NEURO DYNAMICS *and* PSYCHOLOGY

EDITED BY

M. Oaksford and
G.D.A. Brown

Department of Psychology,
University College of North Wales
Bangor

ACADEMIC PRESS
Harcourt Brace & Company, Publishers
London Boston San Diego New York Sydney Tokyo

Chapter 11

Sequence Processing with Recurrent Neural Networks

Nick Chater and Peter Conkey

11.1 Introduction

A large number of interesting cognitive tasks are sequential in nature; they involve the integration and use of information over an extended period of time. Examples include the perception and production of speech, the control of complex movements, and the integration and analysis of perceptual input over successive time-frames. Such tasks pose important challenges for the use of neural networks in modelling aspects of cognition.

The most widely used neural net architecture is composed of ordered layers, such that the units in each layer connect only to units in successive layers. Thus, in such "feedforward" networks, activation flows unidirectionally from the initial "input" layer to the final "output" layer. Such networks are typically trained on a finite set of input–output pairs. The most popular method of training such networks is to use the back-propagation algorithm (Rumelhart, Hinton & Williams, 1986a). During learning, for each input, the desired "target" output is compared with the actual output of the network, and the disparity between the two is computed. The overall performance of the network is measured by the square of the disparities between the actual and desired output of the network, summed over each input–output pair. This error measure is then "back-propagated" through the network, in order to discover to what extent the value of each of the network's units and weights influence the disparity between actual and target output (that is, the partial derivative of the error is calculated with respect to both the activation level of each unit, and the weight on each link). These partial derivatives specify how the values of the weights may be adjusted in order to reduce the squared error. By performing gradient descent in error space, the back-propagation algorithm reconfigures the network so that the difference between the desired and actual output of the network is gradually reduced. A variety of related gradient descent methods have been explored, which use rather more complex minimisation strategies than the steepest descent of back-propagation (see, for example, Fahlman, 1988; Webb, Lowe & Bedworth, 1988).

Back-propagation is able to train a variety of interesting input–output mappings. However, since the output of a feedforward network is influenced only by its current input, at first sight it seems that such networks must be inappropriate for dealing with sequential structure in time. Responding to sequential structure requires the output of the network to be a function of not just its current input, but of the history of recent inputs. Nonetheless, feedforward networks trained by the back-propagation learning algorithm have been widely applied to the processing of sequential inputs, using a variety of methods. Following a discussion of possible approaches and of the relationship between them, we focus on the performance and limitations of the particular approach that we shall term the "copy-back" strategy (Elman, 1988, 1990; Jordan, 1986a,b). We shall focus on small artificial problems so that analysis is relatively straightforward, and will also consider how the copy-back regime learns sequences generated by a very simple artificial grammar.

11.2 Back-propagation and finding sequential structure

11.2.1 Explicitly representing the past

We saw above that the output of a feedforward network is purely a function of its current input, and noted that this would appear to rule out the possibility that such networks can respond selectively to temporal sequences. One way of circumventing this problem is simply to decouple the notion of state (which composes the sequences that the network is to respond to) from the pattern of activity over the input units of the network. Thus, in order to give the network access to m states at successive times, the input units can be divided into m blocks, each representing the state at a particular time. This most straightforward approach is used in a variety of models (Elman, 1990, cites Cottrell, Munro & Zipser, 1987; Elman & Zipser, 1988; Hanson & Kegl, 1987). Elman points out three drawbacks with this approach to this way of representing time:

(1) The past states of the sequence must be buffered before being presented simultaneously, and biology provides no clear examples of this sort of buffering. More generally, it seems undesirable that past states must be explicitly remembered and manipulated by any additional buffering processes which must be prespecified, rather than learnt by the network.

(2) The amount of past state history that the network takes into account is hardwired, in the choice of the number of past states that are put into the input, rather than learnt by the network.

(3) Such an approach does not capture the invariance of the same temporal pattern presented at different absolute times. Elman gives an example in which the state at each time-step is represented by the activation of a single input unit. Then the following pattern vectors might represent the same pattern, shifted in absolute position in time:

.0 .1 .3 .7 .9 .7 .3 .1 .0 .0 .0

.0 .0 .0 .0 .1 .3 .7 .9 .7 .3 .1

However, from the point of view of a network, these input patterns are very dissimilar (consider, for example, their dot-product). A network may, of course, learn to categorise both patterns as instances of the same pattern if separately presented with examples of both. However, crucially, it will be unable to learn to generalise its recognition of a pattern that it has only seen in some of the possible positions to an unseen position. This is a special case of the general problem of position invariant recognition, which is very difficult to learn using feedforward networks trained with back-propagation.

Elman uses these three limitations to motivate his move to a recurrent network architecture — that is, to relax the restriction that connections must be feedforward, thereby permitting activation from past inputs to recirculate in the network so that this can influence the current output. Rather than considering this scheme directly, however, let us show how attempts to overcome certain of the problems that Elman identifies have led to intermediate approaches to the representation of time. This will provide a convenient basis for making comparisons with those methods of training networks which use the back-propagation algorithm. First, let us consider a very slight variant of the present approach which addresses the third, and most serious problem.

11.2.2 Using temporal windows

In the approach above, we have been implicitly assuming that the input units are devoted to the past states at various absolute times (perhaps at times 1, 2, ...m). If the past states are encoded in this way, the third problem indeed arises — the network will learn patterns by their absolute temporal position, and be unable to recognise a pattern that has been shifted in absolute position. Since the inputs represent time absolutely, it is hardly surprising that the outputs will depend on absolute rather than relative temporal position. Alternatively, the input units may represent states in a moving temporal "window." That is, using a window of m time-slots, at time t, the input units may be devoted to representing the states at times t, t–1, t–2..., t–m+1. At each time-step the network produces an output based on the current input. Now the output of the network is sensitive to relative rather than absolute temporal structure. As the temporal window passes over the pattern, the same sequence of "frames" will be produced, regardless of the absolute temporal location of the pattern. Sejnowski and Rosenberg's (1987) NETtalk is a well-known example of a system which employs such a strategy. The network is trained to produce the phoneme which "corresponds" to the middle letter of a seven letter window of text.

11.2.3 Delay links

Although the use of a moving temporal window addresses the third, and most important problem that Elman raises, of responding to relative rather than absolute temporal position, the other two problems remain. The past states must be remembered and manipulated externally, and the amount of past history that the network may take into account is prespecified. The first problem may be tackled by generalising the moving window strategy — introducing delay links.

