Chapter 8

LESSONS FROM LANGUAGE LEARNING

Stefan C. Kremer

Guelph Natural Computation Group
Dept. of Computing and Information Science
University of Guelph

I. INTRODUCTION

Recurrent networks can be categorized into two classes, those that are presented with a constant or one-time input signal and are designed to enter an interesting stable state, and those that are presented with time-varying inputs and are designed to render outputs at various points in time. This chapter concerns the latter which are called dynamical recurrent networks [Kolen]. In this case, the operation of the network can be described by a function mapping an input sequence to an output value, or a sequence of output values. The input and output values are continuous and multi-dimensional, resulting in vector representations. Specifically, we define the behaviour of a network by

$$f : X^t \rightarrow Y^t, \tag{1}$$

where $X = \Re^n$ and $Y = \Re^m$ and n and m represent the dimensionalities of the input and output vectors, respectively. t represents the length of the sequence which is usually given a temporal interpretation.

A. LANGUAGE LEARNING

A special case of this type of operation which is often used in many recurrent networks is the assumption that

- $X = \{0, 1\}^n$ and $Y = \{0, 1\}^m$ for a logistic transfer function

  or

- $X = \{-1, 1\}^n$ and $Y = \{-1, 1\}^m$ for a logistic transfer function

In this situation, the input and output values are discrete. This approach is used in any problem where inputs are selected from a discrete alphabet of valid values and output values fall into discrete categories.

The problem of dealing with input sequences in which each item is selected from an input alphabet can also be cast as a formal language problem. A formal language is defined:

Definition 1 *Formal Language: a set of strings of symbols from some alphabet.*

© 2001 by CRC Press LLC

Typically Σ is used to represent the alphabet, and the input language L is the power-set of Σ:

$$L = 2^{\Sigma}. \tag{2}$$

One of the most simple functions that has a language as its domain is that of identifying a particular subset of this input language. That is, we consider a language

$$L_1 \subseteq L \tag{3}$$

and we define $f_{L_1} : L \rightarrow \{\text{accept}, \text{reject}\}$ as:

$$f_{L_1}(s) = \begin{cases} \text{accept} & \text{if } s \in L_1 \\ \text{reject} & \text{otherwise} \end{cases} \tag{4}$$

This is the classical problem solved by a formal computing machine (such as a finite state automaton, pushdown automaton, or Turing machine) which is said to accept the language L_1.

A recurrent network can be applied to this type of problem as well, and many have studied these networks in this context (see this Chapter's references for detailed list). With a recurrent network, one typically would like the system to learn to make this type of categorization (though there have been numerous papers on the representational powers of these networks independent of their ability to learn to identify member strings; e.g., Horne [1994], Siegelmann [1995].

B. CLASSICAL GRAMMAR INDUCTION

Now, when one talks of learning to make this categorization, one could equivalently talk about learning the language L_1. Yet, if the language L_1 is infinite in size, then to learn L_1 one has to represent it in some finite form. We define this finite form as a grammar:

Definition 2 *Formal Grammar: a finite characterization of a potentially infinite language.*

The classical approach to representing grammars is to use a 4-tuple, $G = (V, T, P, S)$. Here, V represents a set of symbols, known as variables, which are used as intermediate results in the derivation of member strings. Similarly, T represents a set of symbols, called terminals, which defines the alphabet of the language represented by the grammar (i.e., $T = \Sigma$). P is a finite set of rules, called productions, defining how strings of variables and terminals can be rewritten as other strings of variables and terminals in the process of deriving a member string. Specifically, productions take the form $\alpha \rightarrow \beta$, where α and β are strings of symbols from the Kleene closure of the union of variables and terminals: $(V \cup T)^*$. Lastly, S is a special variable called the start symbol.

The process of deriving a legal string for a given grammar can be formalized as follows: First, the current string is initialized to be the start symbol. Second, strings of symbols within the current string that match the left-hand side of one of the productions are replaced by the right-hand side of the production. The second

© 2001 by CRC Press LLC

step is repeated until only terminal symbols remain in the current string, at which point, the current string represents a legal string. Formally, we define the rewrite operator, $\Rightarrow$, by asserting that $\gamma\alpha\delta \Rightarrow \gamma\beta\delta$ if and only if the production $\alpha \rightarrow \beta$ is a member of P. This operator represents one application of step two in the process above. Multiple applications can then be represented by the reflexive-transitive rewrite operator, $\overset{*}{\Rightarrow}$, which is defined as the reflexive and transitive closure of $\Rightarrow$. Applying the latter operator to the start symbol, S, the language described by the grammar, G, is defined as

$$L_G = \{s | s \in T^* \text{and} S \overset{*}{\Rightarrow} w\} \tag{5}$$

C. GRAMMATICAL INDUCTION

This representation of a language by a grammar has lead the field of language learning to be known as *grammatical induction*. This field has been studied extensively in the purely symbolic paradigm for over 30 years. In those 30 years, much knowledge has been acquired and some of this knowledge can be parlayed into techniques for improvement of learning in recurrent networks. This is the focus of this chapter.

D. GRAMMARS IN RECURRENT NETWORKS

Another finite representation of potentially infinite languages is in the form of a weight matrix W in a recurrent network. In this scenario, string acceptance is determined by presenting vector encodings of the input symbols to the network, one at a time. Activations are propagated through the network for multiple cycles, until all input symbols have been presented. At that time, the output units are examined and a decision on membership is made. Because recurrent networks can store information about previous input symbols in the activation values transmitted through their recurrent connections, these networks can render decisions on strings presented one symbol at a time. The decisions made for the input strings will be defined by the weight matrix of the network. Thus, the weights define the set of strings that will be accepted and hence the language. In this sense, the weight matrix is a grammar according to our definition.

Although the representations in the connectionist paradigm are very different from those used in the classical symbolic approach, the problem faced by the two approaches is exactly the same. This means that it may be possible to transfer some insights on the problem of grammar induction from symbolic techniques to recurrent networks.

Clearly, the problem of language learning defined above is only one special case of the kinds of problems that recurrent networks can address. It does not cover situations in which input sequences consist of continuous real-values, nor on problems involving more sophisticated outputs beyond simple accept/reject decisions. Nonetheless, we will discover in what follows that it is an informative case that offers insights into approaching the problem of training recurrent networks in general.

© 2001 by CRC Press LLC

E. OUTLINE

The chapter is organized along the lines of four lessons based on results from the work in symbolic grammar induction. After this introductory section, Lesson 1 shows that the problem of language learning is a surprisingly difficult one. This motivates the remaining sections which focus on how one can simplify the problem of language learning and also other types of recurrent network problems. Lesson 2 focuses on restricting the kinds of languages and other problems which can be learned. Lesson 3 describes techniques for ordering the search for problem solutions to speed the learning time. Lesson 4 explains how ordering training data during the training process helps narrow down the solution possibilities. A conclusions section at the end of the chapter summarizes our results.

II. LESSON 1: LANGUAGE LEARNING IS HARD

We begin by considering two variations on grammar learning. Gold [Gold, 1967] has identified two basic methods of presenting strings to a language learner: "text" and "informant." A text is a sequence of legal strings containing every string of the language at least once. Typically, texts are presented one symbol after another, one string after another. Since most interesting languages have an infinite number of strings, the process of string presentation never terminates.

An informant is a device which can tell the learner whether any particular string is in the language. Typically the informant presents one symbol at a time, and upon a string's termination supplies a grammaticality judgment.

Gold [Gold, 1967] investigated the problem of language identification in the limit. He asked the question: Which classes of languages are learnable with respect to a particular method of information presentation? A class of languages is learnable if there exists an algorithm which repeatedly guesses languages from the class in response to example strings, and "Given any language of the class, there is some finite time after which the guesses will all be the same and will all be correct" [Gold, 1967]. The algorithm does not keep guessing forever or, more precisely, it settles on a particular language and that language is correct.

