best networks from the previous generation were then trained, and the cycle repeated each generation.

There are a number of different approaches to the simulation of evolution (Goldberg, 1989) and the specific details described above were chosen mostly on the basis of trial-and-error. The direct connection (indeed identity) between the genotype and initial phenotype of these networks is obviously not biologically plausible, and we are exploring more realistic models of the interactions among genetics, development, and learning. On the other hand the genotype must be able to *somehow* affect the initial weights. Rather than try to duplicate the details of mutation and other genetic events, especially given the complexity that the development process introduces to the modeling of gene expression, we treated their cumulative effects as a random vector affecting the whole genome. Subsequent experiments suggest that simulating sexual reproduction by creating new initial weight sequences by combining those of two parents would speed up the evolutionary search, but no such crossover operation was used in the simulations reported here.

4 Learning a Simple Context-Free Language

A very simple context-free language consists of strings of the form $\mathtt{a}^n\mathtt{b}^n$, that is: some number of tokens of the symbol `a`, followed by the same number of tokens of the symbol `b`; thus: `ab`, `aaabbb`, `aaaaaabbbbbb`, etc.

Any machine that can recognize whether a string is in this language or not must somehow count up the number of `a`'s it sees, and count down each time it sees a `b`. While in general, context-free languages require a pushdown stack that can store arbitrary symbols at each level of the stack, for this language all that is required is a counter which can be incremented, decremented, and compared with zero.

A recognizer for this language must also keep track of which of two states it is in: an initial state corresponding to the sequence of `a`'s, and a second state corresponding to the sequence of `b`'s. During the period the machine is in the initial state, it may see either an `a` or a `b`; if it sees an `a`, it remains in the initial state and increments the counter; when it sees a `b`, it enters the second state and decrements the counter. While in the second state, it must only see `b`'s, and must decrement the counter for each one.

The networks used for learning this language had 3 input units, 10 hidden units, 3 output units and 7 recurrent connections from hidden units to the input layer. Training consisted of presenting the networks strings from the $\mathtt{a}^n\mathtt{b}^n$ language preceded and terminated by 'space' character. Prediction error was backpropagated through the network to update the connection weights. (The back-propagation learning rate for all of these experiments was 0.1. No momentum term was used.) Each network was trained for a total of 500,000 characters, which works out to about 33,000 strings. (The strings ranged in length

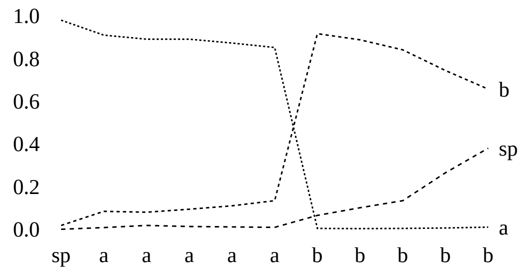

Figure 2: Performance of a randomly initialized network after training on strings in the $\mathtt{a}^n\mathtt{b}^n$ language. Values plotted are the activation values of output units, after the characters from the given string are presented to the network. The symbol '**sp**' stands for the string-delimiting 'space' character. This network had an average prediction error of 0.251 for the set of test strings.

from 4 to 26 characters.) Each network's average per-character prediction error for a set of 12 test strings from the language was then measured.

To assess the ability of networks to find solutions to this problem, 513 randomly initialized networks were trained on strings from the language. The mean value of the networks' per-character prediction error was 0.244, with a standard deviation of 0.031. The best network in this set achieved an average prediction error of 0.168.

The performance of a randomly initialized network after training on strings from the $\mathtt{a}^n\mathtt{b}^n$ language is shown in figure 2. The values plotted are the activation values of the network's output units, after each of the characters from the string is shown to the network. The network correctly predicts the appearances of the `a` character for the first half of the string, and once the first `b` is seen, the prediction for `a` drops to zero. In the second half of the string, the prediction for `b`, initially high, begins to drop off, and the the prediction for the 'space' character begins to rise. Although these predictions accord with the statistics of the training strings, the network has hardly learned the language very well. It should be predicting only `b` until as many `b`'s as `a`'s have been seen, at which point it should predict only the 'space' character. The behavior of the network shown in figure 2 is representative of virtually all of the randomly initialized networks.

A population of 24 of these networks was used as the initial generation of an evolutionary simulation. The fitness value of a network was proportional to the inverse of the average per-character prediction error it achieved after training on 500,000 characters from strings from the language.

By the 155th generation of this simulation, the best network had an error of 0.151 and the average error for all of the networks in the population was 0.179. This is 2.2 standard deviations better than the randomly initialized networks achieved.