Using temporal windows, the hidden units receive inputs from sets of input units which represent each of, say, n states. This requires that the present state and n–1 past states are stored and imposed on the appropriate input units. However, it is not strictly necessary to reduplicate units for storing the input at each time-step. An alternative strategy is to employ delay links — i.e., links for which the output values are a function of the input values at some previous time-step, rather than the present time-step. So an alternative to using n sets of input units to feed into the hidden layer of the network is to use a single set of

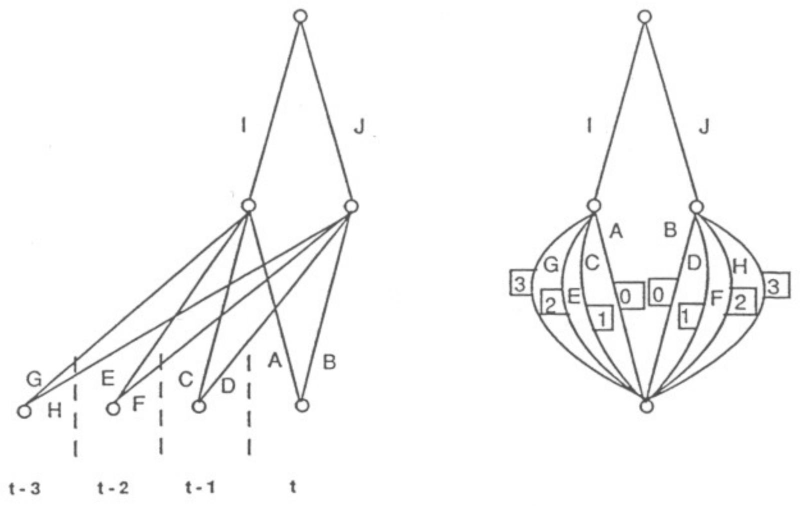

Figure 1. A simple temporal window network (left) with four input nodes, one for each of four times, two hidden nodes and one output node. On the right is the corresponding delay-link network with just a single input node. The delay values on the delay links are indicated inside the boxes on the links. The labels A–J on the links indicate the correspondence between links in the two networks.

input units for which the connection between input and hidden unit is present with delay 0, 1...n–1 (Figure 1). Of course, the use of delay links does not mean that the values of past need not be stored. Rather, they are stored in delay links rather than in the activation of the input units. The apparent computational economy in having fewer units is thus more apparent than real. In particular, the number of connections and biases in the network is not reduced.

The back-propagation algorithm may be used to train the network with delay links in precisely the same way that it trains the corresponding "moving window" feedforward network — indeed, the delay-link network may be thought of simply as a notational variant of the moving window counterpart. However, thinking in terms of delay links suggests natural generalisations outside the domain of the "moving window" approach. In particular, instead of simply having delay links between input and the first layer of hidden units,

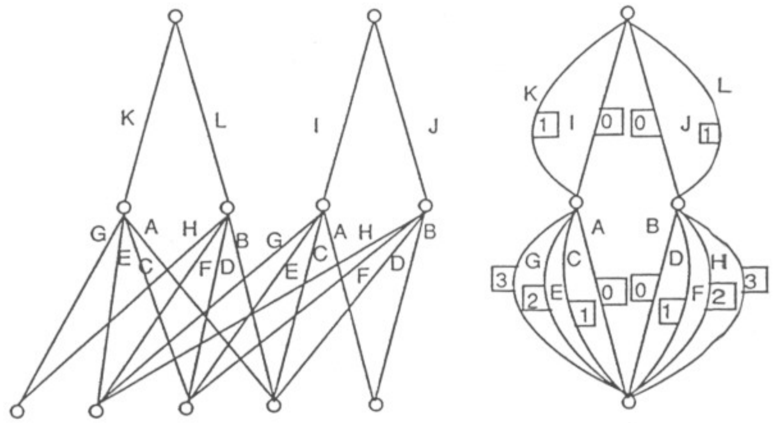

Figure 2. A simple delay-link network (right) with one input node, two hidden nodes and one output node. There are delay links with values of between 0 and 3 between the input and hidden layers, and delay links with values 0 and 1 between the hidden and output layer. This implies that the output of the network is determined by the activity of the input units over five time-steps in total. On the left is the corresponding feedforward network. The labels A–L on the links indicate the correspondence between links in the two networks.

delay links may be used between hidden layers, and between hidden and output layers.

This use of delay links to extract structure from a time-varying input has been used to recognise phonemes in noisy environments (Waibel *et al.*, 1987). It is interesting to note how a network with more than one layer of delay links may be "unfolded" into a standard feedforward network (Figure 2).

Notice that in the feedforward counterpart, many of the links must be constrained to be equal. Thus the time-delay notation is more compact since it does not require that the same weights be copied at several locations. The unfolded feedforward network notation is, however, useful for comparing the various ways in which networks may be designed to be sensitive to structure in time, as we shall see below.

11.2.4 Simple recurrent networks

The remaining point that Elman raises is that ideally a network should be able to decide how large a temporal window it requires to solve a particular problem. In the networks that we have considered so far, the degree to which the network is able to look back in time is, by contrast, fixed by the architecture of the network (the size of the moving input window or time-delay on each of the delay links). What is required is some way of allowing the learning algorithm which structures the developing network to control the extent to which past states are preserved. Further, the learning algorithm should also be able to determine which aspects of the past input states are preserved and which are discarded.

Even in the approaches that we have discussed so far, the learning algorithm has some control over which aspects of the past input are taken into account. In all these cases a fixed number of past states are connected so that they may influence the output of the network. However, if, say, only the most recent three states are actually useful in determining the appropriate output, then the connections to the other redundant states will typically be small and play no role in determining actual output after learning has taken place. Similarly there is a sense in which the network is able to determine which aspects of past input states are to play a role in determining output and which are not. If, for example, each state consists of a two-bit vector, only one bit of which is relevant to the output at future times, then the connections to the relevant bit will be appropriately adjusted by the learning algorithm, and connections to the irrelevant bit will be turned off. However, this is rather an inefficient procedure. The entire past input history (over some long time period) must be remembered, and the network learns to pay attention to the relevant parts of this history and to ignore the irrelevant parts. It would be more satisfactory to have the network choose from the outset which aspects of which past states should be remembered. This suggests that the memory for the past states should not be fixed, but adjusted by the network itself. It is interesting to consider how this might be achieved.

Suppose that we employ a scheme in which the input units encode only the present states. If the network has a conventional feedforward architecture, then its output will be determined only by the present input states, since the activity generated by an input state percolates through the network at each time-step. However, if the network has recurrent connections within its architecture, activation from past inputs will "recirculate" around the network, thus enabling it to have a continuing influence on the output units at future time-steps. For such a network, the output will be a function of the whole input history, rather than simply the previous input.

To the extent that the recurrent connections mediate the influence of previous states on current output, they constitute a "memory" for that input. Crucially, unlike the prespecified memories utilised in the network models above, a memory implemented by recurrent connections may be adjusted dynamically by a learning algorithm as the network evolves. The adjustment of the recurrent connections may thus control both how far back the network "remembers" and which aspects of past state are either stored or discarded as irrelevant. In principle, this approach is an extremely attractive way of allowing networks to learn structure in time. The time-varying signal is presented over a single set of input units, and the network must learn to configure itself so that it recirculates the information from previous inputs that are required to produce the appropriate output. There is no need for past inputs to be stored in extra sets of input units or delay links. (Although such a network does not require that past input values are stored when it is running, it may require these in order to adjust the weights appropriately, a point to which we shall return below.)