Gold showed that this is a surprisingly difficult task. For example, if the method of information presentation is a text, then only finite cardinality languages can be learned. Finite cardinality languages consist of a finite number of legal strings, and are a small subset of the regular sets (the smallest set in the Chomsky hierarchy). In other words, none of the language classes typically studied in language theory are text learnable.

The situation is only slightly more promising if both positive and negative examples from the language are available. Under informant learning, only two kinds of language are identifiable in the limit. These are regular sets (which are those languages having only transition rules of the form $A \rightarrow wB$, where A and B are variables and w is a (possibly empty) string of terminals), and context-free languages (which are those languages having only transition rules of the form $A \rightarrow \beta$, where A is a single variable). Other languages, however, like the recursively enumerable languages (those having transition rules, $\alpha \rightarrow \beta$, where α and

© 2001 by CRC Press LLC

β are arbitrary strings of terminals and non-terminals) remain unlearnable.

The fact that regular sets and context-free languages are learnable under the informant learning paradigm by no means implies that such learning is practical. Pinker points out that "in considering all the finite state grammars that use seven terminal symbols and seven auxiliary symbols (states), which the learner must do before going on to more complex grammars, he must test over a googol (10^{100}) candidates" [Pinker, 1979]. This reveals that even for tiny computational machines (seven states) the language learning problem is often intractable if no *a priori* knowledge is available to remove some of the machines from consideration.

The conclusion which must be drawn from Gold's and Pinker's observations is that grammatical induction is an exceptionally complex task. So complex, in fact, that it cannot be solved as originally posed by Gold. In the following sections, we shall present a number of modifications to the original problem which overcome the inherent difficulties implied by Gold's and Pinker's conclusions and thus allow the problem to be solved.

Even though Gold and Pinker worked in a symbolic paradigm for language learning, they made no assumptions particular to this approach. The same conclusions can be applied to the problem of learning languages by connectionist networks. This means that if recurrent networks are to solve language learning problems or even more complicated problems, we must modify our approach to learning in order to overcome the intractability and extreme slowness inherent in language learning tasks.

III. LESSON 2: WHEN POSSIBLE, SEARCH A SMALLER SPACE

The difficulty of any search depends on the number of candidate solutions that must be considered, called the hypothesis space.

A. AN EXAMPLE: WHERE DID I LEAVE MY KEYS?

We begin by considering a simple example: Suppose you are unable to find your car keys. We shall assume that the keys are somewhere in the house. A simple search algorithm might involve searching the house from top to bottom starting on the upper floor and moving down to the basement. This represents an exhaustive brute-force search like the scenario suggested by Pinker when he described considering all grammars with seven terminal and seven non-terminal symbols.

Now suppose you know for a fact that you have not been upstairs or downstairs since you last used the keys. In this case, it would be sensible to reduce the search space from the entire house to just the ground floor. This would no doubt lead to a more efficient search and you would expect to find your keys sooner. Thus, reducing the search space increases search efficiency.

Of course, there is a drawback to a reduced search space. Suppose you had forgotten that you had in fact traveled upstairs and left the keys there. Now your search of only the ground floor would be guaranteed to fail. A reduced hypothesis space is useful only if it does not exclude the goal.

© 2001 by CRC Press LLC

B. REDUCING AND ORDERING IN GRAMMATICAL INDUCTION

Naturally, the techniques of hypothesis space reduction and ordering described in the previous example are applicable to search in general-not just car key searches. As such, they can be used to make the task of grammatical induction solvable or tractable. The notion of hypothesis space reduction in the context of grammatical induction refers to searching for a grammar consistent with the training data in a class which is smaller than the class of unrestricted (Chomsky type-0) grammars.

Symbolic grammar induction systems have used the class of context-free grammars (Chomsky type-2) and the class of regular grammars (Chomsky type-3) as reduced hypothesis spaces. However, the fact that Gold showed that even the smallest of these classes is not learnable based solely on text training data, combined with the fact that most interesting grammars belong to the larger classes, have made these restrictions unpopular techniques for hypothesis space reduction.

A more useful technique is to devise a class of grammars which lies tangential to Chomsky's hierarchy. Such a tangential class contains some grammars which are not regular and some grammars which are not context-free but contains only a subset of the unrestricted grammars. By using a class tangential to the Chomsky hierarchy as one's hypothesis space it will be possible to represent some of the grammars which only fall into the unrestricted class, while at the same time reducing the size of the hypothesis space so as to identify members of the space based on input data more rapidly. Of course, as with the car key example, it is critical to choose a hypothesis space that contains those grammars which are to be learnable.

This type of hypothesis restriction was first suggested by Chomsky [Chomsky, 1965]. While working on the problem of human language acquisition, he proposed that only those grammars possessing the basic properties of natural languages should be considered as candidates for grammatical induction. By weighting the naturalness of languages based on a specific set of properties, he proposed an induction algorithm which considered only those languages which were consistent with the training sample and had a sufficiently high weight.

Another popular technique for restricting the space is to employ the *universal base hypothesis*. Under this hypothesis, different grammars are defined by means of a two-step process. First, a universal base grammar which all different grammars use is defined. Then, a restricted class of rewrite rules are employed to translate from the symbols of the universal base grammar to a variety of derived grammars. The grammars derived in this fashion form a reduced hypothesis space which can then be used to define a grammatical induction algorithm. This approach is fundamental to Wexler and Culicover's [Wexler, 1980] model of human language learning.

C. RESTRICTED HYPOTHESIS SPACES IN CONNECTIONIST NETWORKS

Restricted hypothesis spaces in symbolic grammatical induction systems are typically described in terms of restrictions on the type of grammar rules they em-

© 2001 by CRC Press LLC

ploy. In recurrent networks, the languages that the network recognizes are determined by three factors: (1) the network topology (sometimes called architecture), (2) the number of hidden units, and (3) the connection weights in the network. Thus, we can restrict the kinds of languages (or equivalently grammars) that can be learned by adjusting network topologies, number of hidden units and/or weights.

D. LESSON 2.1: CHOOSE AN APPROPRIATE NETWORK TOPOLOGY

The computational power of a number of topologies, given appropriate weights, has been studied. While some topologies are potentially as powerful as Turing machines, others are much more restricted in their computational power. Initially, one might be tempted to select the most powerful network for all applications, but the arguments above reveal that it may be wiser to select a more restricted architecture in order to make the learning algorithm tractable, especially if one knows that a solution to the current problem can be found by the computationally weaker architecture.

One of the first architectures suggested for processing time-sequence data was the window in time network used in the classic NETtalk [Sejnowski, 1986]. It has also been used by a variety of other authors including Lang et al. [Lang, 1990], Lapedes and Farber [Lapedes, 1987], and Waibel et al. [Waibel, 1989]. The topology of this network consists of a feedforward network which is presented with a finite history of input patterns called an input window (Figure 1). Since this window is of finite length, there will always be inputs which fall outside of this window (i.e., are too old). This means that there will always be certain kinds of strings that this network cannot correctly classify, namely strings who's categorization depends on symbols that fall outside the window. More specifically, Giles, Horne, and Lin [Giles, 1995] were first to recognize that Kohavi [Kohavi, 1978] had previously called this subclass of finite state automata ”definite machines.” Kremer [Kremer, 1995a] also developed a grammatical formulation in the form of a 4-tuple for this language.

