5 A Focused Backpropagation Algorithm for Temporal Pattern Recognition

Michael C. Mozer
Department of Computer Science, University of Colorado

ABSTRACT

Time is at the heart of many pattern recognition tasks (e.g., speech recognition). However, connectionist learning algorithms to date are not well-suited for dealing with time-varying input patterns. This chapter introduces a specialized connectionist architecture and corresponding specialization of the back-propagation learning algorithm that operates efficiently, both in computational time and space requirements, on temporal sequences. The key feature of the architecture is a layer of self-connected hidden units that integrate their current value with the new input at each time step to construct a static representation of the temporal input sequence. This architecture avoids two deficiencies found in the back-propagation unfolding-in-time procedure (Rumelhart, Hinton, & Williams, 1986) for handing sequence recognition tasks: first, it reduces the difficulty of temporal credit assignment by focusing the back-propagated error signal; second, it eliminates the need for a buffer to hold the input sequence and/or intermediate activity levels. The latter property is due to the fact that during the forward (activation) phase, incremental activity *traces* can be locally computed that hold all information necessary for back propagation in time. It is argued that this architecture should scale better than conventional recurrent architectures with respect to sequence length. The architecture has been used to implement a temporal version of Rumelhart and McClelland's (1986) verb past-tense model. The hidden units learn to behave something like Rumelhart and McClelland's "Wickelphones," a rich and flexible representation of temporal information.

INTRODUCTION

Connectionist models have proven successful in a variety of pattern recognition tasks (e.g., Hinton, 1987; Sejnowski & Rosenberg, 1987). In some respects,

these models are amazingly powerful, more so than the human brain. For instance, take a real-world image composed of light intensities and randomly rearrange pixels in the image. Most connectionist architectures can learn to recognize the permuted image as readily as the original (Smolensky, 1983), whereas humans would no doubt have great difficulty with this task. In other respects, however, the pattern recognition abilities of connectionist models are quite primitive. While humans have little trouble processing temporal patterns—indeed, all input to the sensory systems is intrinsically temporal in nature—few connectionist models deal with time. Because time is at the essence of many pattern recognition tasks, it is important to develop better methods of incorporating time into connectionist networks.

Figure 1 depicts an abstract characterization of the temporal pattern recognition task. Time is quantized into discrete steps. A sequence of inputs is presented to the recognition system, one per time step. Each element of the sequence is represented as a vector of feature values. At each point in time, the system may be required to produce a response, also represented as a vector of feature values, contingent on the input sequence of that point. In the simplest case, shown in Figure 1, a response is required only after the entire input sequence has been presented.

Many important problems are of this class. For instance, recognizing speech involves sampling the acoustic signal at regular intervals and producing as output a representation of phonemes or words. Similarly, natural language processing consists of analyzing a sequence of words to yield a structural or semantic description. And event perception can be viewed as analyzing a sequence of snapshots of the visual world to produce a description of the ensuing event. Freyd (1987) has argued that even for some static objects, perception may be dynamic in the sense that a temporal dimension is incorporated into the perceptual analysis.

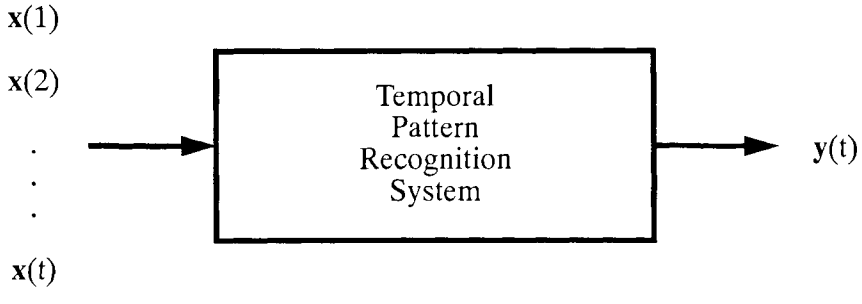

Figure 1. Abstract characterization of the temporal pattern recognition task. $\mathbf{x}(t)$ indicates the input pattern at time t, $\mathbf{y}(t)$ the output pattern.