Such simple recurrent networks (SRNs) were developed by Jordan (1986a,b) and Elman (1988) and provide a powerful tool with which to model the learning of many aspects of linguistic structure (for example, Elman, 1990, 1991; Shillcock, Levy & Chater, 1991; Weckerly & Elman, 1992) and there has been some exploration of their computational properties (Chater, 1989; Cleeremans, Servan-Schreiber & McClelland, 1989; Servan-Schreiber, Cleeremans & McClelland, 1991). The presence of recurrent connections allows past activation to influence current output, which means that output can respond to sequential structure in the input. The extent to which such networks can be taught to learn interesting sequential structure depends on the learning algorithm employed. A natural approach is to apply the back-propagation training algorithm which has proved so successful in training non-recurrent feedforward networks to learn interesting static input–output patterns.

Below, we shall discuss various ways in which this approach to training neural networks to learn sequences can be followed, and discuss what is learnt in a simple artificial language learning experiment. The structure of the discussion is as follows. First we discuss a number of ways in which the back-propagation algorithm can be adapted to train recurrent networks to learn sequences, concentrating on two options, Elman's (1990) "copy-back" scheme, and back-propagation through time (Rumelhart, Hinton & Williams, 1986a,b). We note that there are theoretical reasons to suppose that the copy-back regime

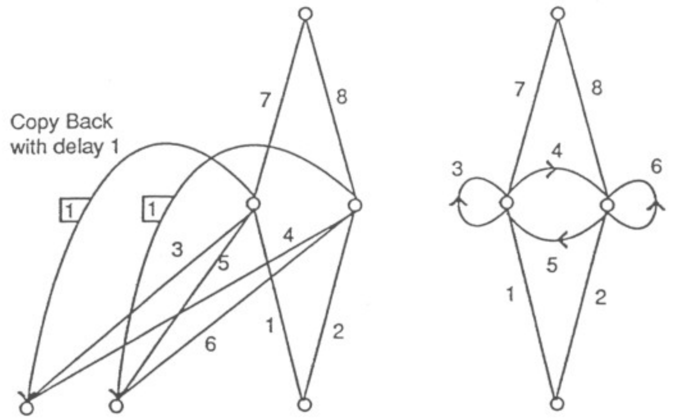

Figure 3. A simple recurrent network (right) with one input node, one output node, and two hidden nodes with recurrent connections. Left is shown the corresponding copy-back network. The labels 1–8 on the links indicate the correspondence between links in the two networks. The "copy" links (with delay value boxes) are simply a mechanism for ensuring that the copy units have the pattern of activation generated by the hidden units at the previous time-step. These links are not modifiable connections but have a fixed weight of 1, and a time delay of 1. Thus the copied pattern is an undistorted version of the previous hidden unit pattern. The "copy" units are linear, with a bias of zero.

will learn less well, and this conclusion is borne out in our simulations. However, the copy-back approach is computationally inexpensive and has provided impressive results in a number of language processing tasks. In the next section, we investigate the scope of this method further, and follow Elman in investigating the nature of the hidden unit representations developed for a network which learns to predict the next element in sequences generated by a simple grammar. A number of very different measures over the hidden units are found to generate very similar syntactic/semantic clustering. These clusters are also shown to be implicit in the statistics of the sequences learnt. This suggests that network performance can usefully be analysed in terms of the statistical structure of the input sequences, and that the applicability of SRNs to real natural language data can be assessed by analysing relevant aspects of its statistical structure.

11.3 Training simple recurrent networks

11.3.1 Copy-back or back-propagation through time?

Adapting back-propagation to a recurrent network can be thought of in two stages. First, the network is "unfolded" into a feedforward network which has the same behaviour, and then this feedforward network can be trained in the standard way. There are many ways in which this unfolding can be achieved.

The most popular method involves unfolding the network by providing an additional input — the "context" units — which corresponds to the values of the hidden units at the previous time-step (Elman, 1990) (Figure 3). The context units are dependent on the previous inputs, amongst which is the previous value of the context units. Hence the behaviour of the network is influenced not just by the current input but by the sequence of past inputs. While activation is propagated forwards through the network from arbitrarily far back in time, error is only propagated back to the context units.

An alternative approach is to unfold the network through several time-steps (Rumelhart *et al.*, 1986a,b) so that each weight has several "virtual incarnations" and to back-propagate through the resulting network (Figure 4). The overall weight change is simply the sum of the changes recommended for each incarnation. This "back-propagation through time" can in principle be back-propagated through the entire training history of the network (Rohwer, personal communication) but is typically implemented by unfolding through a small number of time-steps. The copy-back scheme can be viewed as a special case of back-propagation through time, in which the back-propagation of error stops at the first copy of the hidden units — the context units. (There are also a number of other ways of training such networks, such as Williams & Zipser (1989) and Zipser (1990) who show how explicitly unfolding the net can be avoided; Fahlman (1991) and Mozer (1988) who concentrate on training nets where only self weights on the hidden units can be recurrent; and Pearlmutter (1990) who shows how to extend this method to continuous time tasks.)

The more the network is unfolded, the better the approximation of the feedforward network to the underlying recurrent network, and the better the network learns to respond to sequential material. The minimal unfolding embodied in the copy-back scheme should therefore produce the poorest learning, although it uses the least computational resources. This is borne out in the comparative studies below.

A natural assumption is that the number of steps back that error is propagated will precisely fix the number of previous steps the network can