The preceding architecture is not a recurrent network though it can be used in applications where recurrent networks are used (which explains why we discussed it here). A variation on the Window-in-Time topology is to use two temporal windows: One window on the input symbols (as in WIT memories) and a second on the output symbols produced by the network (Figure 2). In this network, output values are fed back into the network as inputs. Because of its similarity to infinite impulse response filters (IIRs), this type of topology has been called neural network IIR. Narendra and Parthasarathy [Narendra, 1990] have used this type of short-term memory.

Locally recurrent [Frasconi, 1995] networks use a different kind of recurrence. In these networks the activation values of hidden nodes are computed according to the formula

$$a_j(t) = f(net_j(t)), \tag{6}$$

© 2001 by CRC Press LLC

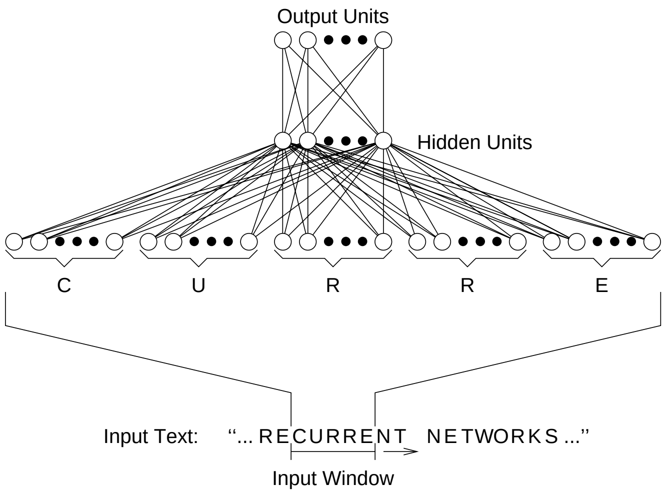

Figure 1. Window-in-time Network.

where

$$net_j(t) = \sum_i w_{ji} \cdot a_i(t) + w_{jj} a_j(t-1). \tag{7}$$

Here, w_{jj} represents a recurrent, time-delay connection from the unit to itself (Figure 3). A potential advantage of this type of network over the previous networks discussed is that it can adapt its internal representation. Whereas the previous networks had their memory fixed by the network's inputs or target outputs, this type of network can adapt their internal representations to the given task and it is these internal representations that are fed through the recurrent connections. Despite this, these networks are still limited in their representational capacity [Frasconi, 96]. In Kremer [1999], some specific problems that these networks cannot represent are identified.

Another approach, which has been widely used, is based on computing the network's internal state using a single-layer first-order feedforward network [Rumelhart, 1986] which uses the previous state (also called context) and the current input symbol as input (Figure 4). This approach is used in Elman's [Elman, 1991a, Elman, 1990] Simple Recurrent Networks (SRN), Pollack's [Pollack, 1989, Pollack, 1990] Recurrent Auto-Associative Memory (RAAM), Maskara and Noetzel's [Maskara, 1992] Auto-Associative Recurrent Network (AARN), and Williams and Zipser's [Williams, 1989] Real Time Recurrent Learning (RTRL) networks. Unlike locally recurrent networks, where the only time-delayed connection that a unit receives is from itself, in these networks time-delayed connections can come from any of the network's internal units. This gives this topology the computational power of finite state automata [Kremer, 1995], or if infinite precision units are used, the computational power of Turing machines [Siegelmann, 1991, Siegel-

© 2001 by CRC Press LLC

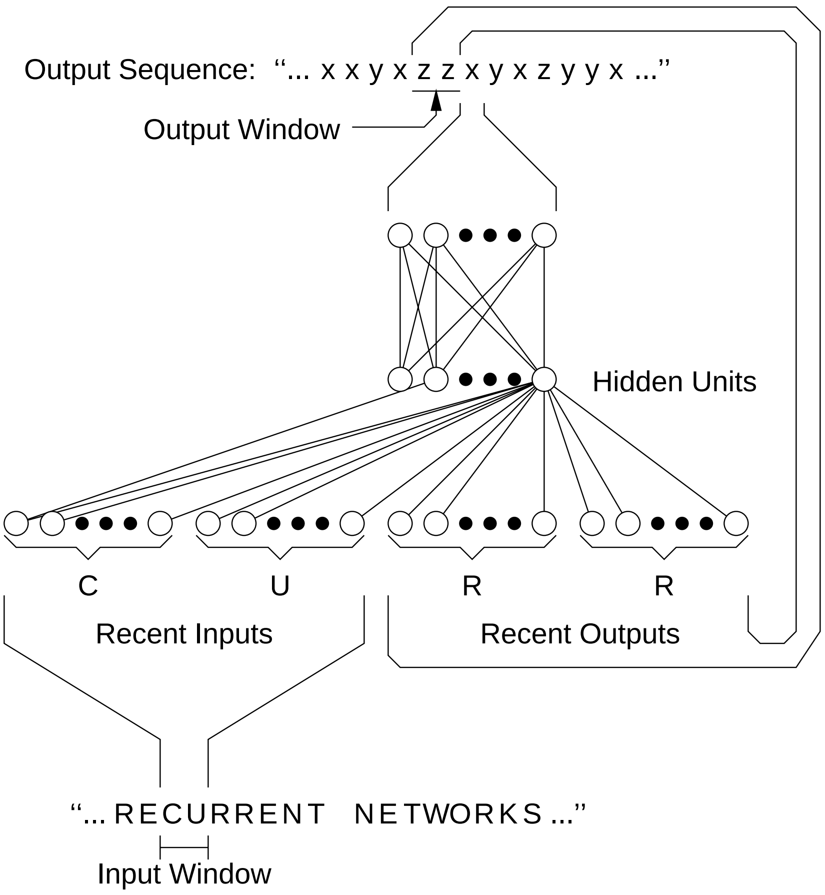
Output Sequence: ‘‘... x x y x z z x y x z y y x ...’’
Output Window
Hidden Units
C
U
R
R
Recent Inputs
Recent Outputs
‘‘... R E C U R R E N T   N E T W O R K S ...’’
Input Window

Figure 2. Neural Network IIR.

© 2001 by CRC Press LLC

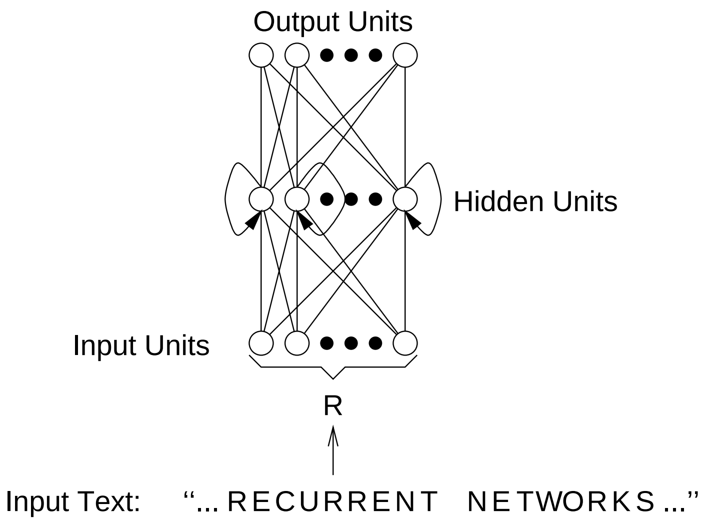

Figure 3. Example of a Locally Recurrent Network.

mann, 1992]. A variation on this approach has also been developed that uses second order connections between input and previous hidden and current hidden unit activations [Giles, 1990].

These four different network topologies have different representational capacities. It is important when selecting a topology to choose one which has the representational power to solve the task at hand, but not more representational power than necessary, because this will extend the search space of potential solutions which can make learning take much longer or make it intractable altogether.