PREVIOUS CONNECTIONIST APPROACHES TO TEMPORAL PATTERN RECOGNITION

One popular approach to temporal pattern recognition has been to construct a buffer to hold the n most recent elements of the input sequence (Elman & McClelland, 1986; Elman & Zipser, 1988; Landauer, Kamm, & Singhal, 1987; Lapedes & Farber, 1987; McClelland & Elman, 1986; Plaut, Nowlan, & Hinton, 1986; Tank & Hopfield, 1987; Waibel, Hanazawa, Hinton, Shikano, & Lang, 1987). Such a buffer can be implemented using a shift register or delay lines. The buffer turns a temporal recognition problem into a spatial recognition problem in which all relevant information for making a response is simultaneously available. Because connectionist models are relatively good at spatial recognition problems, this approach seems assured of some success.

However, the approach has four serious drawbacks. First, the buffer must be sufficient in size to accommodate the longest possible input sequence. With an n-element buffer, no sequence of duration greater than n can be recognized. Thus, the longest possible sequence—the longest interval over which context may play a role—must be known in advance. Even if a fairly large buffer can be built, say one sufficient to recognize a phoneme or demisyllable, what about higher levels of analysis—words, phrases, sentences, paragraphs? At some point, one needs to deal with the fact that not all input information can be available to the system simultaneously. A hierarchy of buffers may partially alleviate this problem, but will still require the advance specification of maximum sequence duration.

A second drawback of using a buffer is that by making a great deal of information simultaneously available, much computation is required at each time step. Essentially, all information within the buffer must be reprocessed whenever the buffer state changes. This is not a problem if one has dedicated parallel hardware, but simulations of such a connectionist system on a serial machine can be computationally expensive.

Third, when a buffer is used as input to a connectionist network, each element of the buffer must be connected to higher layers of the network. Consequently, as the buffer grows, so does the number of weights. This means that a large number of training examples must be used or else the network will not generalize well (Hinton, 1989). Another solution to this problem is to constrain the weights in some manner so as to reduce the number of free parameters (Lang, 1987; Le Cun, this volume; Waibel et al. 1987).

Fourth, the use of a buffer makes it difficult to achieve invariance under translation in time. Because the buffer turns shifts in time into shifts in space (i.e., buffer position), the representation of an input sequence occurring at one time will have little resemblance to that of an input sequence occurring at another time. Consequently, if training and test sequences are misaligned, the sequences

will likely not be recognized as the same. One way of minimizing this problem is to shift inputs continuously across the buffer to ensure that each sequence is presented in each position, both during training and testing. However, this solution introduces noise into the training phase and additional computation into the testing phase.

These deficiencies of the buffer model argue that the spatial metaphor for time is not viable; a richer, more flexible representation of time is needed. Similar arguments have been raised elsewhere (Elman, 1990; Stornetta, Hogg, & Huberman, 1987; Watrous & Shastri, 1987). Despite its drawbacks, the buffer model has two properties in common with any model of temporal pattern recognition. First, some memory of the input history is required. Second, a function must be specified to combine the current memory (or *temporal context*) and the current input to form a new temporal context:

$$\mathbf{c}(t+1) = f(\mathbf{c}(t), \mathbf{x}(t)),$$

where $\mathbf{c}(t)$ is a vector representing the context at time t, $\mathbf{x}(t)$ is the input at time t, and f is the mapping function. The buffer model is a simple scheme, where the temporal context consists of the n most recent sequence elements, and f is implemented by the shift-register operation of the buffer. Given the inadequacy of the buffer model, one would like to discover ways of representing temporal context that avoid turning intrinsically temporal information into spatial information.

One idea is based on the fact that, in a connectionist network, the connections from one set of units to another implement a mapping. Thus, by representing the input $\mathbf{x}(t)$ and context $\mathbf{c}(t)$ as patterns of activity on two sets of units, the weights connecting the input units to the context units and the context units to themselves specify a mapping function f. Jordan (1987) and Stornetta et al. (1987) have explored this approach using fixed weights that do not change with experience. In the Stornetta et al. work, there is one context unit per input unit, and each context unit is connected to itself and its corresponding input unit. In other words, the mapping function is

$$f(\mathbf{c},\mathbf{x}) = k_1\mathbf{c} + k_2\mathbf{x},$$

where k_1 and k_2 are fixed constants.

This type of network has no spatial representation of time. Consequently, the architecture does not require replicated input units, in contrast to the buffer model which requires n copies of the input units for an n-element buffer. Further, this architecture does not place a rigid upper bound on the amount of temporal context that can be considered, whereas the buffer model can remember only the n most recent sequence elements. Nonetheless, this approach is rather inflexible in that the mapping function used to construct the temporal context is predetermined and fixed. As a result, the representation of temporal context must be sufficiently rich that it can accommodate a wide variety of tasks; it cannot afford

to discard too much information. The alternative to this general, task-independent representation is one suited to the task being performed, wherein only the input information relevant to the task need be retained.