Figure 3 shows the activation values of some of the units of a network that evolved in this experiment, as it is

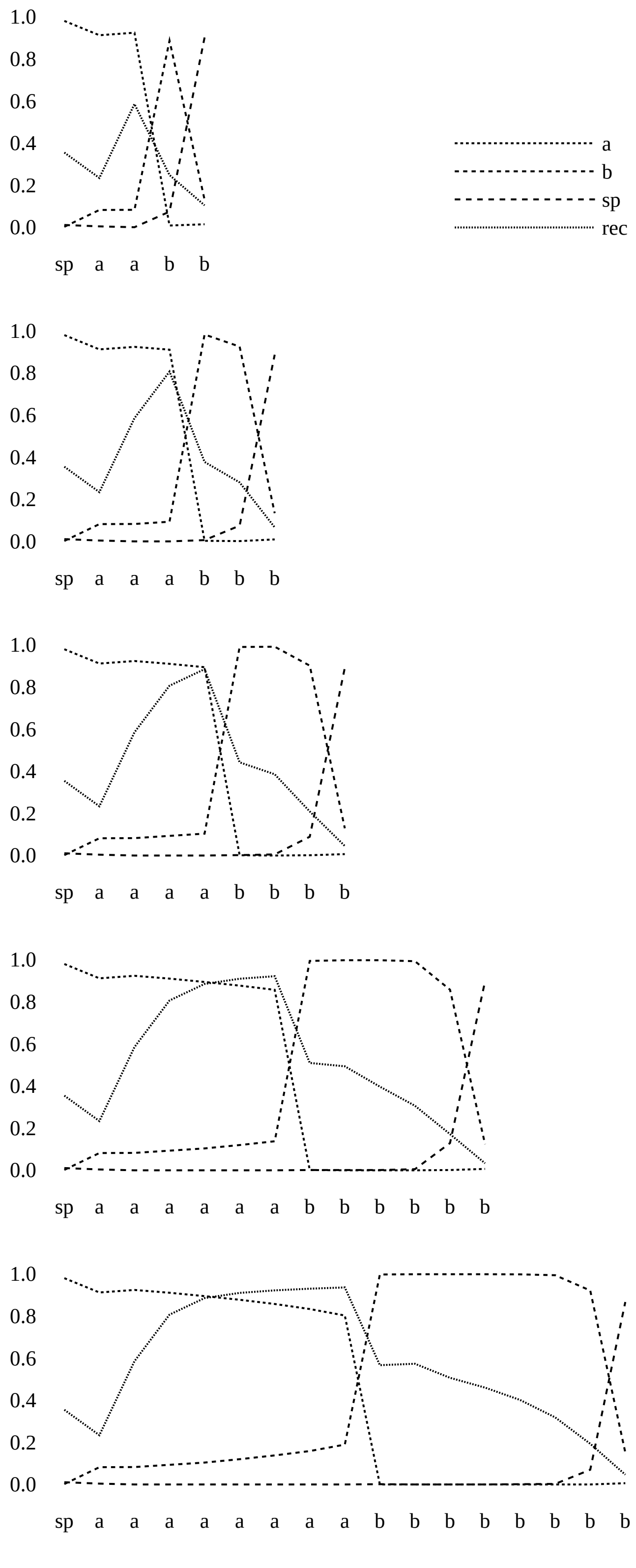

Figure 3: Activation values of units of an evolved network after training, while recognizing strings of various lengths. Plots labeled `a`, `b`, and `sp` show activation values of output units. The plots labeled `rec` show the activation values of a unit feeding one of the network's recurrent connections.

shown a string of characters. Note that the network predicts the a character very strongly after seeing the initial 'space' character, and then predicts a until it sees a b. From then on it predicts b until it has seen the same number of a's as b's, at which point it predicts the 'space' character.

The strong prediction of a for the first half of the string is interesting because this prediction does not match the statistics of the strings the network was trained on. Since a b can follow an a at any point in the string, the statistics would predict a and b at roughly equal levels throughout the first half of the string (or perhaps an initially higher value for a, decreasing uniformly throughout the first half of the string). However if the string is long enough, the average error the network receives over the length of the string is smaller if it receives a one-time error of 1.0 at the point where the first b appears, rather than an error of about .5 for each character in the first half of the string. The network has found a solution that reflects not the statistical regularities in its input, but the correct underlying rule.

The plots labeled rec show the activation value of one of the units feeding a recurrent connection in the network as it is presented the characters from the string. The activation value of this unit illustrates how the network manages to recognize the strings correctly. The unit is behaving as a counter: each a increases its activation value until b is seen, after which the value decreases. When the activation value of this unit approaches zero, the network predicts the 'space' character. As described above, the behavior of this unit is precisely what is required for the network to recognize the language $a^n b^n$.

The networks that emerged in the final generation of the simulation were able to learn to recognize strings whose level of embedding was 12, which was the level of embedding in the longest string the networks were trained on. If the networks were shown longer (and therefore more deeply embedded) strings, they would usually still be predicting b strongly at the end of the string.

5 Learning from a Class of Languages

The population of networks in the first experiment was able to combine evolutionary search with backpropagation learning to learn a particular language. An attractive solution would have been for a network to need no training at all — it would be "born" with the connections already correct to recognize the target language. This solution is not implemented in humans. Children are able to learn whatever language they are exposed to, no matter what language their parents spoke. While there are no innate biases to learn *particular* languages, it is possible that all human languages share certain abstract structural features, and there are innate biases to learn languages with those structural features.

We explored this idea by simulating the evolution of a population of networks trained on languages generated by a class of related grammars. In each generation, each of the members of a population of networks was trained on a particular language. A network's ability to learn the language it was trained on was used to compute the network's fitness value, and hence whether it survived into the next generation and reproduced. But in the next generation the network's offspring would, in general, face a different language. To flourish over the generations, networks had to develop biases not for a specific language, but for those properties shared by all of the languages in the class.