E. LESSON 2.2: CHOOSE A LIMITED NUMBER OF HIDDEN UNITS

Another way, to limit the power of recurrent networks, is to limit the number of hidden units, which obviously constrains the kinds of computations the network can perform. It is fairly obvious that the types of constraints imposed by limiting hidden units in recurrent networks will not fall along the lines of the classical Chomsky hierarchy of languages. Instead, in recurrent networks, a hierarchy based on decision regions in the geometry of the input and internal representation spaces will form. These types of network-based hierarchies may even fall along lines which more closely resemble the distinctions between natural and artificial languages since the network-hierarchies are a consequence of a parallel processing architecture which may be considered more brain-like than the grammatical rules which distinguish symbolic language hierarchies.

© 2001 by CRC Press LLC

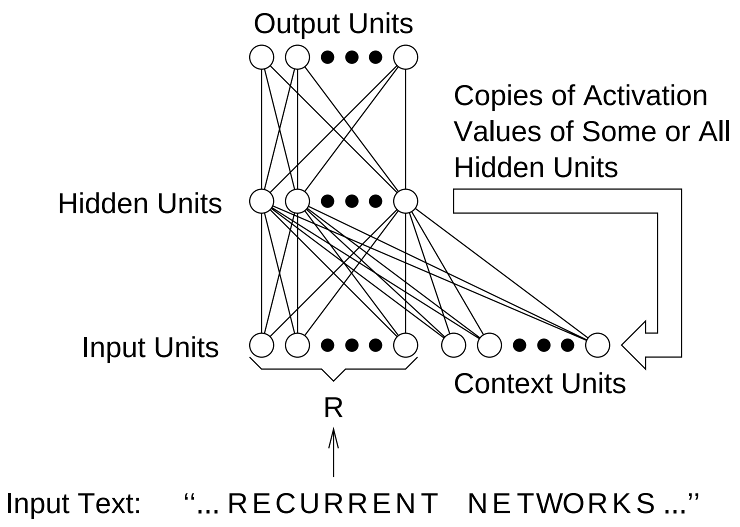
Output Units
Copies of Activation Values of Some or All Hidden Units
Hidden Units
Input Units
Context Units
R
Input Text: "... RECURRENT NETWORKS ..."

Figure 4. Example of a Network that Uses Old Hidden Unit Activations to Compute New Activations.

F. LESSON 2.3: FIX SOME WEIGHTS

Choosing a limited number of hidden units also effectively reduces the number of weights in the network. Since it is the weights which determine the computation performed, this will naturally constrain these computations. An alternative to limiting the number of weights is to fix the values of some of the weights in the network. This effectively reduces the degrees of freedom in the system, reducing the search space for the learning algorithm and thus offering a potential speed-up to learning. Of course, in order to be of use in solving a given problem, the fixed weights in the network should incorporate some *a priori* knowledge about the problem to be solved.

Fixing weights, however, cannot guarantee that the supplied *a priori* knowledge will actually be incorporated in the grammar induced by a recurrent network, because the trained weights in the network can overpower or nullify the contributions of the fixed weights. Suppose that the trained weights of the connections leading into node i in some recurrent network are much larger than the fixed weights leading into the same node. Since the signals transmitted through each connection are multiplied by the connection's weight and then summed together by node i, the effect of the fixed weights will be negligible compared to the effects of the larger trained weights. In this situation, the trained weights overpower the fixed weights.

Now suppose that all the connections leading out of node j are trained and have a very small weight after the training process. In this case any fixed weights leading into node j will affect the activation value of the node, but this activation

© 2001 by CRC Press LLC

value will be ignored by the rest of the network due to the small outgoing weights. In this sense, the trained weights nullify the effect of the fixed weights.

Of course, a network which ignores *a priori* knowledge in either of these two ways will further limit its representational capacity. That is, a recurrent network with n nodes will be able to represent a large class of languages. A recurrent network of the same size which has some fixed weights and uses those fixed weights to compute its behaviour will be able to represent a smaller class of languages. Finally, a recurrent network with n nodes which has some fixed weights but does not use these weights in its computation (either because they are overpowered or because they are nullified) will be able to represent the smallest class of languages.

Frasconi, Gori, Maggini, and Soda [Frasconi, 1995] have explored fixing network weights based on *a priori* knowledge about an isolated word recognition task to be solved. Specifically, they develop a network consisting of two separate networks with a common 1-Layer output function. One of the networks (called "K") consists entirely of fixed weights whose values are assigned based on the available knowledge. The other network (called "L") has adaptive weights whose values are learned based on training data. By using this modular approach, these authors are able to prevent the trained weights from overpowering the fixed weights. However, the output layer can still ignore the values of the state nodes in K by setting all weights originating from the K-memory to small values. We defer the discussion of the type of *a priori* knowledge used by the authors and how this knowledge is encoded into connection weights to the original papers [Frasconi, 1991, Frasconi, 1995].

Frasconi et. al.'s networks are able to achieve a recognition rate as high as 92.3% in empirical performance tests. The authors indicate that this is a significant achievement due to the fact that the task of isolated word recognition is complicated by the fact that the words used are composed only of vowel and nasal sounds. They further argue that their approach is more efficient than ones which do not use *a priori* knowledge. Unfortunately, the authors do not provide any empirical data comparing networks with *a priori* data to networks without *a priori* data.

From these considerations and empirical results, we can conclude that it is advisable to consider incorporation any knowledge of the kinds of solutions that we want our network to find into the connection weights of the network. A number of such encoding techniques have been developed for different kinds of networks, and the reader is referred to [Frasconi, 1991, Frasconi, 1995, Giles, 1992b, Giles, 1993] for detailed discussions of encoding.

G. LESSON 2.4: SET INITIAL WEIGHTS

While it is obvious that fixing weights in a recurrent network restricts the hypothesis space, it is less apparent that initial weights can also restrict the space. To recognize the latter fact, we must realize that the search of the hypothesis space in recurrent networks is usually governed by a gradient-descent algorithm. This implies that each candidate grammar considered during the search must have a smaller error value than the previous. But, since the initial weights of a network

© 2001 by CRC Press LLC

define a grammar and since that grammar is assigned an error value, it must be the case that all grammars with higher error values than the initial grammar are omitted from the search. Thus, the initialization of weights can serve to restrict the hypothesis space by causing all grammars with higher error values to be rejected outright. Figure 5 illustrates an initial set of weights (i.e., a point in weight space), a fictional error function, and those grammars which are not explored during the search algorithm (shaded grey).

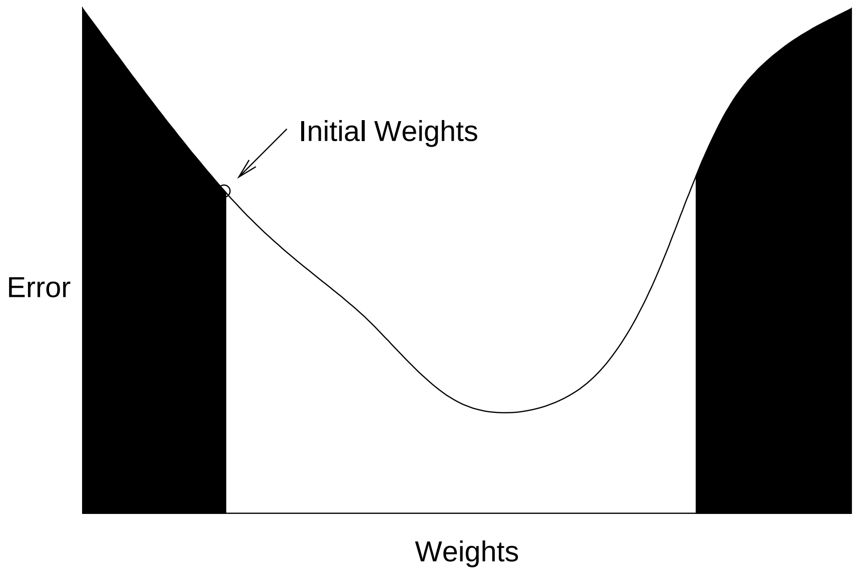

Figure 5. Initializing Weights to Limit Space.