Connectionist learning algorithms provide a means of adaptively constructing internal representations. However, the most promising and popular algorithm, back propagation (Rumelhart et al. 1986), is designed for feedforward networks. In order to represent temporal context, recurrent networks are required because the current context must depend on the previous. Back propagation can be used to train recurrent networks if the network is "unfolded in time" (Rumelhart et al., 1986; see also chapters in this volume by Bachrach and Mozer, Nguyen and Widrow, Servan-Schreiber et al., and Williams and Zipser). The basic trick can be seen by comparing the recurrent network in Figure 2a to the equivalent unfolded network in Figure 2b. The network in Figure 2a consists of four layers: input, context, hidden, and output. The input pattern is integrated with the current context to form a new context. The context is then mapped, by way of the hidden layer, to an output. In Figure 2b, the same functionality is achieved by replicating the input and context layers for each element of the input sequence and by constraining the weights such that the input-to-context connections and context-to-context connections are equal across time. Rather than having four pools of units, the unfolded network contains $2 + 2t$ pools, where t is the number of elements in the input sequence. Because the unfolded network is feedforward, the back-propagation algorithm can be applied to adjust the connection strengths so that a given input sequence will yield a target output.[1] The weight constraints are enforced by maintaining only one set of input-to-context and context-to-context weights. During the back-propagation phase, the changes prescribed for each weight at each level are summed to give a net weight change.

Training a network using the unfolding procedure has two significant drawbacks, however. First, part of the architecture must be replicated for each time step of the input sequence. In implementing the unfolding procedure, it is not actually necessary to replicate processing units; units need be instantiated only once if each unit remembers its activity level at each point in time—that is, maintains a stack of activity levels. During forward propagation, values are pushed on to this stack, and during back propagation values are popped in reverse temporal order.[2]

Second, the unfolding procedure creates a deeply layered network through which error signals must be propagated. This is bad not only because the time to perform back propagation is proportional to the depth of the network, but also

[1]Intermediate output values could readily be trained by replicating the hidden and output units at intermediate time steps, and injecting an additional error signal into the network via the output units.

[2]Almeida (1987) and Pineda (1987) have proposed a variation of the back-propagation algorithm for recurrent networks that does not need an activity-level history. However, the algorithm assumes a fixed input, and hence is suitable for pattern completion and other relaxation problems, not for analyzing a time-varying input pattern.