Each of the languages in the class used the symbols a, b, c, and d. In a given language, each of these four symbols is assigned to one of three categories: the 'push' symbols, the 'pop' symbols and the 'idle' symbols. In a grammatical string from any language in the class, each 'push' symbol must be matched on its right by a 'pop' symbol. Any number of 'idle' symbols can appear anywhere in a string, except that the first symbol in the string must be a 'push' symbol, and the last symbol in the string must be its matching 'pop' symbol.

Each language had at least one 'push', 'pop', and 'idle' symbol, and one of the three categories had two symbols. The 36 possible languages defined this way fall into three subclasses, corresponding to the languages with two 'push' symbols, two 'pop' symbols and two 'idle' symbols. There are 12 languages in each subclass, differing only in which symbols are assigned to which category.

For example one language in the class has a and b as the 'push' symbols, d as the 'pop' symbol and c is the 'idle' symbol. The string 'baadcadcdd' is in this language.

The intuition being explored here is that the specific lexical items used in languages are their most arbitrary features. Underneath lexical differences, languages may share aspects of linguistic structure (word order, case systems, phonological or morphological processes, etc.), and there may be underlying regularities common to all languages. Thus a language with c and d as its 'push' symbols, a as its 'pop' symbol, and b as its 'idle' symbol, would be structurally very similar to the language in the previous paragraph, although its strings would look quite different.

To recognize a string from one of these languages, an automaton must keep track of the value of a counter. Whenever it sees a 'push' symbol, it should increment the counter; when it sees a 'pop' symbol, it should decrement the counter; and when it sees an 'idle' symbol, it should keep the counter at the same value. When the counter reaches zero, the end of the string has been found. This computation is involved in recognizing all languages in the class, and hence the ability to quickly learn to perform it would be an ideal innate bias for the networks to acquire. Then, when exposed to a specific language during training, the network would only have to learn the specific mapping of the symbols to their categories.

In this experiment, networks with 5 inputs, 10 hidden layers, 5 outputs and 1 recurrent connection were used. The single recurrent connection was used in order to make the language-learning task as difficult as possible

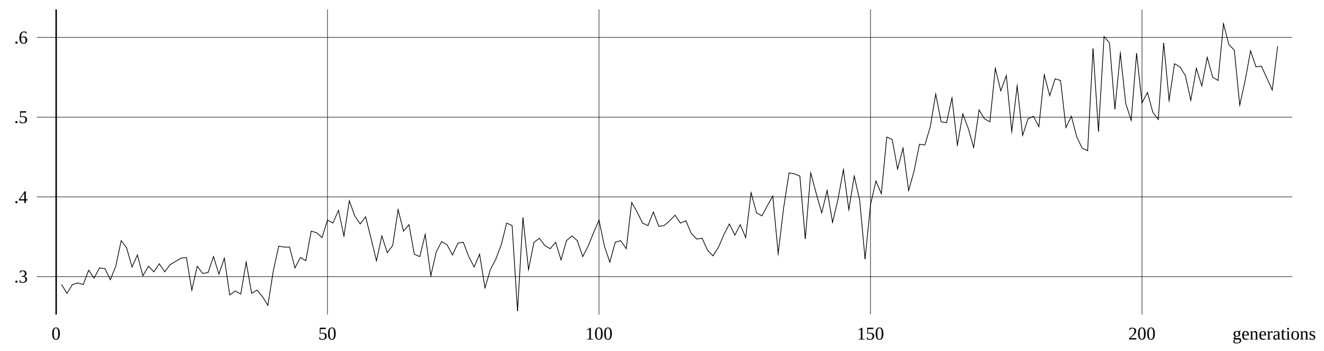

Figure 4: The evolution of networks trained on languages from the class of context-free languages. The value plotted is the average of the pred scores of the networks in each generation.

for the networks, given the relative simplicity of the languages compared to the $a^n b^n$ language.

The networks were trained on strings of characters as in the first experiment. However in this simulation, the performance of a network after training was assessed by computing the average value of its 'space' output unit at the end of each of a set of test strings. This value will be referred to as the "pred" value of the network. Ideally, it ought to be 1.0 — indicating that the network has correctly predicted the end of each of the strings. This value is more informative than the average per-character prediction error for these languages, as the the specific character that can follow another is much less constrained than in the earlier language. As before, the character prediction error seen was used for backpropagation training of the networks.

The difficulty of this task was assessed by training 436 randomized networks with varying numbers of hidden units and recurrent connections for 500,000 characters on languages from the class. The average pred value after training was 0.375 with a standard deviation of 0.117. The best network had a pred value of 0.827. Of 103 networks with a single recurrent connection, the average pred value was 0.280 with a standard deviation of 0.057; the best network had a pred value of 0.457.

The evolutionary simulation was organized in a similar fashion as the first experiment, with the pred value of each network used to compute its fitness. Figure 4 shows a record of the simulation. The value plotted at each generation is the average pred value for the networks in the population. After an initial period of relatively aimless search, the members of the population steadily improve their aptitude at learning the languages. (This run represents about one week's computation on a Sun Sparc Station 10.)