It is interesting to note that "good" *a priori* knowledge will tend to significantly reduce the hypothesis space, while "bad" knowledge tends not to reduce the hypothesis space as much, because good *a priori* knowledge will tend to result in a network having a small error value. Since only those networks and grammars with even smaller error values are explored, the hypothesis space will tend to be greatly reduced. Conversely, bad *a priori* knowledge will tend to result in a network with a large error value. In this case there will be many recurrent networks and grammars having smaller error values and hence the hypothesis space will tend to remain large. This is an extremely useful property since it implies that good information will tend to have a large (positive) effect while bad information will tend to have very little effect.

There is, however, one serious drawback in choosing initial weights to restrict the hypothesis space: local minima in the error function. If the function mapping weight values to network error is non-monotonic, then it may be the case that to get to a smaller error value one must first travel though a region (in weight space) of larger error. Since the gradient descent algorithm travels only down the

© 2001 by CRC Press LLC

error gradient, such smaller error values can never be achieved. That is, the initial weights do not limit the hypothesis space to all networks with smaller error values, but rather only to those recurrent networks lying within the current basin of attraction. If the attractor at the bottom of this basin represents a local minimum (as opposed to a global minimum), then the hypothesis space will be unduly restricted to exclude the best solutions. This is illustrated in Figure 6.

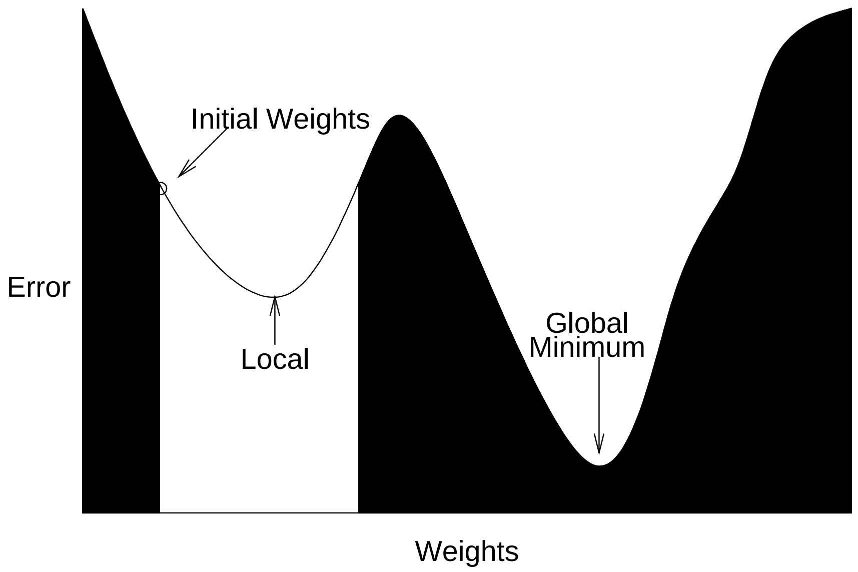

Figure 6. How initial weights can reduce the hypothesis space to exclude optimal solutions.

We can conclude that setting initial weights to an approximation of the solution is almost always desirable and one approach to overcoming the learning difficulties suggested by Gold and Pinker.

IV. LESSON 3: SEARCH THE MOST LIKELY PLACES FIRST

Another way to speed search (in general) is to order or bias the hypothesis space based on some heuristic. Suppose you are a habitual car key loser and that you keep track of where your keys turn up after each search. The results of such record keeping might be something like: coat pocket: 53%, hallway shelf: 27%, kitchen table: 16%, beside telephone: 3%, in refrigerator: 1%. If you know that most of the time the missing key has been located in your coat pocket, then it makes sense to begin your search there. That is, it is logical to order your hypothesis space and bias it in favour of the coat pocket. But just like a bad hypothesis space reduction can hinder search, a bad ordering can also impede an

© 2001 by CRC Press LLC

effective search. For example, using the hypothesis ordering designed for your car keys to find a pitcher of orange juice would clearly be very inefficient.

Just as hypothesis space restriction can be used to simplify the search for a grammar, hypothesis space ordering has also been applied to grammatical induction within the symbolic paradigm. In this case, a working hypothesis about the grammar (from the hypothesis space) is used as a starting point. Then, as new evidence about the grammar is presented in the form of training data, a change to the hypothesis is made. The nature of this change is defined by some heuristic. That is, certain types of hypothesis changes will be favored over other changes even if both are consistent with the training data. The chosen hypothesis change results in a new working hypothesis, and the process is repeated. Typically all of the possible hypothesis changes are evaluated and the resulting hypotheses are evaluated according to some weighting scheme. Then only the highest valued new hypothesis is selected as the new working hypothesis. This is analogous to a best-first search algorithm.

A weighting scheme could be based on complexity, for example, by assigning a weight inversely proportional to the number of auxiliary symbols (states) used by each grammar. This weighted selection process effectively orders the grammars of the hypothesis space. While searching for a grammar which is consistent with the training data, this ordering favors certain solutions over others. Ideally, good solutions to the problem to which the grammatical induction system is applied would be considered first and, thus, learning would be speeded.

While we have seen in the previous section that setting initial weights can restrict a hypothesis space, it is perhaps even more natural to think of setting initial weights as a technique for ordering the exploration of that remaining space. Obviously the initial weights define the first potential grammar which is explored by the induction algorithm. The exploration of subsequent grammars is governed by the learning algorithm. When the learning rate used by the gradient-descent algorithm is small, each grammar considered will lie close to the previous candidate grammar in the recurrent networks weight space. Since the output and state of a recurrent network are governed by functions continuous in the connection weights of the network, a small change in a connection weight will tend to result in a small change in output and state. This means that the exploration of the hypothesis space will proceed via similar grammars.

One advantage of using an ordering technique as opposed to a hypothesis space restriction technique is that there is often some uncertainty associated with *a priori* knowledge about a task. This implies that an irreversible decision, like eliminating certain grammars from consideration, is less desirable than an approach which can eventually ignore incorrect information. Setting the initial weights of a network can operate in this fashion, since even if the first weights are wrong and the network updates weights in small steps, the network will still be able to eventually explore other regions of the hypothesis space. This conclusion has been empirically verified by Giles and Omlin [Giles, 1993]. They initialized the weights of recurrent networks to implement one automaton, A, and then trained the recurrent networks to represent another (different) automaton, B. Despite the

© 2001 by CRC Press LLC

fact that this imposed an ordering on the hypothesis space which caused the network to explore automaton A first, the network was still able to eventually find and learn automaton B.

Specifically, Giles and Omlin [Giles, 1993] trained a recurrent network to implement a randomly generated 10-state finite-state automaton. Then, they initialized the weights of the automaton to encode a different randomly generated 10-state automaton. The authors discovered that, so long as the assigned weight values assigned were not too large (> 2), the networks were able to learn the correct automaton in spite of the "malicious" information provided by weight initialization. Of course, the authors also found that learning times were significantly longer for "malicious" information than for correct information. Giles and Omlin's results indicate that even if *a priori* knowledge is incorrect, an ordering scheme such as initializing weights can sometimes still find the correct solution.