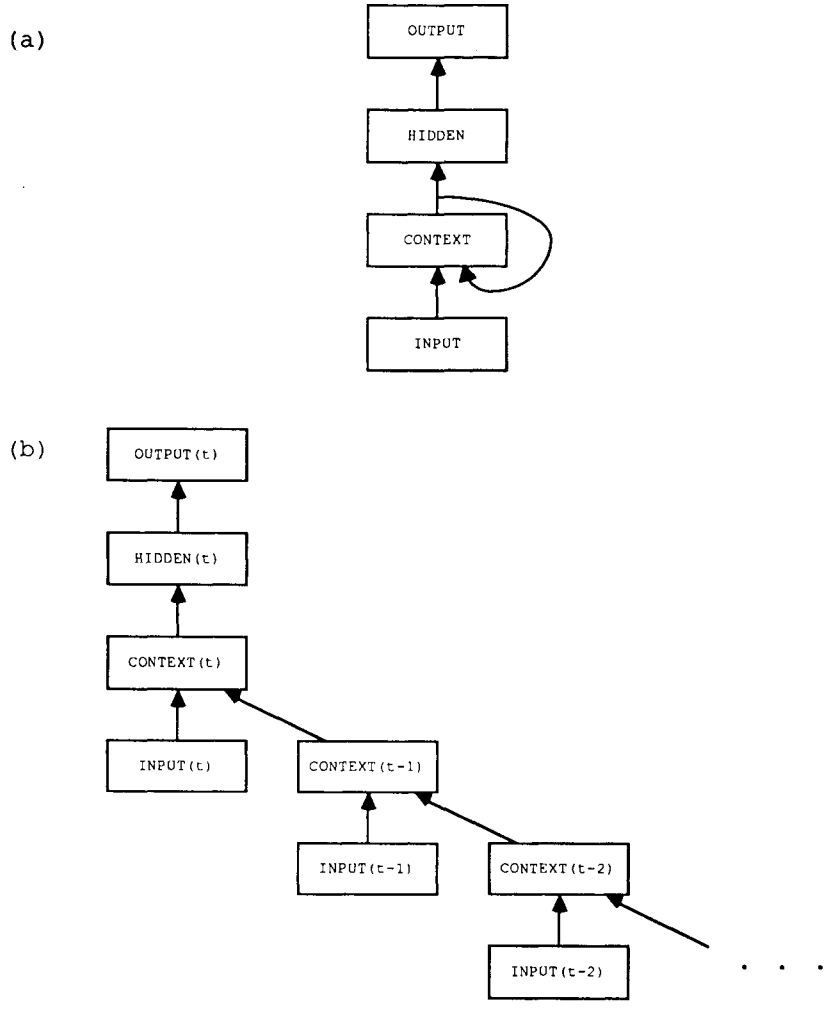

Figure 2. (a) A four-layered recurrent network consisting of input, context, hidden, and output units. Each labeled box indicates a set of processing units. The arrows indicate complete connectivity from one layer to another; that is, each unit in one layer is connected to each unit in the other. (b) The same network unfolded in time. The input and context layers are replicated for each element of the sequence. The weights are constrained so that the input-context connections are equal across time, as are the context-context connections.

because the further back an error signal is propagated, the more dispersed it becomes. To explain what I mean by "dispersed," consider that the purpose of back propagation is to assign blame to each unit for its contribution to the error. However, the assignment of blame to a given unit is meaningful only in the context of the response properties of units higher in the network. If the present responses of these units do not resemble their eventual responses, then the lower-layer unit cannot obtain an independently informative measure of its contribution to the error; back propagation through the upper layers will effectively redistribute the error signal in some random fashion. Thus, in deep networks, especially where the relevant input signal is found in the lower layers, learning can be very slow. This argument is born out by empirical comparisons of learning speed using temporal versus spatial patterns made by myself and by Steven Nowlan (personal communication).

CONSTRAINTS ON NETWORK ARCHITECTURE

Given the inherent difficulties in using back propagation to train a recurrent network to recognize temporal sequences, one useful tack is to look for constraints on solutions the network might discover that could simplify the learning problem. This strategy of building a priori structural knowledge into the network directly can often speed learning and improve generalization (Hinton, 1989).

Consider the sort of representation one might like to obtain in the context layer of the network in Figure 2a. This representation should satisfy four criteria. First, it must be a static encoding of the temporal input pattern, one that holds on to whichever features of the input are needed to produce the desired response. Second, it must be capable of encoding sequences of varying lengths with a fixed number of units. Third, it must be capable of encoding relationships between events. And fourth, it should provide a natural basis for generalization.

Wickelgren (1969) has suggested a representational scheme that seems to satisfy these criteria and has been applied successfully in several connectionist models (Mozer, 1991; Rumelhart & McClelland, 1986; Seidenberg, 1990). The basic idea is to encode each element of a sequence with respect to its local context. For example, consider the phonetic encoding of a word. Wickelgren proposed context-sensitive phoneme units, each responding to a particular phoneme in the context of a particular predecessor and successor. I will call these units *Wickelphones*, after the terminology of Rumelhart and McClelland. If the word *explain* had the phonetic spelling /eksplAn/, it would be composed of the Wickelphones $_{-}\mathbf{e}_{\mathbf{k}}$, $_{\mathbf{e}}\mathbf{k}_{\mathbf{s}}$, $_{\mathbf{k}}\mathbf{s}_{\mathbf{p}}$, $_{\mathbf{s}}\mathbf{P}_{\mathbf{l}}$, $_{\mathbf{p}}\mathbf{l}_{\mathbf{A}}$, $_{\mathbf{l}}\mathbf{A}_{\mathbf{n}}$, and $_{\mathbf{A}}\mathbf{n}_{-}$ (where the dash indicates a word boundary). Assuming one Wickelphone unit for each possible phoneme triple, activation of a word would correspond to a distributed pattern of activity over the Wickelphone units.