Figure 5 illustrates the performance of one of the networks that evolved. The activation of the 'space' output unit and the unit feeding the recurrent connection are plotted against the symbols of a string. The recurrent connection is used as a counter: the 'push' symbols `a` and `b` increment its value; the 'pop' symbol `d` decrements it, and the 'idle' symbol `c` modifies the value only slightly. When the recurrent value decreases far enough, the network signals the end of the string.

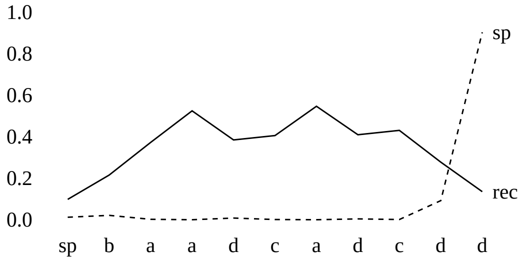

Figure 5: Activation values of units in one of the networks that evolved to learn languages from the class of context-free languages, after being trained on a specific language from the class. For this language, the symbols `a`, and `b` are 'push' symbols, `d` is the 'pop' symbol, and `c`, is the 'idle' symbol. The solid line plots the activation value of the unit feeding into the single recurrent connection of the network after the given character is seen. The dotted line plots the network's prediction of the end of string 'space' character.

After 150 generations, the average performance of the entire population is better than the best performance for the randomly initialized networks, and the performance of the population steadily increases after that. The networks are indeed developing an innate bias towards learning the languages in the class.

Some aspects of the way this innate bias works were investigated by looking at the performance of a network from a late generation of the simulation before it had been trained on any language. Its connection weights were therefore those it had inherited. When tested on a string from a language, its 'space' output unit is strongly activated: it is almost 1.0 for each character of the string; and the recurrent activation is essentially 0.0 throughout the string. Thus the "newborn" network is apparently hypothesizing the minimal language consistent with the data it has seen: namely the empty set. (This is a rather simple and extreme version of the sorts of strategies studied in formal learning theory (Wexler & Culicover, 1980).) Furthermore, this prediction of the 'space' symbol is innately associated with a zero recurrent input. This association will be reinforced later when the recurrent connection begins to be used as a counter, and its being zero signals the end of the string. Further investigation of the innate biases of networks to learn related languages is discussed below.

6 Critical Periods for Learning Languages

The development of a number of animal faculties shows a so-called "critical period" effect: Unless the animal is exposed to the appropriate environmental input at a certain period of its development, it may never acquire the full faculty. For example Marler (1991) describes how song sparrows deprived of the opportunity to hear the songs of other sparrows between the ages of 20 and 50 days will ultimately learn very reduced versions of the normal song of their species.

There is evidence for the existence of a critical period in the acquisition of language by humans. Newport (1990) describes how the ultimate ability of deaf children to master sign language decreases markedly with the age of first exposure, with significant degradation noticeable if the children were older than around five years when they first encountered sign language. Similar results are obtained in studies of the ability to learn a second language.

Lennenberg (1967), proposes that the critical period is the result of the maturation, and subsequent disappearance, of a specific language-learning mechanism. Newport suggests an explanation in which the child's initially limited cognitive abilities assist in the ability to learn language, for example by limiting the size of utterances that the child can store in memory and analyze. A related idea was explored by Elman (1993) in training recurrent neural networks to recognize recursively embedded strings by initially limiting the lengths of the strings, corresponding to either limited cognitive resources, or externally restricted input.

The experiments described here suggest a new explanation of critical period effects. A consequence of a network's innate biases being realized as specific values of initial connection weights, is that this bias can be degraded substantially if the weights of the network diverge from those that evolved. The backpropagation learning algorithm changes the weights of the network in response to an error signal based on a comparison between an expected output and the actual output of the network. If the network is trained on spurious input — strings not from the language it will eventually learn — its connection weights will be adjusted to values that differ from those that evolved. With enough such training, the weights of the network will diverge so far from the innate values that the network will have lost its innate ability to learn a language.

We presume that the network is trained on input patterns available in its environment. Ordinarily, these will be strings from the language that is ultimately to be learned. Deprived of exposure to a language, the network will be trained on whatever unstructured input exists; we model such unstructured input as random strings presented to the network.

Marchman (1993) proposes an explanation of the critical period effect that also does not invoke an exogenous maturational process. Her explanation, based on a neural network learning model, is that the network becomes "entrenched" in a particular solution after some training, and cannot find its way to new solutions. The explanation being presented here is compatible with Marchman's account, and indeed entrenchment effects will be described below. The difference is that in our account the effect is the result of the network's weights being removed from some ideal zone for learning, as opposed to being stuck in some other solution, as in Marchman's proposal.

One of the networks that evolved to learn languages from the class was trained for varying periods with random strings. After some number of characters of such training, the network was then trained on one of the languages in the class, for 500,000 characters. As illustrated in the left hand plot of figure 6 the ultimate performance of the network on learning a language decreases with the length of exposure to random input.