V. LESSON 4: ORDER YOUR TRAINING DATA

The previous two sections discussed methods that could be used to speed the learning process before network training begins. This section describes a technique to provide information to the network during training. In the traditional grammar induction paradigm, the learner is required to identify a grammar based on a set of positive (and optionally a set of negative) example strings. Under input ordering, the data available to the learner consists not of a set of strings, but of a sequence of strings. That is, there is an order associated with the input data. If input strings are presented in a non-random order, then the position of a string within the sequence can represent an additional source of information about the grammar to be induced.

For an input ordering to be advantageous, two criteria must be met: (1) The presentation of a string s at time t must encode some information other than the mere fact that the string is a member (or not a member) of the language. (2) The learning system must be informed of the import of this encoded information and use it to limit or order the exploration of the remaining search space. Only when both of these criteria are met can a computational advantage be realized.

A. CLASSICAL RESULTS

We begin our discussion of input ordering by examining how input ordering works and what it can achieve. Gold [Gold, 1967] proposed a type of input string orderings which can improve the classes of grammars that can be induced using only positive input strings (text learning). This ordering scheme uses indirect negative information to learn languages which cannot be learned from positive information alone. This is done by using the absence of a string at a particular point in a sequence to infer that the string is illegal.

Suppose an order on all possible strings (grammatical and ungrammatical) is known to the learning system and this order defines how the environment provides the input data (e.g., alphabetical order). Note that there is an important distinction between knowing the order in which a sequence of strings is presented and the actual sequence of strings. An ordering defines a relation between all possible

© 2001 by CRC Press LLC

strings for a given alphabet, i.e., Σ^*, and thus defines where each string should belong (if it were legal), whereas the input sequence generally consists of only a subset of Σ^* and defines the actual set of legal strings.

Now assume that the induction environment presents all the grammatical strings to the learner according to the given order. Then, by omitting a string at the appropriate time, the environment essentially informs the learning mechanism that the given string is ungrammatical. Since this implies that the training set effectively contains both grammatical and ungrammatical strings, it is equivalent to informant learning as defined by Gold. Since Gold has already shown that primitive recursive languages are identifiable in the limit under informant learning, this class of languages must also be learnable under ordered text learning. While this strict sense of ordering is obviously an unrealistic idealization for practical grammar induction systems, Gold's work does point out the power that an ordering scheme can provide.

A less stringent ordering scheme has been proposed by Feldman [Feldman, 1972]. He showed that even an effective approximate ordering of the input strings could be used to convey indirect negative information. If there exists a point in time by which every grammatical sentence of a given length or less has appeared in the sample, then a learner capable of computing this point in time can also compute which sentences are not in the language (this could be the case in human language learning if children were spoken to in short sentences). Once again this is equivalent to a learner's being provided with both grammatical and nongrammatical strings, appropriately labeled.

The common thread to both of these techniques is the fact that the learner reacts differently to the same set of strings presented in differing orders. More specifically, strings which are presented early cause the learner to make certain assumptions about remaining strings which affects the order in which potential grammars are considered, or the size of the hypothesis space which is explored. More efficient and tractable learning can be accomplished by tailoring the learning algorithm and the input sequence to each other.

B. INPUT ORDERING USED IN RECURRENT NETWORKS

A simple ordering scheme which can be placed on strings is to sort them in order of increasing length. Das et al. [Das, 1993] have used a recurrent network training scheme whereby short simple strings are presented first, and progressively longer strings are presented as learning proceeds. They contend that "incremental learning is very useful when (a) the data presented contains structure, and (b) the strings learned earlier embody simpler versions of the task being learned" [Das, 1993], a well-known concept in machine learning theory. In this situation, the fact that short strings are presented early, together with the fact that these strings embody simple versions of later strings, implies that it is possible to use the strings which have already been presented to make certain implicit logical inferences about strings which have not yet been presented. A grammar induction system can be designed to use these inferences to dynamically reduce or re-order

© 2001 by CRC Press LLC

the space grammars it can induce.

For example, when a string of length n is presented as input to Das et al.'s system, it is possible to conclude that all strings which are shorter than n and have not yet been presented must be ungrammatical. This implies that these shorter strings will not be presented at a later point in time. In this sense, additional information about future strings (i.e., that they will not contain certain short strings) is transmitted by the ordered data. We will see shortly how a network learning system could function in this fashion.

Giles et al. [Giles, 1991, Giles, 1992, Giles, 1992a] and Miller and Giles [Miller, 1993] have used another simple ordering scheme: alphabetical ordering. If input strings are presented in strict lexicographic order, then the presentation of a string, s, implies that all lexicographically preceding strings which have not been presented must be ungrammatical, in the case of text learning, and must be of irrelevant (don't care) grammaticality, in the case of information learning. In this sense, an alphabetical presentation order can convey additional information (regarding the grammaticality of unpresented strings). Once again, a learning system which is tuned to this type of ordering in the sense that it restricts or orders the space of inducible grammars dynamically could perform better than a system in which input is not ordered. (Empirical data describing Giles et al.'s and Miller and Giles' results is described in the papers listed above.)

Both lengthening and alphabetical input orderings are very restrictive in the sense that they precisely prescribe the order of presented strings. For practical applications, it is often more desirable to use a less stringent ordering. Consider a case where input strings are presented in phases. In the first phase, all short strings, and only short strings, are presented. In later phases other strings are presented. We shall refer to this type of partial ordering as multi-phase uniform complete since the strings presented in the first phase are uniformly short and completely represented. A multi-phase uniform complete input ordering can provide additional information in the same sense that a lengthening input ordering does, with the exception that assumptions about strings which have not been presented can only be made at the end of a phase, as opposed to after each string. Giles et al. [Giles, 1990] have used a multi-phase uniform complete ordering to train recurrent networks. A similar ordering technique has been used by Elman [Elman, 1991a, Elman, 1991] who used a form of ordering to train his networks.

C. HOW RECURRENT NETWORKS PAY ATTENTION TO ORDER

We have now seen that the training environments used for recurrent networks sometimes contain additional implicit information (beyond the grammaticality of individual strings) in the form of string ordering. This represents one of the two components required for a more efficient learning system. The second component is a learner that uses the additional information. In this section we examine how input ordering affects the solutions explored and found by recurrent networks, thereby addressing this second component. Specifically, we examine two types of order sensitivity: engineered sensitivity and natural sensitivity.

© 2001 by CRC Press LLC

One way of ensuring that a learning system makes use of input ordering is to specifically design an induction algorithm around an ordering scheme. Since every symbolic algorithm “is equivalent to and can be ‘simulated’ by some neural net” [Minsky, 1967], it is not at all surprising that it is possible to realize such a hand-crafted algorithm in the form of a connectionist network. As an example, Porat and Feldman [Porat, 1991] have designed a connectionist network which implements an algorithm which induces FSA based on alphabetically-presented input strings. In order to implement the algorithm, however, the connectionist network requires an extremely complex control structure (compared to typical connectionist networks) and has both hardwired and mutable connections. Thus, the resulting learning system seems more like a connectionist-iterative learning hybrid than a purely connectionist architecture.

An alternative to designing the learning system to accommodate a particular input order is to design the input order to accommodate the learning system. This is typically done in recurrent networks, where network design is based on principles such as simplicity, homogeneity, and local processing. Having designed the network according to these principles, the researcher can only ensure cooperation between input order and learner by adjusting the input order to suit the network’s own natural sensitivities to this order. In a sense, the researcher has assumed part of the burden of learning the language. It turns out that the order of pattern presentation affects recurrent network (and other network) learning greatly, because initial weight changes in a network can draw solutions toward a certain local minimum from which the recurrent cannot later escape. This occurs because recurrent networks do not perform true gradient descent.