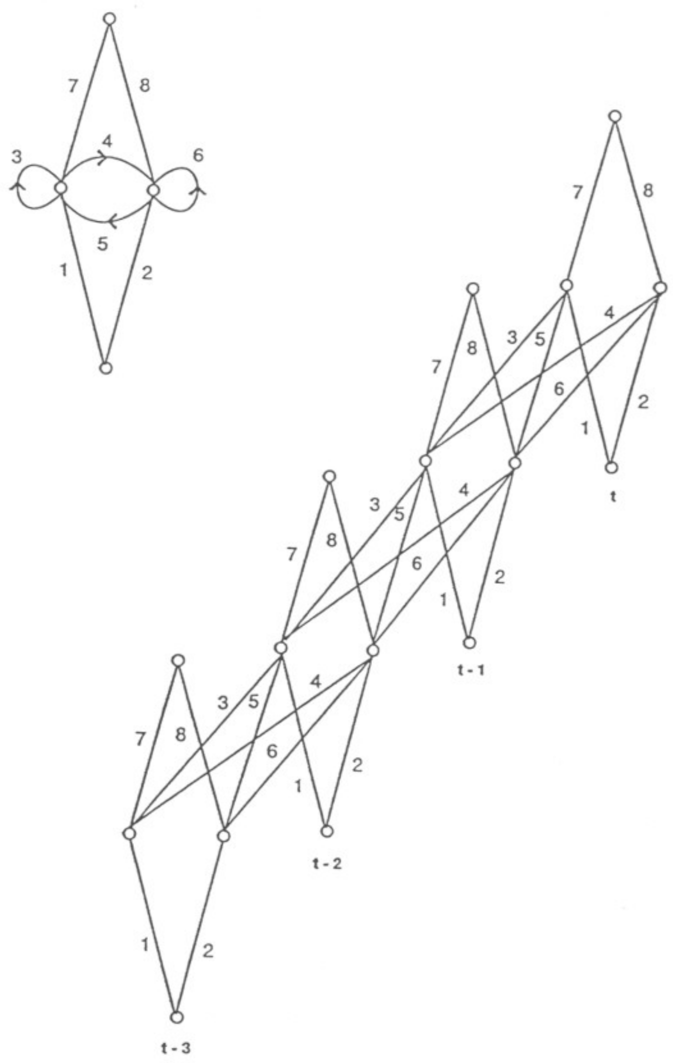

Figure 4. A simple recurrent network (left) with one input node, one output node, and two hidden nodes with recurrent connections. On right is shown the corresponding feedforward network unfolded for four time-steps. The labels 1–8 on the links indicate the correspondence between links in the two networks — each link with the same label has the same weight value.

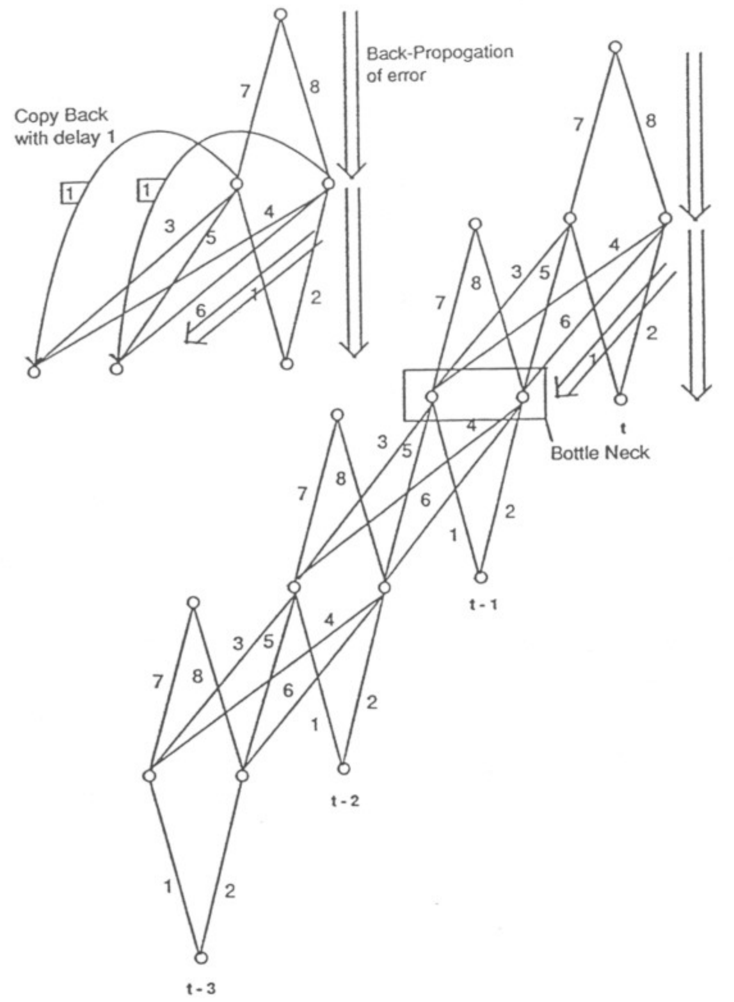

Figure 5. The copy-back back-propagation scheme for training recurrent networks. Top left is the copy-back instantiation of a simple recurrent network. The flow of back-propagation through is indicated by the double arrows. This flow of back-propagation is also shown for the network expanded into a feedforward network. Notice that back-propagation flows back only for the first "copy" of the recurrent network, rather than through the whole of the feedforward network. The labels 1–8 on the links indicate the correspondence between links in the two networks.

learn about. If this were true, the copy-back scheme would only be able to respond to the current and previous input, and would thus not be able to learn any interesting sequences. However, as long as the relevant temporally distant information "percolates through," even in some degraded form, to a point in the unfolded network to which error is propagated, the weights forward of that point can be adjusted to utilise that information successfully. Hence, the last point in the network to which error is propagated forms a "bottleneck," through which temporally more distant information must pass if the network is to be able to learn to respond to it (see Figure 5) (Chater, 1989). If the information about temporally distant inputs which is required for solving a particular problem is, as it happens, implicit in the hidden units values at the bottleneck (in such a way that the network after the bottleneck can extract it), then it will be possible to solve the problem. If not, the network will be unable to solve the problem, since it has no mechanism for adjusting the weights prior to the bottleneck so that the information is received successfully.

These considerations suggest that an important factor determining whether or not an SRN will be able to respond to temporally distant material is whether or not the relevant information is useful for predicting intervening material. If it is, then the network will tend to encode the information during the intervening time, and hence it is more likely to reach the bottleneck. In many sorts of sequential material, including natural language, the same kinds of information will be useful for both short- and long-term prediction, and hence a copy-back strategy stands a chance of responding to inputs from well into the past. However, in a task in which temporally distant information is not useful for predicting intervening material, learning with the copy-back scheme should be poor. Below, we report a paradigm example of such a task, the task of learning to be a delay-line.

While our primary interest in this first set of simulations was comparing the performance of copy-back and back-propagation through time, a secondary interest was in the effect of using or not using context units in back-propagation through time. If the network is unfolded several time-steps, the contribution of the context units at the bottleneck to the final output may be very small, and the large number of intervening layers may make it difficult to learn to respond to this input, even if it is informative. From a theoretical point of view, not using context units is attractive, since the network can then be viewed as learning a fixed input–output set (or a sample from a fixed distribution), and hence the proof that back-propagation performs gradient descent is valid. For most problems, the presence or absence of context units seems to have little effect on performance, and we discuss this briefly below.

We report simulations on three very simple tasks using binary sequences, discrete XOR, continuous XOR and learning to be a delay-line.

11.3.2 Simulations

Discrete XOR. Consider a binary sequence in which two out of three bits are generated at random, and the third is the XOR of the previous two. The task is to attempt to predict the next value in the sequence. This task is difficult, since only every third bit is in principle predictable. Optimal performance is the correct prediction of these bits, and an output of 0.5 otherwise.