More dramatic degradation in the performance of the network is observed if the evolved network is trained on one language for a while, and then switched to another, as is illustrated in the right hand plot of figure 6. The network was first trained for some number of characters on languages from one of the three language subclasses. Language class 1 has two 'push' characters; language class 2 has two 'pop' characters, and language class 3 has two 'idle' characters. After the network was trained for the indicated number of characters on the other language, it was then trained for 500,000 characters on a language from class 1. (If the initial training was on a class 1 language, the subsequent training was on a different class 1 language.) The final average pred values for a set of test strings from the language is plotted. Note that there is little interference from a language of the same class, but that a language from one class can decrease the effectiveness of the network at learning a language from another class, sometime to *less* than the average performance of randomized networks at learning the same language.

Apparently what is happening here is that the innate biases the networks have evolved tend to place the networks' initial weights in zones from which training will guide the networks to good solutions. A period of random training will jiggle the weights around, but won't move them out of the zone right away. On the other hand, exposure to structured input will move the weights out of the initial bias zone towards a particular solution. Once out of that initial zone, a network will have a hard time finding its way back into it. In fact, enough training on structured input of the wrong kind (here, a different language class) will actually put the weights of the network into a zone from which learning cannot reach the optima available to most randomly initialized networks. This illustrates Marchman's notion of "entrenchment."

The behavior of the network illustrated by the right-

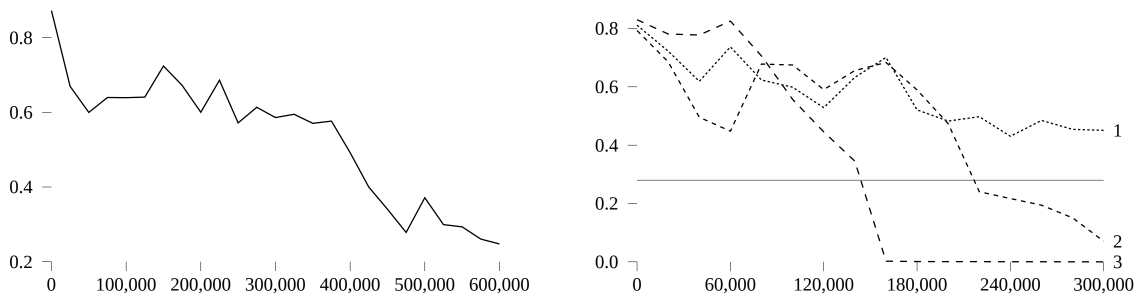

Figure 6: Performance of a network in learning a language after exposure to spurious input. One of the networks that evolved to learn from the class of related languages was chosen. The plot on the left illustrates the final pred value for the network after first being trained on the indicated number of characters from random sequences, and then trained on 500,000 characters from a language in the class. The plot on the right shows the final pred value for the network after first being trained on strings from a language from each of the three subclasses of languages for the indicated number of characters, and then trained on 500,000 characters from a language of class 1. The solid line indicates the average performance of randomly initialized networks. Both plots display average values taken over 20 training runs each. Note that the scales of the abscissas are different in the two plots.

hand plots in figure 6 also shed light on how the networks' innate biases work. The fact that there is less interference between members of the same language class suggests that the initial weights of the networks are such that they can quickly locate solutions that exploit the commonalities of the languages of the same class. Once such a solution is in place, it is possible for a network to learn a new mapping from symbols to their categories. On the other hand, once a network begins to develop a solution for a language of one class, that solution strongly interferes with the network's ability to learn both a new language's structure and its mapping from symbols to categories.

7 Learning Temporal Boolean Functions

Some of these issues were explored by investigating the evolution of networks on a simpler (and therefore more quickly trainable) recognition task. Each of the networks in a generation was trained on one of four temporal versions of two-input boolean functions. The four functions are:

`xor`	exclusive or of current and previous inputs
`-xor`	negation of `xor`
`prev`	value of previous input
`-prev`	negation of previous input

These functions were chosen because they were found by experiment to be the most difficult of the 16 temporal boolean functions for randomly initialized networks to learn.

An evolutionary simulation of 200 generations was performed with a population of 24 networks. Each network had 1 input unit, 3 hidden units, 1 output unit, and 1 recurrent connection. In a generation, each network was trained with 100,000 characters from one of the temporal boolean functions, and the network's final average prediction error was used to determine its fitness. As with the networks evolving to learn languages from the class of context-free languages, the offspring of networks which did well in a particular generation at learning the function they were trained on would, in general, face a different function than their parent did. Hence the selective pressure was toward acquiring whatever innate bias would help learn all of the functions, not just a specific one.

The following table shows the performance of networks after being trained on 100,000 characters from the temporal boolean functions. The column headed "Randomized" shows the final average prediction error of networks with random initial weights; the column headed "Evolved" shows the final average prediction error of networks whose initial weights were found by the evolutionary simulation. The results in this table are averaged over 400 training runs each.