Recall that, in order to efficiently approximate the gradient, weight adjustments, $\Delta w_{ji}(t)$, are made piecewise over time. This implies that the component of the gradient caused by a pattern presented at time t is computed after the weight adjustment caused by the pattern at time $t-1$ has been made. This, in turn, means that successive weight adjustments are not commutative. To better understand the implications of this fact, we consider a simple example. Suppose we have a language consisting of only two training strings. Suppose also that the network error, for each of these two strings is given in Figure 7a and b, and the total error for both strings is given in Figure 7c). Now suppose that the network’s initial weights and corresponding errors are represented by the point labeled “B” in all three graphs. Clearly if the network is first trained only on the string whose error function is depicted in (a), then the network’s weights will move to the point labeled “C.” Subsequent training with the second string will keep the network’s weights at “C” since it represents a local minimum in the second string’s error space. By contrast, if the network is first trained using only the second string, (b), then the network will converge to point “A.” Again, subsequent training will not change the weights in the network. Thus, it is clear that the order of string presentation during training limits the hypothesis space (range of weights) which is considered in searching for an error minimizing solution during later string presentations.

While it is easy to see that input ordering affects hypothesis space search during learning, it is much more difficult to identify the ideal ordering scheme for a

© 2001 by CRC Press LLC

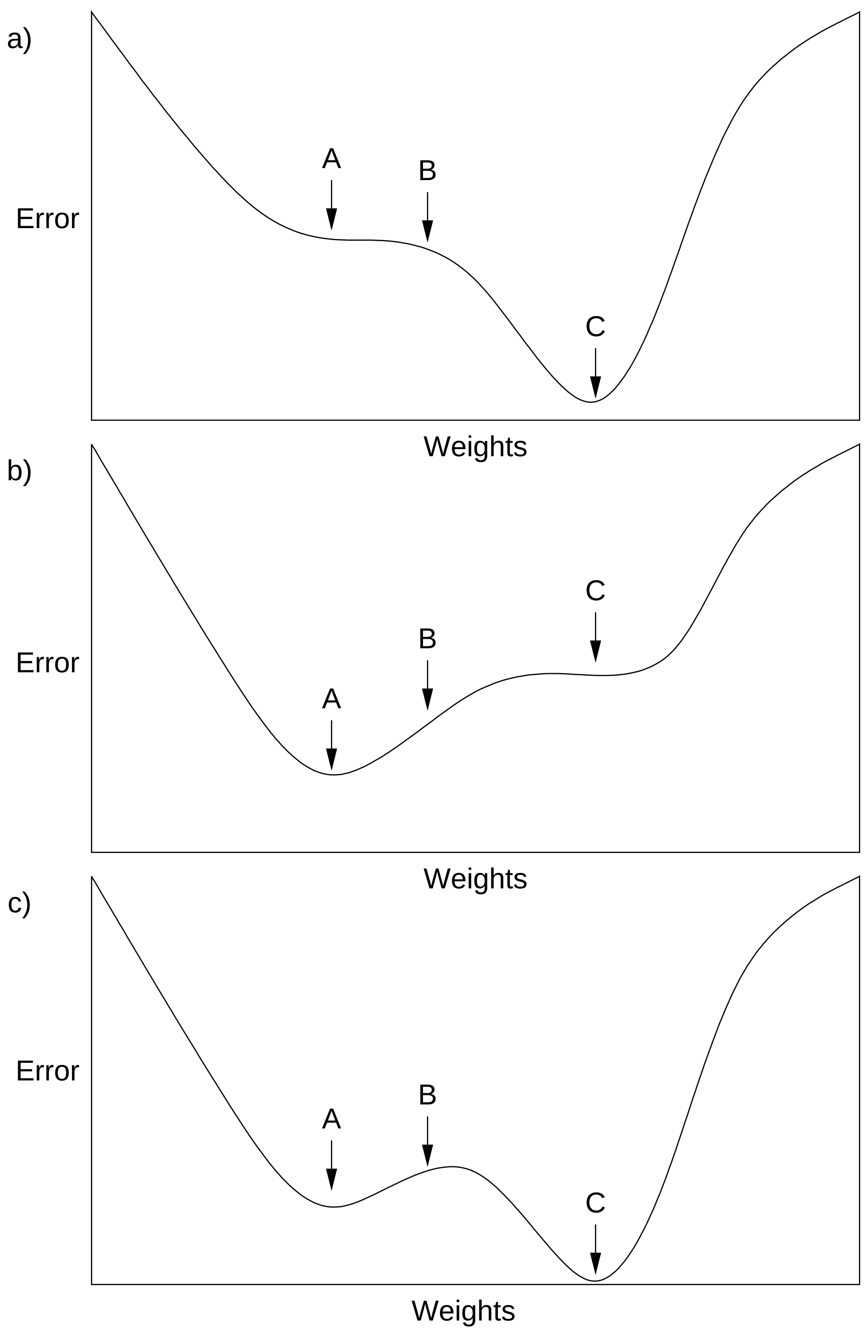

Figure 7. Why Input Order Matters.

© 2001 by CRC Press LLC

recurrent network. In the example above, presenting the string (a) before string (b) will restrict the range of weights to include the global minimum of the error space. By contrast, presenting string (b) prior to string (a) also restricts the range of weights, but the restricted range does not include the global minimum of the error space. Thus, in this simple example, it is important to present string (a) first.

In more general terms, it is always best to present strings whose error functions have local minima at the same points in weights space as the total error function has global minima. Since the presentation of a string adjusts the weights of the network so that the string's error is reduced, the network's weights will approach a local minimum in the presented string's error function. Ideally, this local minimum in the string's error function will correspond (or lie close to) a global minimum in the total error. Since the specific strings satisfying this condition depend entirely on the language to be learned by the recurrent network, we cannot identify a general ideal ordering scheme. Instead we turn to the empirical evidence to show that the ordering schemes described above do in fact correspond to the natural sensitivities to input order in recurrent networks.

Das et al. [Das, 1993] compared training recurrent networks with a lengthening input ordering to training the same recurrent networks with a random ordering of strings. They observed a 50% reduction in training time for the lengthening ordering scheme. Giles et al. [Giles, 1992a] also observed an improvement in training times when they presented strings in alphabetical order and concluded that "the sequence of strings presented during training is very important and certainly gives a bias in learning" [Giles, 1992a] and that "training with alphabetical order . . . is much faster and converges more often than random order presentation" [Giles, 1992a].

While the performance improvements realized by the ordering schemes of Das et al. [Das, 1993] and Giles et al. [Giles, 1992a] took the form of accelerated learning, Elman [Elman, 1991a, Elman, 1991] used ordering to learn an otherwise unlearnable grammar. In two learning experiments, Elman's multi-phase consistent complete ordering approach was used after previous attempts to train the network on the entire data set (complex and simple sentences) failed. In both cases, Elman found that, "when the network was permitted to focus on the simpler data first, it was able to learn the task quickly and then move on successfully to more complex patterns" [Elman, 1991a]. This evidence clearly shows that input ordering can be used in the connectionist domain (just as it has in the symbolic paradigm) to improve learning efficiency and tractability.

VI. SUMMARY

In this chapter, we have examined the problem of language learning as a special case of training recurrent networks. By examining results from the field of grammatical induction from the past 30 years, we have discovered 4 useful lessons that can be applied to training recurrent networks. The first lesson is that learning languages is hard regardless of paradigm used. Since language learning is one of the simplest cases of the kinds of learning that recurrent networks are tasked with,

© 2001 by CRC Press LLC

we must infer that learning in recurrent networks is difficult in general. From this first lesson, we turn our attention to making learning easier.