With a fixed number of Wickelphone units, it is possible to represent uniquely

arbitrary strings of varying length. This means that the unordered set of Wickelphones is sufficient to allow for the unambiguous reconstruction of the ordered string. There are difficulties if the string contains repeated substrings (e.g., *Mississippi*), but these difficulties can be overcome (Mozer, 1990). The Wickelphone representation can be generalized to arbitrary sequences by substituting sequence elements for phonemes. In the general case, I call the context-sensitive encoding a *Wickelement* representation.

The trouble with Wickelements is that there are too many of them. Rumelhart and McClelland reduced the number of Wickelphones by devising a more compact and distributed encoding that depended on features of phonemes rather than the phonemes themselves. The number of units can also be reduced on the grounds that not all Wickelements are needed for every task. For instance, a $_{\mathrm{p}}\mathbf{k}_{\mathrm{t}}$ Wickelphone is unnecessary for representing English words. Thus, it would be desirable to learn only the task-relevant Wickelements.

How might the network in Figure 2a be modified to learn an internal representation in the context layer that resembled Wickelements? First, to obtain local context-sensitive codes, the sequence might be presented in local "chunks." This can be achieved by turning the input layer into a small buffer, so that at time t the input pattern consists of the sequence elements at, say, times $t - 2$, $t - 1$, and t. Then the context units can detect conjunctions of sequence elements, or conjunctions of features of sequence elements. Once activated by a pattern in the input, the context units should remain on. Thus, it seems sensible to have self-connected context units, but not to connect each context unit to each other, say an activation function like

$$c_i(t + 1) = d_i c_i(t) + s[net_i(t)], \tag{1}$$

where $c_i(t)$ is the activity level of context unit i at time t, d_i is a decay weight associated with the unit, s is a sigmoid squashing function, and $net_i(t)$ is the net input to the unit:

$$net_i(t) \equiv \sum_j w_{ji} x_j(t),$$

$x_j(t)$ being the activity of input unit j at time t, w_{ji} the connection strength from input unit j to context unit i. Thus, a context unit adds its current activity, weighted by the decay factor, to the new input at each time. The decay factor allows old information to fade over time if $d_i < 1$. Such decay connections have proven useful in other work (Jordan, 1987; Miyata, 1988; Stornetta et al., 1987; Watrous & Shastri, 1987).

To summarize, a recurrent network with this architecture, which I call the *focused* architecture for reasons that will become clear shortly, differs from a *full* recurrent architecture in three respects: (1) the input layer consists of a small temporal buffer holding several elements of the input sequence; (2) connectivity

in the context layer is restricted to one-to-one recurrent connections; and (3) integration over time in the context layer is linear.[3]

THE FOCUSED BACK-PROPAGATION ALGORITHM

It turns out that the focused architecture has properties that overcome the two major limitations discussed earlier of a full recurrent architecture and the unfolding-in-time training procedure. First, back propagation is "focused": the error signal does not disperse as it propagates back in time. Second, to adjust the weights, back propagation in time—and saving an activity history stack—is unnecessary.

To get an intuition as to why these two statements are true, consider the weight update procedure when a t-element sequence is presented to a focused recurrent network. Following the sequence, at time t, the network is shown a target output vector. Comparing this vector to the actual output vector yields an error signal, E. To adjust weights according to the back propagation gradient descent procedure, it is necessary to compute

$$\delta_i(\tau) \equiv \frac{\partial E}{\partial c_i(\tau)}$$

for $\tau = 1, \ldots, t$. This can be achieved by back propagating in time, from the context layer at time t to $t - 1$ to $t - 2$ and so forth. However, from Equation 1 it is clear that

$$\frac{\partial c_i(\tau)}{\partial c_i(\tau - 1)} = d_i.$$

Consequently,

$$\delta_i(\tau - 1) \equiv \frac{\partial E}{\partial c_i(\tau - 1)} = \frac{\partial c_i(\tau)}{\partial c_i(\tau - 1)} \frac{\partial E}{\partial c_i(\tau)} = d_i \delta_i(\tau). \tag{2}$$

The error signal, $\delta_i(\tau)$, changes just by a constant multiplicative factor, d_i, as it is propagated back in time. Thus, there is a simple relationship between the δ_i's at various points in time.

[3]The simple recurrent network (SRN) architecture described by Elman (1990) and Servan-Schreiber et al. (this volume) has some superficial similarity with the focused architecture. In particular, there is a set of feedback connections in the SRN which are both one-to-one and linear. However, these connections do not correspond to the recurrent connections in the focused architecture. The SRN is simply a generic three-layer architecture with complete recurrent connectivity in the hidden layer. This is not always clear because the SRN is usually drawn with two copies of the hidden layer, one representing the activities at the current time step and the other (often called the context) representing the activities at the previous time step. The one-to-one linear connections serve only to preserve the previous hidden activity for one time step.