	Randomized		Evolved	
boolean function	*mean error*	*standard deviation*	*mean error*	*standard deviation*
`xor`	.178	.088	.0151	.036
`-xor`	.182	.088	.0006	5.00×10^{-6}
`prev`	.205	.077	.0001	7.35×10^{-7}
`-prev`	.202	.080	.0001	1.02×10^{-6}

The networks have apparently evolved some sort of bias towards learning the boolean functions. At least some of the nature of this bias is illustrated in the table below. A network from the last generation of the simulation run was given a sequence of inputs before it was trained on any of the functions. The responses of its output unit and the unit feeding its recurrent connection are shown in the center columns. Note the activation value of the recurrent unit is very close to the input to the network. This value will be available to the network as input when the next input is seen. This innate response of the network means that when trained on the temporal boolean function, the network just has to learn

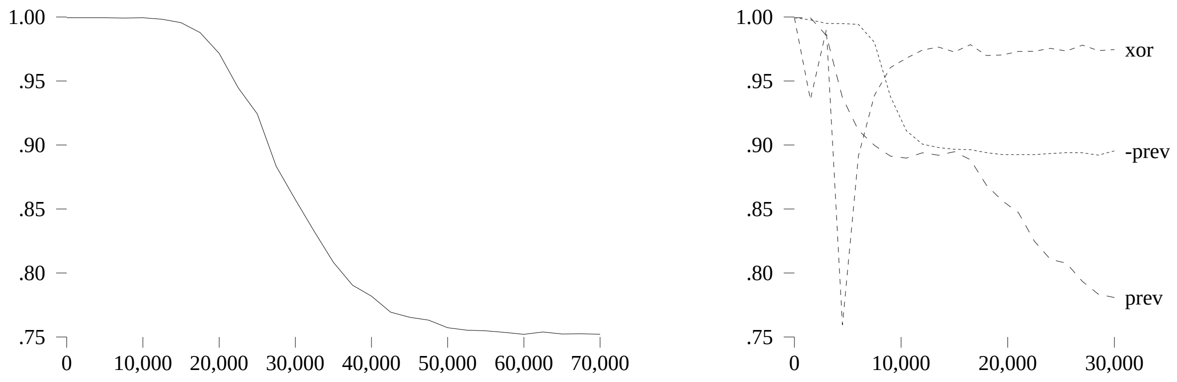

Figure 7: Performance of one of the networks that evolved to learn temporal boolean functions after exposure to spurious input. The value plotted is 1.0 minus the average error of the network for a set of test strings after training for 100,000 characters, after being exposed to spurious input for the indicated number of characters. The plot on the left shows the result for a network exposed to random sequences. The plot on the right shows the results for the network exposed first to the indicated temporal boolean function, and then to the `-xor` function. The values on both graphs are averaged over 400 training runs.

the non-temporal version of that boolean function. The right hand columns in the table show the performance of the network after training with 100,000 characters from the `xor` function. The behavior of the recurrent values haven't changed at all, and the network is correctly computing the temporal `xor` function.

		Before Training		After Training	
input	*correct output*	*actual output*	*rec value*	*actual output*	*rec value*
1	1	.19	.96	.97	.96
0	1	.10	.04	.98	.04
0	0	.16	.04	.02	.05
1	1	.19	.96	.97	.96
1	0	.12	.96	.03	.96
0	1	.10	.04	.98	.04

Figure 7 shows the performance of the network after exposure to spurious input. The left hand plot shows how the network's ultimate prediction error after training is affected by initial training on random sequences; it exhibits the classic shape of a critical period effect[1] — for a short period (in this case amounting around 20,000 characters) the spurious input has no effect at all on the ultimate performance of the network. The ultimate performance then drops rapidly, until after enough time (around 50,000) characters, the behavior of the network is the same as that of randomly initialized networks.

The plot on the right of figure 7 displays the final performance of the network after first being trained on one temporal boolean function for the indicated number of characters, and then trained on the `-xor` function.

The different functions have different effects on the ultimate ability of the network to learn `-xor`. Learning `xor` first has very little effect on the ultimate performance of the network (except for a very strong effect at 4500 characters, which will be discussed below). Learning `-prev` has almost no effect until around 7000 characters and then begins to have a slight, but significant effect. Learning `prev` has a rather dramatic effect on the ultimate performance of the network, decreasing it to that of randomly initialized networks after 30,000 characters.

The glitch at 4500 characters in the plot labeled `xor` provides a clue to how learning another function interferes with the subsequent learning of `-xor`. As was shown above, the networks have evolved a set of initial weights that result in their recurrent connections passing the input value back into the network with the next character. When the network has been trained on 4500 characters of the `xor` function, that behavior still occurs, and the network is also producing output values that are just beginning to approximate `xor`: the output values when a 1 is expected are slightly more than .5 and the output values when a 0 is expected are slightly less than .5.

If the function being learned is then switched to `-xor`, the network can respond to the new training in one of two ways: either by changing the values of weights computing the boolean function from the character input and the recurrent input, or by changing the values of the weights that result in the recurrent connection feeding the current input back into the network. The latter choice is made most of the time, and the recurrent connections begin to carry intermediate values. Since the recurrent pattern was a major source of the networks innate bias towards learning the functions, they then behave essentially as randomly initialized networks do.

After as few as 2000 more characters of initial training on `xor` however, this effect disappears and the network is able to learn `-xor` correctly even after 30,000 characters of exposure to `xor` first. Apparently after solidly learning `xor`, the network is able to locate a solution that involves negating an output unit without modifying the weights that determine the values passed along its recurrent connection.

[1]See, for example, figure 2 of Newport (1990), or figures 2.18 and 2.19 of Marler (1991).