Architecture	Hidden units	Average squared error		
		Position 1	Position 2	Position 3
copy-back	4	0.278 ± 0.003	0.273 ± 0.003	0.16 ± 0.02
copy-back	7	0.282 ± 0.002	0.288 ± 0.002	0.11 ± 0.01
copy-back	10	0.283 ± 0.001	0.289 ± 0.002	0.10 ± 0.01
unfolded with contexts	2	0.267 ± 0.004	0.271 ± 0.002	0.20 ± 0.02
unfolded with contexts	3	0.276 ± 0.004	0.280 ± 0.003	0.16 ± 0.03
unfolded with contexts	4	0.275 ± 0.004	0.280 ± 0.004	0.18 ± 0.03
unfolded with contexts	5	0.278 ± 0.004	0.283 ± 0.003	0.12 ± 0.02
5 unfolds no contexts	2	0.268 ± 0.004	0.270 ± 0.004	0.19 ± 0.02
5 unfolds no contexts	3	0.282 ± 0.003	0.285 ± 0.003	0.12 ± 0.01
5 unfolds no contexts	4	0.281 ± 0.003	0.286 ± 0.003	0.11 ± 0.02
5 unfolds no contexts	5	0.289 ± 0.001	0.287 ± 0.001	0.089 ± 0.004

Table 1. Performance on the discrete XOR task with 50 epochs of training.

Copy-back and back-propagation schemes (both with and without context) were trained on XOR (Table 1). The results were averaged over 50 trials, with 50 training epochs over 3000 input–output pairs with learning rate 0.1 and momentum 0.9. For back-propagation through time, the net was unfolded 5 time steps (these factors are constant through all the simulations we report here). The weights were initialised randomly between –5 and 5. If the weight starts are very much smaller than this, "copy-back" learning is slow. One explanation for this is that when the inputs to the processing units are small, the sigmoid activation function is nearly linear. Thus, the network closely approximates a layered network of linear units, which is known to be unable to compute XOR.

For a network of a given size, performance is far better with back-propagation through time than using the copy-back scheme, which requires far more hidden units to attain comparable results. This pattern is consistently obtained in a comprehensive range of simulations (Conkey, 1991) (notice that the standard deviations of the errors obtained are small in all these simulations).

Turning to our second concern, performance using back-propagation through time is not significantly different with or without context, despite the fact that context could in principle have provided very useful information, because the "no context" network may not be able to determine from just five time-steps which bits are predictable and which are not. If the last five bits were

...0 1 1 1 0

then the third and fifth bits are both the XOR of their predecessors. In this case it is not in principle possible for these unfolded nets without context units to know whether the next bit is the result of an XOR or is random. There is not enough information in the training input to uniquely determine the phase at which inputs are predictable. Since this ambiguity occurs in almost 60% of cases in the training set, the ability to use past context to disambiguate (effectively storing a regular "pulse" indicating which bits are predictable) would be advantageous. However, it appears that the network is not able to learn to utilise this information in practice.

Continuous XOR. A much easier sequential analogue of the XOR task is what we call continuous XOR. In this problem the input sequence is just a string of random bits and the output at each time-step is the XOR of the current and last inputs. So input and output might be:

Input: ...0 1 1 0 0 0 1 1 1 0 1...
Output: ...1 0 1 0 0 1 0 0 1 1...

The task is much easier than "discrete XOR," since a correct prediction can always be made, and hence the output can be determined purely on the basis of the last two inputs, rather than having to pay attention to the larger context to give information concerning the pattern of predictability and unpredictability in the sequence. The results obtained on this task are shown in Table 2. Both copy-back and unfolded networks learn this task fairly easily although for the same (small) number of hidden units the unfolded network performs better.

Architecture	Hidden units	Learning rate	Av. squared error
copy-back	2	0.1	0.1485
copy-back	3	0.1	0.0425
copy-back	4	0.1	0.0131
unfolded	2	0.1	0.0927
unfolded	3	0.1	0.0034

Table 2. Learning performance on continuous XOR.

Learning to be a delay-line. The analysis of the copy-back learning algorithm above suggested that it should be poor at learning to respond to temporally distant input, unless the temporally distant information has been used in intermediate predictions. This suggests that while in many interesting problems (such as that of learning a grammar with some recursive structure, detailed in Elman 1991) the net can respond to temporally distant information, this will be extremely difficult if the nature of the distant dependency is unrelated to the intervening material. The simplest such task is learning to be a delay-line — to reproduce a random binary input stream delayed by several time-steps.

A recurrent network with $n+1$ hidden units can act as a delay-line of n, given appropriate weights. One intuitively attractive solution is for the hidden units to act as buffers for the input so that one unit has output at time t of $i(t)$ another $i(t-1)$ and so on back to $i(t-n)$. Figure 6 illustrates weights that would implement this solution for a delay-line of three in a network with four hidden units.

Table 3 shows a typical sample of results. While back-propagation through time is able to learn the delay-line task quite well (and with only $n+1$ hidden units for small delays n), the copy-back scheme can only learn to respond to small delays with relatively large numbers of hidden units. This is explicable in terms of the theoretical discussion above — the more hidden units the more likelihood that relevant information will by chance percolate through the network and thus that the network will be able to learn to use this information. Learning performance is also very much less consistent with the copy-back regime.

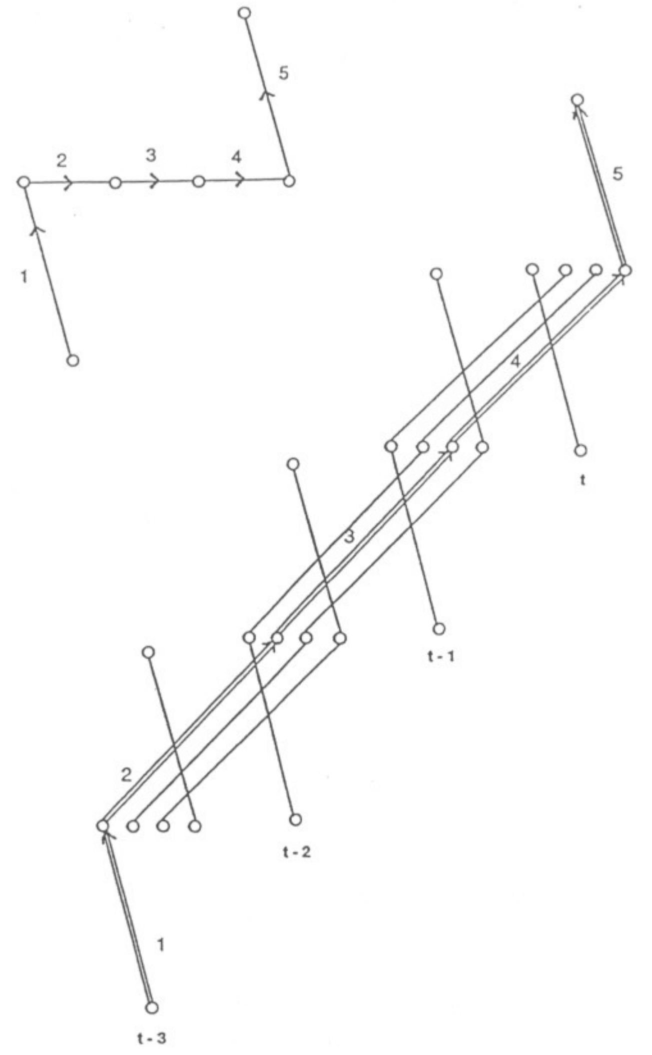

Figure 6. A handwired delay-line of three, using four hidden units, shown both as a recurrent network (top left) and expanded as a feedforward network (right). The flow of activation from the input at three time-steps back to the current output is indicated by double arrows. The labels 1–5 on the links indicate the correspondence between links in the two networks. This localist solution was not adopted by the networks trained by back-propagation through time.