The second lesson revealed that, while it is tempting to select the most representationally powerful computational tool possible for language learning tasks, this is a dangerous choice since the representational power is inversely related to the effectiveness of learning. Thus, we will often want to select a smaller search space. There are 4 ways of accomplishing this in recurrent networks: (1) selecting an appropriate network topology, (2) selecting an appropriate number of internal units, (3) fixing some weights with appropriate values, and (4) setting the initial weights to restrict the search space.

The third lesson showed that ordering the exploration of the hypothesis space can also be very advantageous. This can be accomplished by setting initial weights. Empirical evidence revealed that this is a very effective technique to speed learning which does not doom the training process even if malicious incorrect data is used.

The fourth lesson focussed on the effect of ordering training data. This technique represents a method for indirectly providing information about which strings are not in the language. A simple example revealed that recurrent network based language learners are capable of using string ordering to effect the learning process.

While language learning is only one potential application of recurrent networks, it is an informative one. These examinations have revealed effective techniques for language learning derived from previous research. In most cases, these techniques have already been implemented as heuristics for improving the training of recurrent networks with significant success. This chapter serves to ground these techniques in a formal theory, thereby giving insights into why they work and why, how, and when they should be applied. An extended version of this work can be found in Kremer [1996a].

REFERENCES

Chomsky, N., *Aspects of the Theory of Syntax,* MIT Press, Cambridge, 1965.

Das, S., Giles, C. L., Sun, G.-Z., Using prior knowledge in a NNPDA to learn context-free languages. In *Advances in Neural Information Processing Systems*, Hanson, S. J., Cowan, J. D., Giles C. L., Eds., Morgan Kaufmann Publishers, 5, 65, 1993.

Elman, J., Finding structure in time, *Cognitive Science*, 14, 179, 1990.

© 2001 by CRC Press LLC

Elman, J., Incremental learning, or the importance of starting small, Tech. Rep. CRL Tech Report 9101, Center for Research in Language, University of California at San Diego, La Jolla, CA, 1991.

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

Feldman, J., Some decidability results on grammatical inference and complexity, *Information and Control*, 20, 244, 1972.

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

Frasconi, P., Gori, M., Maggini, M., Soda, G., A unified approach for integrating explicit knowledge and learning by example in recurrent networks, In *1991 IEEE INNS International Joint Conference on Neural Networks – Seattle*, 1, IEEE Press, 811, 1991.

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

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

Giles, C., Sun, G., Chen, H., Lee, Y., Chen, D., Higher order recurrent networks & grammatical inference. In *Advances in Neural Information Processing Systems*, 2, Touretzky, D. S., Ed., Morgan Kaufmann Publishers, 380, 1990.

Giles, C. L., Chen, D., Miller, C., Chen, H., Sun, G., Lee, Y., Second-order recurrent neural networks for grammatical inference. In *1991 IEEE INNS International Joint Conference on Neural Net works – Seattle*, 2, IEEE Press, 281, 1991.

Giles, C. L., Horne, B., Lin, T., Learning a class of large finite state machines with a recurrent neural network, *Neural Networks*, 8(9), 1359, 1995.

Giles, C. L., Miller, C. B., Chen, D., Sun, G. Z., Chen, H. H., Lee, Y. C., Extracting and learning an unknown grammar with recurrent neural networks. In *Advances in Neural Information Processing Systems*, 4, Moody, J. E., Hanson, S. J., Lippmann, R. P., Eds., Morgan Kaufmann Publishers, Inc., 317, 1992.

© 2001 by CRC Press LLC

Giles, C. L., Omlin, C., Inserting rules into recurrent neural networks. In *Neural Networks for Signal Processing II, Proceedings of The 1992 IEEE Workshop* Kung, S., Fallside, F., Sorenson, J. A., Kamm, C., Eds., IEEE Press, 13, 1992.

Giles, C. L., Omlin, C., Extraction, insertion and refinement of symbolic rules in dynamically-driven recurrent neural networks, *Connection Science*, 5(3,4), 307, 1993. Special Issue on Architectures for Integrating Symbolic and Neural Processes.

Gold, E. M., Language identification in the limit. *Information and Control*, 10, 447,1967.

Horne, B. G., Hush, D. R., Bounds on the complexity of recurrent neural network implementations of finite state machines. In *Advances in Neural Information Processing Systems*, 6, Cowan, J. D., Tesauro, G., Alspector, J., Eds., Morgan Kaufmann Publishers, Inc., 359, 1994.

Kohavi, Z., *Switching and Finite Automata Theory*, second ed., McGraw-Hill, Inc., New York, 1978.

Kolen, J. F., Kremer, S. C.,, Eds. *A Field Guide to Dynamical Recurrent Networks*, IEEE Press, To Appear.

Kremer, S. C., On the computational power of Elman-style recurrent networks, *IEEE Transactions on Neural Networks*, 6(4), 1000, 1995.

Kremer, S. C., *A Theory of Grammatical Induction in the Connectionist Paradigm*, PhD thesis, University of Alberta, 1995a.

Kremer, S. C., Identification of a specific limitation on local-feedback recurrent networks acting as Mealy/Moore machines, *IEEE Transactions on Neural Networks*, 10(2), 433, 1999.

Lang, K. J., Waibel, A. H., Hinton, G., A time-delay neural network architecture for isolated word recognition, *Neural Networks*, 3(1), 23, 1990.

Lapedes, A., Farber, R., Nonlinear signal processing using neural networks prediction and system modelling, Tech. Rep. LA-UR-262 or LA-UR87-2662, Los Alamos National Laboratory, Los Alamos, 1987.

Maskara, A., Noetzel, A., Forced learning in simple recurrent neural networks. In *Proceedings of the Fifth Conference on Neural Networks and Parallel Distributed Processing*, 107, 1992.

© 2001 by CRC Press LLC

Miller, C., Giles, C. L., Experimental comparison of the effect of order in recurrent neural networks, *International Journal of Pattern Recognition and Artificial Intelligence*, 7(4), 849, 1993. Special Issue on Neural Networks and Pattern Recognition.

Minsky, M., *Computation: Finite and Infinite Machines*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1967.

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

Pinker, S., Formal models of language learning, *Cognition*, 7, 217, 1979.

Pollack, J., Implications of recursive distributed representations. In *Advances in Neural Information Processing Systems*, 1, Touretzky, D., Ed., Morgan Kaufmann, 527, 1989.

Pollack, J. B., Recursive distributed representations, *Artificial Intelligence*, 46, 77, 1990.

Porat, S., Feldman, J., Learning automata from ordered examples, *Machine Learning*, 7(2-3), 109, 1991.

Rumerlhart, D., Hinton, G., Williams, R., Learning internal representation by error propagation, In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, McClelland, J. L., Rumelhart, D., and the P.D.P. Group, Eds., 1: Foundations. MIT Press, Cambridge, MA, 1986.

Sejnowski, T. J., Rosenberg, C. R., NETtalk: a parallel network that learns to read aloud, Tech. Rep. JHU/EECS-86/01, John Hopkins University Electrical Engineering and Computer Science, 1986.

Siegelmann, H., Sontag, E., Turing computability with neural nets, *Applied Mathematics Letters*, 4(6), 77, 1991.

Siegelmann, H., Sontag, E., On the computational power of neural nets, In *Proceedings of the Fifth ACM Workshop on Computational Learning Theory*, ACM, 440, 1992.

Siegelmann, H., Sontag, E., On the computational power of neural nets, *Journal of Computer and System Sciences*, 50(1), 132, 1995.

© 2001 by CRC Press LLC

Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., Lang, K., Phoneme recognition using time-delay neural networks, *IEEE Transactions on Acoustics, Speech and Signal Processing*, 37(3), 328, 1989.

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

Williams, R. J., Zipser, D., A learning algorithm for continually running fully recurrent neural networks, *Neural Computation*, 1(2), 270, 1989.

© 2001 by CRC Press LLC