In these experiments, the innate biases of the networks implement a partial solution to the problem that the networks face when learning each of the temporal boolean functions. Spurious input can degrade the network's ultimate performance in two ways: either by drifting the weights of the network away from the values implementing the innate partial solution; or by interrupting the standard development of the innate partial solution toward its being a complete solution for some particular function.

8 Discussion

Figures 3 and 5 demonstrate that recurrent neural networks can learn to implement a counter in the course of being trained to recognize strings from context-free languages, and therefore satisfy the requirement posed by the theory of formal automata. The networks are limited to a finite value for the maximum depth of recursive nesting in the strings, but so would any actual hardware implementation of an abstract automaton.

The languages our networks recognize are members of a subclass of the full class of context-free languages: they are members of the class of languages recognizable by *deterministic* automata. It is difficult to imagine how a network could efficiently implement a nondeterministic automaton — the network would somehow have to encode a complex representation of a search state in its activation values. On the other hand, it has been argued that deterministic automata are sufficient for processing natural languages (Marcus, 1980).

In order for the networks to be able to learn to recognize context-free languages, they had to start learning with proper sets of initial connection weights. The values of those initial weights were found by a simulation of evolution. The enormous investment in computational power illustrated by evolutionary searches like that shown in figure 4 make it questionable whether such searches really are a good way to obtain networks which can recognize strings from grammars. However the point is not to construct individual networks capable of recognizing specific languages. Instead, these results demonstrate that the members of a species can evolve biases to learn different, but related, languages reliably and efficiently. These biases would be crucial for the members of a species whose subgroups face different, but related, communicative situations.

The innate biases the networks acquire seem to confer sensitivity to the underlying regularities of a class of languages. In the case of learning the temporal boolean functions, the networks inherited a partial solution to the task they had to learn. Such sensitivity to underlying regularities, and partial solutions, makes the learning task faced by the network easier. Only the more superficial aspects of the language need to be learned upon exposure to examples. This was the initial motivation for the proposal of innateness in the first place: a difficult (or impossible) learning task is made tractable by innate mechanisms or biases. However no domain-specific innate acquisition mechanism needs to be introduced. Instead all that is required is a general-purpose learning mechanism in an appropriate initial configuration.

The sensitivity of the networks to the underlying regularities of the class of languages could make one suspect that the network has an innate set of formal constraints characterizing the abstract form of a grammatical string. Language acquisition on this model consists in discovering specific rules consistent with this set of general constraints (Pinker, 1989). But any constraints the network possesses, and any rules it learns, are represented only implicitly, in the connection weights of the network. Indeed it might be best not to think of these initial connection weights as a specific set of formal constraints and rules, but as the states of a dynamical system interacting with an environment (the set of training examples). It seems likely that the behavior of such systems would submit to quite a different sort of analysis than that of formal learning theory (Wexler & Culicover, 1980), leading to a different characterization of the process of human language acquisition than that of acquiring rules subject to innate constraints.

Though these investigations were motivated partly by Fodor & Pylyshyn's (1989) critical analysis of the adequacy of neural network models of cognition, the results presented here are a long way from a refutation of their argument. The networks have not been shown to manipulate abstract recursive structures to perform valid inference, or any other of the sort of semantically sound "structure sensitive" operations that Fodor & Pylyshyn take to be criterial of cognition.

It is important, however, to note that their arguments for the existence of recursively structured mental representations rest on observed human abilities to perform certain tasks: to find two sentences to be synonymous, for example; or to draw a valid inference from a set of premises. It is a *hypothesis* that recursively structured representations are needed to account for these abilities. The networks described here, for example, are able to recognize recursively characterizable structures with no recursively structured representations. If it turns out that more and more cognitively relevant abilities can be achieved without recursively structured representations, than the fact that neural networks are, or are not, capable of such representations becomes less relevant.

It is likely that there are a number of interacting explanations for the critical period effects observed in the development of some cognitive abilities. For one thing, it is possible there are adaptive benefits for animals to acquire cognitive abilities in a certain sequence or at certain times (it is probably beneficial for children, for example, to learn language while rather young). Also there might be a value towards focusing cognitive effort on one problem at time, perhaps under the control of general maturational processes.

The explanation of the critical period effect proposed here is a direct consequence of innate biases being realized as the initial state of a general-purpose learning mechanism. Such effects would be observed whether or not any other maturational process or mechanism exists.

We are currently exploring better ways to analyze the initial weights of the evolved networks, to better understand exactly what the innate biases are encoding. We are also attempting to evolve networks that can learn more complex and more realistic grammars. All of our evolutionary simulations so far have involved formal languages that we imposed on the networks to learn. We intend to explore the evolution of populations of networks whose fitness is based on their performance on a communication task, to see what kinds of languages arise.

9 Conclusion

The proposal of "innateness" as a solution to puzzles in cognitive development has been around since ancient times. With the advent of computational modeling of evolutionary processes comes the possibility of evaluating concrete proposals for the mechanisms of innateness. The experiments reported herein are an exploration in that direction.

The ability to recognize and generate complex temporal patterns, necessary for language as well as other activities, is important enough to expect that some innate ability to learn to do so would be beneficial. Recurrent neural networks could implement that ability, with the innate biases realized as appropriate values for the networks' initial connection weights.

Acknowledgments

I would like to thank Rik Belew and Jeff Elman for valuable advice and encouragement.