Architecture	Delay	Hidden units	Learning rate	Av. squared error	Passes
copy-back	1	2	0.05	0.22	> 100
copy-back	1	2	0.10	0.21	> 100
copy-back	1	3	0.05	0.20	> 100
copy-back	1	3	0.10	0.16	> 100
copy-back	1	4	0.05	0.13	> 100
copy-back	1	4	0.10	0.097	> 100
unfolded	1	2	0.10	0.001	3
copy-back	2	3	0.05	0.254	> 100
copy-back	2	3	0.10	0.256	> 100
copy-back	2	4	0.05	0.262	> 100
copy-back	2	4	0.10	0.260	> 100
copy-back	2	7	0.05	0.278	> 100
copy-back	2	7	0.10	0.264	> 100
copy-back	2	10	0.05	0.160	> 100
copy-back	2	10	0.10	0.239	> 100
unfolded	2	3	0.10	0.031	3

Table 3. Learning to be a delay-line.

The inability of the copy-back scheme to learn to respond to long time delays contrasts with good performance reported predicting dependencies in small-scale language tasks where the intervening material is relevant (Elman, 1991).

The results of these experiments bear out the theoretical analysis that back-propagation through time leads to better learning than the copy-back scheme. However, back-propagation through time is computationally more expensive, and the copy-back scheme may be able to learn many interesting tasks. One particularly intriguing result is that the averaged hidden unit patterns appear to encode the syntactic/semantic categories for a toy grammar (Elman, 1990). The studies reported below repeat, extend and analyse this result, and argue that such a clustering is to be expected given the statistics of the sequences learnt.

Lexical category	Word
NOUN-HUMAN	man, woman, boy, girl
NOUN-ANIMATE	cat, mouse, dog
NOUN-INANIMATE	book, rock, car
NOUN-AGRESSOR	dragon, monster, lion
NOUN-FRAGILE	glass, plate
NOUN-FOOD	cookie, bread, sandwich
VERB-INTRANSITIVE	think, sleep, exist
VERB-TRANSITIVE	like, chase
VERB-AGENT/PATIENT	move
VERB-PERCEPTUAL	smell, see
VERB-DESTROY	break, smash
VERB-EAT	eat

Table 4. Word categories used by Elman (1988, 1990).

Word 1	Word 2	Word 3
NOUN-HUM	VERB-EAT	NOUN-FOOD
NOUN-HUM	VERB-PERCEPT	NOUN-INANIM
NOUN-HUM	VERB-DESTROY	NOUN-FRAG
NOUN-HUM	VERB-INTRAN	
NOUN-HUM	VERB-TRAN	NOUN-HUM
NOUN-HUM	VERB-AGPAT	NOUN-INANIM
NOUN-HUM	VERB-AGPAT	
NOUN-ANIM	VERB-EAT	NOUN-FOOD
NOUN-ANIM	VERB-TRAN	NOUN-ANIM
NOUN-ANIM	VERB-AGPAT	NOUN-INANIM
NOUN-ANIM	VERB-AGPAT	
NOUN-INANIM	VERB-AGPAT	
NOUN-AGRESS	VERB-DESTROY	NOUN-FRAG
NOUN-AGRESS	VERB-EAT	NOUN-HUM
NOUN-AGRESS	VERB-EAT	NOUN-ANIM
NOUN-AGRESS	VERB-EAT	NOUN-FOOD

Table 5. Sentence types used by Elman (1988, 1990).

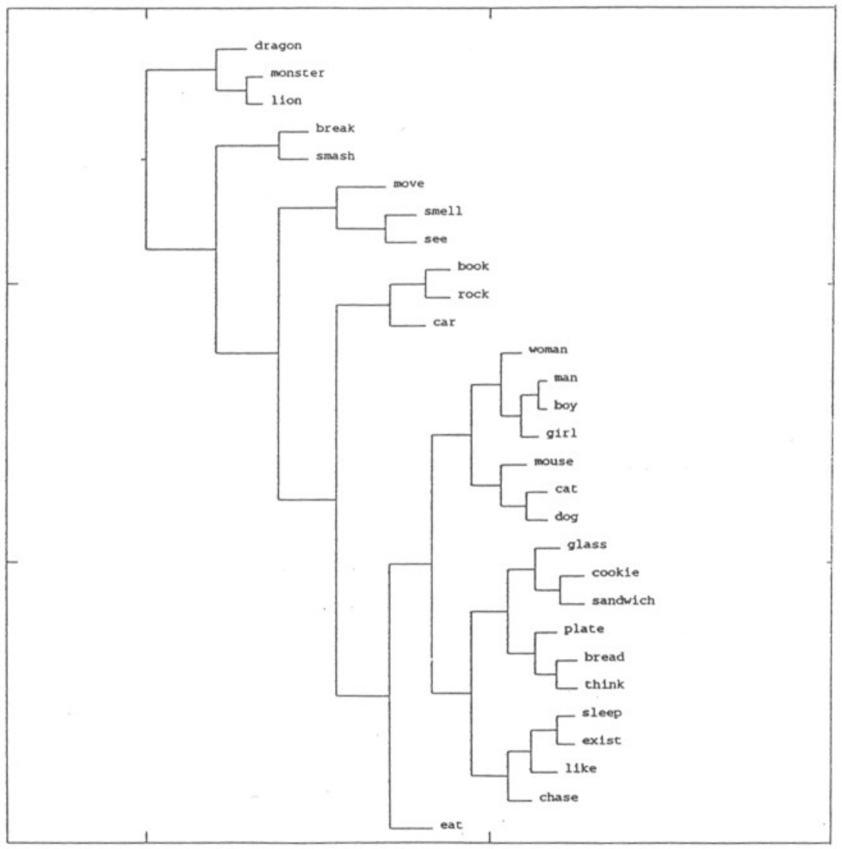

Figure 7. Clustering by current word.

11.4 Incidently recognising linguistic structure

Elman 1988; 1990) used the copy-back regime to train a net to predict the next item in a continuous text sequence, generated by a simple grammar shown in Tables 4 and 5.

Whereas Elman represented each "word" by a random bit vector, we used a completely localist representation, thus using 29 input units to represent the 29 words. As in Elman's simulations there is no explicit marker for the end of a

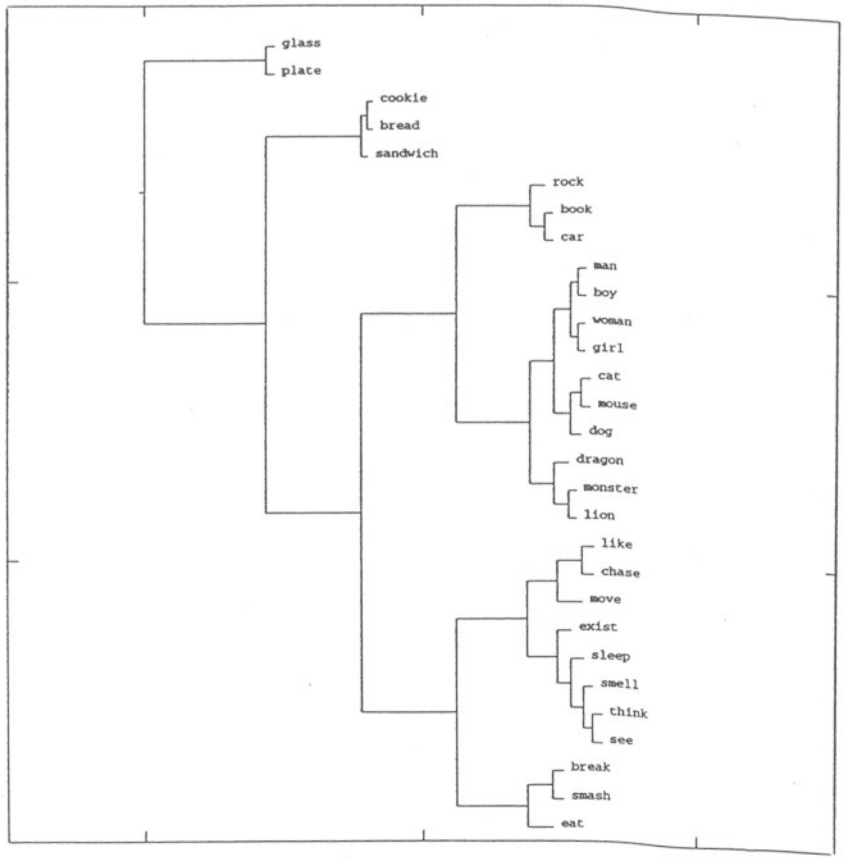

Figure 8. Clustering by predicted word.

sentence. One hundred and fifty hidden units and 150 corresponding context units were used.

Conditional probabilities for the next word, given the sentence so far, were calculated from the data set. The RMS errors relative to this benchmark were 0.2 per pattern in both cases, whereas Elman obtained 0.05. This difference may be a result of our choice of a localist input representation. We then followed Elman in cluster analysing the hidden unit activation evoked on presentation of

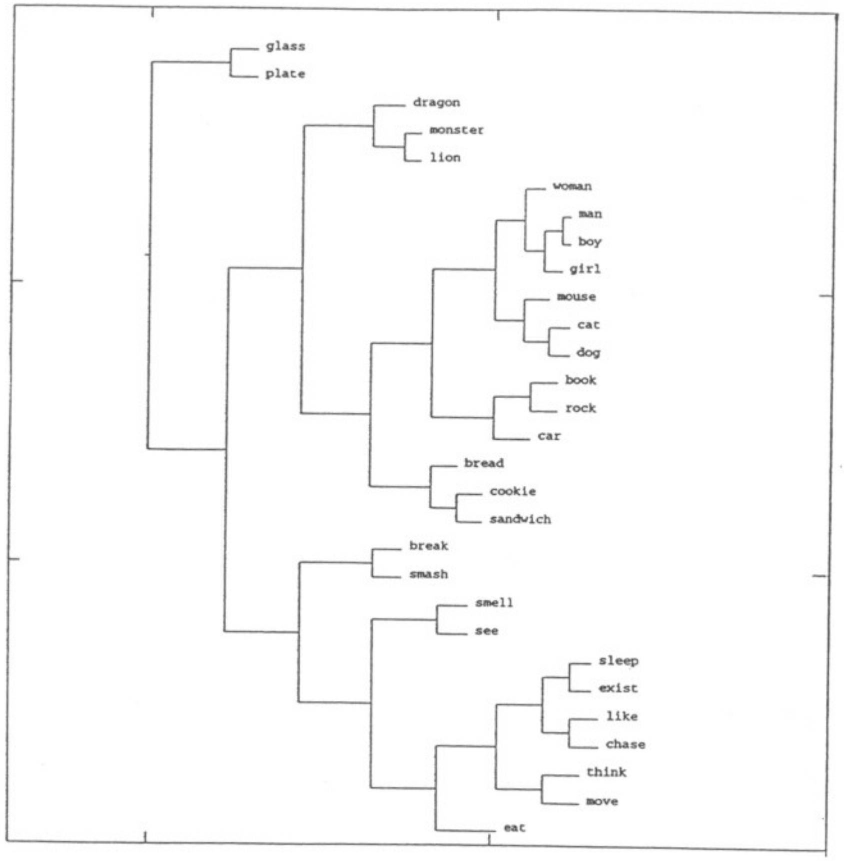

Figure 9. Clustering by change of hidden unit pattern.

each word. These were averaged to give a single 150 element vector for each of the 29 words.

The results from a typical net (Figure 7) do not give as good a clustering as that obtained by Elman. There is poor separation of nouns and verbs, and some confusion between different classes of each. Clustering on the basis of current input may not be the best measure, since the hidden unit values must encode previous input relevant to prediction, not just the current word. We also clustered hidden unit states averaged by the entire sentence so far. Hence a more attractive alternative is to average hidden unit patterns together on the basis of the word predicted.