References

Baldwin, J. M. (1896). A new factor in evolution. *American Naturalist*, 30, pp. 441–451.

Belew, R. K. (1989). When both individuals and populations search: Adding simple learning to the genetic algorithm. *Proceedings of the 3rd International Conference on Genetic Algorithms,* J. D. Schaefer, (ed.), Morgan Kaufman.

Belew, R. K., McInerney, J., and Schraudolph, N. N. (1991). Evolving networks: Using the genetic algorithm with connectionist learning. In *Artificial Life II, SFI Studies in the Sciences of Complexity, Volume X.* C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, (eds.), Addison-Wesley, pp. 511–547.

Chomsky, N. (1957). *Syntactic Structures.* The Hague: Mouton.

Chomsky, N. (1987). *Knowledge of Language: Its Nature, Origin, and Use.* New York: Praeger.

Christiansen, M. (1992). The (non)necessity of recursion in natural language processing. *Proceedings of the Fourteenth Annual Meeting of the Cognitive Science Society,* Indiana University, Bloomington.

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

Elman, J. L. (1990a). Finding structure in time. *Cognitive Science*, 14, pp. 179–211.

Elman, J. L. (1990b). Distributed representation, simple recurrent networks, and grammatical structure. *Machine Learning*, 7, pp. 195–225.

Elman, J.L. (1993). Learning and development in neural networks: The importance of starting small, *Cognition*, 48, pp. 71–99.

Fodor, J., & Pylyshyn, Z. (1988). Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28, pp. 3–71.

Giles, G. L., Sun, G. Z., Chen, H. H., Lee, Y. C., and Chen, D. (1990). "Higher order recurrent networks & grammatical inference." In *Advances in Neural Information Processing Systems*, 2, D. S. Touretzky (ed.), San Mateo CA: Morgan Kaufman.

Gold, E. M. (1967). Language identification in the limit. *Information and Control*, 16, pp. 447–475.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning.* Reading, MA: Addison-Wesley.

Hinton, G., & Nowlan, S. J. (1987). How learning can guide evolution. *Complex Systems*, 1, pp. 495–502.

Hopcroft, J. E. & Ullman, J. D., (1979). *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley.

Jordan, M. (1986). *Serial Order: a Parallel Distributed Processing Approach.* ICS Report No. 8604. Institute for Cognitive Science; University of California at San Diego.

Kolen, J. F., & Pollack, J. B. (1990). Back propagation is sensitive to initial conditions. *Complex Systems 4*, pp. 269–280.

Lennenberg, E. H. (1967). *Biological Foundations of Language.* New York: John Wiley & Sons, Inc.

Marchman, V. A. (1993). Constraints on plasticity in a connectionist model of the English past tense. *Journal of Cognitive Neuroscience*, 5, pp. 215–234.

Marcus, M. (1980). *A Theory of Syntactic Recognition for Natural Language.* Cambridge, MA: MIT Press.

Marler, P. (1987). The instict to learn. In *The Epigenesis of Mind: Essays on Biology and Cognition.* Susan Carey & Rochel Gelman (eds.), Hillsdale, New Jersey: Lawrence Erlbaum Associates. pp. 37–66.

Newport, E. (1990). Maturational constraints on language learning. *Cognitive Science*, 14, pp. 11–28.

Nolfi, S., Elman, J. L., and Parisi, D. (1990). *Learning and Evolution in Neural Networks.* CRL Technical Report 9019. Center for Research on Language. University of California at San Diego.

Pinker, S. (1989). *Learnability and Cognition.* Cambridge, MA: MIT Press.

Pollack, J. B. (1987). *On Connectionist Models of Natural Language Processing.* PhD Thesis, Computer Science Department, University of Illinois, Urbana. Available as Memorandum MCCS-87-100, Computing Research Laboratory, New Mexico State University.

Pollack, J. B. (1990). Language acquisition via strange automata. In *Proceedings of the Twelfth Annual Conference of the Cognitive Science Society.*

Pollack, J. B. (1991). The induction of dynamical recognizers. *Machine Learning,* 7, pp. 227–252.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations.* David E. Rumelhart, James L. McClelland and the PDP Research Group. Cambridge, MA: MIT Press, pp. 318–362.

Servan-Schreiber, D., Cleeremans, A., and McClelland, J. L. (1988). *Encoding sequential structure in simple recurrent networks.* CMU Technical Report CMU-CS-335-87). Carnegie-Mellon University, Department of Computer Science.

Siegelmann, H. T. (1993) *Foundations of Recurrent Neural Networks.* PhD Dissertation. Rutgers University, Graduate Program in Computer Science.

Sun, G. Z., Chen, H. H., Giles, G. L., Lee, Y. C., and Chen, D. (1990). Connectionist pushdown automata that learn context-free grammars. *Proceedings of the International Joint Conference on Neural Networks, Vol. 1.* M. Caudill (ed.) Hillsdale, NJ: Lawrence Erlbaum.

Weckerly, J., & Elman, J.L. (1992). A PDP approach to processing center-embedded sentences. In *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society.* Hillsdale, NJ: Erlbaum.

Wexler, K., & Culicover, P. (1980). *Formal Principles of Language Acquisition.* Cambridge, MA: MIT Press.