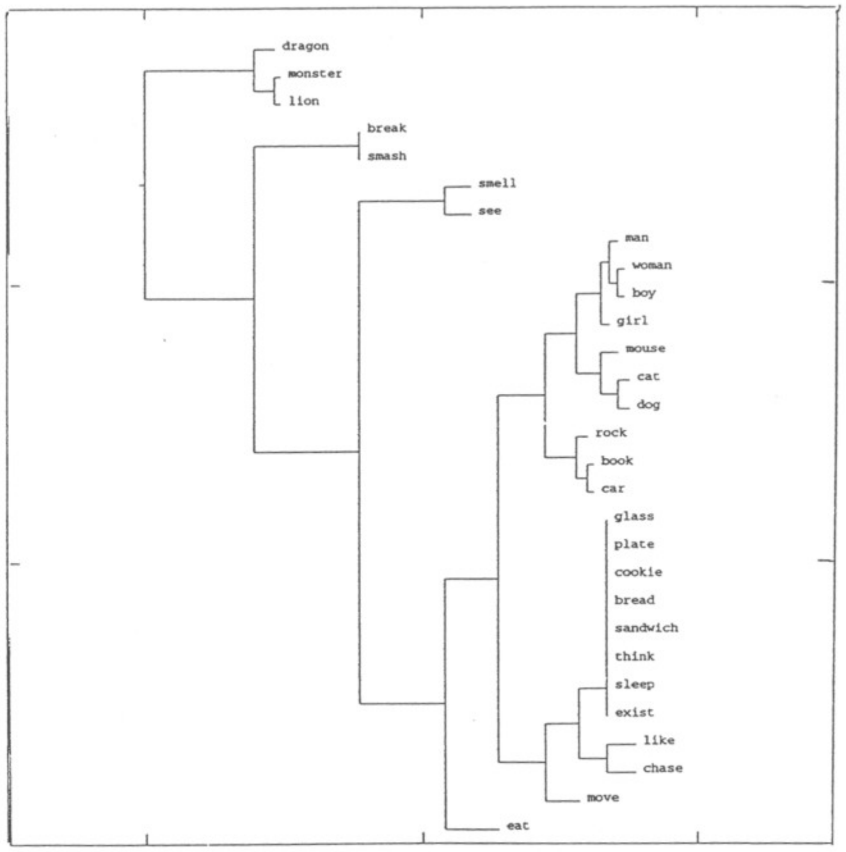

Figure 10. Conditional probabilities clustered by preceding word.

This measure, which completely cross-classifies the data with respect to the original measure, does indeed produce much better clusters, shown in Figure 8. Using this measure the clusters obtained well reflect the underlying syntactic categories of the grammar, with, for example, nouns being separated from verbs, and different kinds of nouns being very well segregated and verbs segregated somewhat less precisely.

A further possibility is to cluster not the hidden unit pattern associated with an incoming word, but the change in hidden unit representation brought about by that word. Again a good clustering is obtained (Figure 9), comparable in quality with that obtained by clustering with respect to the target word.

It seems that a variety of measures of hidden unit values produce clusters corresponding to linguistically interesting categories. There would appear not to be an "optimal" clustering of this data set to which all of these measures are approximating, since each of the measures considered above correspond to statistics of the data sets, which can be directly measured. For example, Elman's original measure of averaging hidden units on the basis of the past word corresponds to grouping words by the conditional probabilities of successive words. We measured this quantity directly, and then cluster analysed (Figure 10) to produce remarkably similar results to those obtained from the network (Figure 7) — this means that the network is successfully sampling the relevant statistic. Similar results can be obtained by comparing the two other measures with statistical analogues (clustering words on the basis of the conditional probabilities of the preceding words, and the change in conditional probabilities expected after a word is input, respectively). Since the copy-back scheme is sampling these statistics successfully, there seems to be no room for improvement using back-propagation through time, and thus we predict that the clusters from back-propagation through time will produce similar results. The limitation on performance is the structure of the data rather than nature of the network used.

These results suggest that the hidden unit patterns that recurrent neural networks develop can be viewed as reflecting quite directly the statistical structure of the sequences learnt. Furthermore, particular statistical measures of hidden unit activation may closely correspond to a related statistic of the sequence itself.

11.5 Conclusions and future directions

The first set of simulations reported confirmed the theoretically motivated expectation that the back-propagation through time is superior to (the less expensive) copy-back training for learning sequential structure. Experiments with large copy-back networks suggest that the hidden unit representation is successfully sampling statistics of the underlying sequential material, and we predict that back-propagation through time should, therefore, produce very similar clusters. Of course, if the underlying grammar, and hence the relevant

statistics, are more complex, then back-propagation through time may be able to sample these statistics better.

To what extent can it be expected that the performance of SRNs on simple artificial grammars can be scaled up to deal with more complex artificial grammars, and even real natural language data? This is, in practice, an important issue, since training SRNs becomes very difficult as the language to be learnt becomes more complex. One reason for this is that as the language becomes more complex the next word is less and less predictable, and hence the error score can be reduced less by learning. For example, suppose that we assume a localist coding for words, and that we assume that in a given context there are ten equally likely next words. The best prediction that the network can make is to assign 0.1 activation to the ten units corresponding to each of these words, and to assign 0 to all of the other units. This minimizes the expected sum squared error, but this is still rather high: namely $(0.1^2+ 0.1^2+ 0.1^2+ 0.1^2+ 0.1^2+ 0.1^2+ 0.1^2+ 0.1^2+ 0.1^2+ 0.9^2) = 0.9$. In particular, the error is very little different from the expected sum squared error of 1 if the network uniformly predicted that all units would be off.

This suggests three interesting avenues for further research:

(1) Investigation of the relationship between statistical analysis of the hidden unit representations and direct analysis of the original data set, both using the copy-back and back-propagation through time regimes.

(2) Exploration of real natural language data directly by cluster analysing using simple statistics to explore what peformance can be expected from a neural network model. Since extracting the relevant statistics directly is far less computationally expensive than training an SRN, this may be heuristically extremely useful. Furthermore, the statistical approach will be possible, even when the size of the lexicon and the complexity of the grammar is such that an SRN cannot be successfully trained. This might, for example, reveal in principle limitations on the power of an SRN type approach, and thus direct attention away from attempts to find tricks to make an SRN architecture learn in certain kinds of large- scale problem.

(3) Investigation to discover whether statistics which are revealing of linguistic structure can be implemented more directly in a network, so that a full-size network can be built which is able to handle real

natural language data. These last two avenues have recently been explored by Finch and Chater (1991, 1992, this volume) with encouraging results.

Chapter 12

Learning Syntactic Categories: A Statistical Approach

Steven Finch and Nick Chater

12.1 The bootstrapping problem

The acquisition of language is remarkably swift and successful despite the exquisite complexity of what is acquired and the incomplete and errorful character of the data upon which acquisition is based. The problem is particularly difficult, since both the categories over which linguistic rules are defined, and the rules themselves must be found (if both of these must be learnt, rather than being prespecified). That is, the learner faces a "bootstrapping" problem (Finch & Chater, 1991, 1992): linguistic rules presuppose the linguistic categories in terms of which they are stated; and the validity of linguistic categories depends on whether or not they support perspicuous linguistic rules. Given this interdependence of rules and categories, it is not clear how acquisition can occur, except by searching the vast number of possible of categories/rules combinations at once.

The bootstrapping problem arises in the acquisition of all aspects of linguistic structure, whether phonological, syntactic or semantic. Indeed, similar problems arise in learning the structure of almost any new domain. For example, in learning an academic subject, say elementary physics, learners must somehow acquire both the relevant concepts and the correct rules of inference defined over those concepts. For example, learners must grasp the concepts of momentum, force and so on, as well as the rules for how these concepts can be manipulated and interrelated. The bootstrapping problem is acute since these two projects are thoroughly interdependent — understanding the concepts presupposes some understanding of the rules in which they figure,