Because of Equation 2, whatever error is propagated back to the context unit at time t stays within that unit as the error is passed further back in time, in contrast to a full recurrent network where the error is redistributed among the context units with each backwards pass due to cross connections between units. Error propagation with this focused architecture is therefore focused and should not disperse in time—an apparent limitation of the full recurrent architecture.

Error propagation with the focused architecture is also superior in a second respect. Because of the simple relationship described by Equation 2, it is not necessary to explicitly back-propagate in time to compute $\delta_i(\tau)$ from $\delta_i(t)$. Instead, if each connection has associated with it an *activity history trace* that is incrementally updated during the forward (activation) pass, these traces can be used to exactly achieve the effects of back propagation in time.

The appendix derives formulas for $\partial E/\partial d_i$ and $\partial E/\partial c_i(t)$ in terms of the activity traces, which yield the following weight update rules. For the recurrent connections, d_i, the rule is

$$\Delta d_i = -\epsilon \delta_i(t) \alpha_i(t),$$

where $\alpha_i(0) = 0$ and

$$\alpha_i(\tau) = c_i(\tau - 1) + d_i \alpha_i(\tau - 1).$$

Similarly, the weight update rule for the input-context connections, w_{ji}, is

$$\Delta w_{ji} = -\epsilon \delta_i(t) \beta_{ji}(t),$$

where $\beta_{ji}(0) = 0$ and

$$\beta_{ji}(\tau) = s'[net_i(\tau)] x_j(\tau) + d_i \beta_{ji}(\tau - 1).$$

These weight update rules are not just heuristics or approximations; they are *computationally equivalent* to performing the back propagation in time.

Choosing a Squashing Function for the Context Units

An interesting issue arises in the choice of a squashing function, $s[u]$, for the context units. If we indeed hope that the context units learn to behave as Wickelement detectors, we would like the squashing function to have range 0–1, in order that the response is 0.0 if the input pattern does not match the Wickelement or 1.0 if it does. But to the extent that the unit is not a perfect Wickelement detector (as will be the case initially), it will produce small positive responses on each time step. Consequently, the activity level of the context unit will grow in proportion to the number of sequence elements, an undesirable property that could result in unstable learning behavior and poor generalization. One remedy is to use a zero-mean squashing function, say with the range -0.5 to $+0.5$. Because positive and negative values will cancel when summed over time, the

activity of a context unit should be independent of sequence length (at least initially, when the weights are uncorrelated with the inputs). However, the context unit will be unable to respond to Wickelements in the manner described: it is impossible to set the weights so that the zero-mean squashing function yields a positive value if the input pattern matches the Wickelement or 0.0 otherwise. To summarize, a squashing function with the range -0.5 to $+0.5$ seems appropriate initially, when weights are untrained and the output of a unit is more-or-less random, but the range 0.0 to 1.0 seems necessary to perform binary discriminations in which one of the desired output levels is 0.0.

Although one solution might be to manually adjust the range of this function as the network learns, I have opted for a different approach: to allow the network to *learn* the zero point of the function. For each context unit, I have introduced an additional parameter, z_i, the zero point, and defined the squashing function for unit i to be

$$s_i[u] \equiv \frac{1}{1 + e^{-u}} + z_i.$$

If z_i is 0.0, the range of the function is 0.0 to 1.0; if z_i is $-.5$, the range is -0.5 to $+0.5$.

As with the other parameters in the network, z_i can be adjusted by gradient descent. The update rule, derived as those for d_i and w_{ji}, is

$$\Delta z_i = -\epsilon \delta_i(t) \gamma_i(t),$$

where $\gamma_i(0) = 0$ and

$$\gamma_i(\tau) = 1.0 + d_i \gamma_i(\tau - 1).$$

Related Work

Several researchers have independently discovered the idea of computing an activity trace during the forward pass as an alternative to back propagation in time. Williams and Zipser (1989, this volume) report on a generalization to arbitrary recurrent network architectures; this generalization is of questionable practical use, however, because the number of traces grows with the cube of the number of units. Bachrach (1988), Gori, Bengio, and De Mori (1989), and Yoshiro Miyata (personal communication) have studied a version of the current architecture with a more complex context-unit activation function in which the recurrent input is contained inside the squashing function:

$$c_i(t + 1) = s[net_i(t)], \tag{3}$$

where

$$net_i(t) \equiv d_i c_i(t) + \sum_j w_{ji} x_j(t).$$

In this case, the weight update rules are as before with

$$\alpha_i(\tau) = (c_i(\tau - 1) + d_i\alpha_i(\tau - 1))\ s'[net_i(t)]$$

and

$$\beta_{ji}(\tau) = (x_j(\tau) + d_i\beta_{ji}(\tau - 1))\ s'[net_i(t)].$$

In practice, Miyata and I have found Equation 1 to work better than Equation 3 because squashing the recurrent input tends to cause the context units to forget their values over time. Bachrach (1988) has analyzed the nature of this forgetting more formally.

SIMULATION RESULTS

Implementation Details

The simulations reported in the following sections used an architecture like that shown in Figure 2, except that the hidden layer was not needed; the context layer mapped directly to the output layer.

The initial input-context and context-output connection strengths were randomly picked from a zero-mean Gaussian distribution and were normalized so that the L1 norm of the fan-in (incoming) weight vector was 2.0. The z_i were initially set to -0.5, and the initial d_i were picked at random from a uniform distribution over the interval 0.99–1.01.

A "batch" updating procedure was used during training; that is, the weights were updated only after a complete presentation of the training set (an *epoch*). Momentum was not used. Learning rates were determined individually for each set of connections: input-context, decay, zero points, and context-output. The learning rates were set dynamically after each epoch according to the following heuristic:

$$\epsilon_k = mse^{\mu}\rho \min\left(\omega, \frac{W_k}{\nabla_k}\right),$$

where ϵ_k is the learning rate for connections of type k, *mse* is the mean-square error across output units and patterns for the previous epoch, W_k is the mean L1 norm of the fan-in weight vector for connections of type k, ∇_k is the *mean* L1 norm of the fan-in gradient vector for input-context and context-output connections and the *maximum* magnitude for the decay and zero-point connections, and μ, ρ, and ω are constants. The *mse* term serves to decrease the learning rate as the error becomes smaller; μ is a discounting factor for this term (set to values in the neighborhood of 0.75–1.0). The second term defines a "nominal" learning rate which is set so that, on average, weight updates change each unit's fan-in weight vector by a fixed proportion (ρ, generally 0.02) of its current magnitude. The parameter ω specifies an upper limit on the step size when ∇_k becomes

extremely small. This rule produces learning rates for the decay and zero-point terms that are about one-tenth of the other learning rates; this relatively small step size seems necessary to ensure stability of the network.

Although I had hoped to devise a rule for automatically adjusting the learning rates that was architecture and problem independent, the foregoing rule does not satisfy this requirement. The parameters μ, ρ, and ω had to be fine tuned for most applications to give optimal performance. However, the rule did work much better than fixed learning rates and other variants that I experimented with.

Learning Wickelements

Starting with a simple example, the network was trained to identify four sequences: **_DEAR_**, **_DEAN_**, **_BEAR_**, and **_BEAN_**. Each symbol corresponds to a single sequence element and was represented by a binary activity pattern over three units (Table 1). The input layer was a two-element buffer through which the sequence was passed. For **_DEAR_**, the input on successive time steps consisted of **_D**, **DE**, **EA**, **AR**, **R_**. The input layer had six units, the context layer two, and the output layer four. The network's task was to associate each sequence with a corresponding output unit. To perform this task, the network must learn to discriminate **D** from **B** in the first letter position and **N** from **R** in the fourth letter position. This can be achieved if the context units learn to behave as Wickelement detectors. For example, a context unit that responds to the Wickelements **_D** or **DE** serves as a **B-D** discriminator; a unit that responds to **R_** or **AR** serves as an **N-R** discriminator. Thus, a solution can be obtained with two context units.

Fifty replications of the simulation were run with different initial weight configurations. The task was learned in a median of 488 training epochs, the criterion for a correct response being that the output unit with the largest value was the appropriate one. Figure 3 shows the result of one run. The weights appear in the upper half of the figure, and activity levels for each input sequence in the lower half. The weights are grouped by connection type, with the input-

TABLE 1
Symbol Encoding

Symbol	*Activity Pattern*		
A	0	0	0
B	0	0	1
E	0	1	0
D	0	1	1
N	1	0	0
R	1	0	1
—	1	1	0

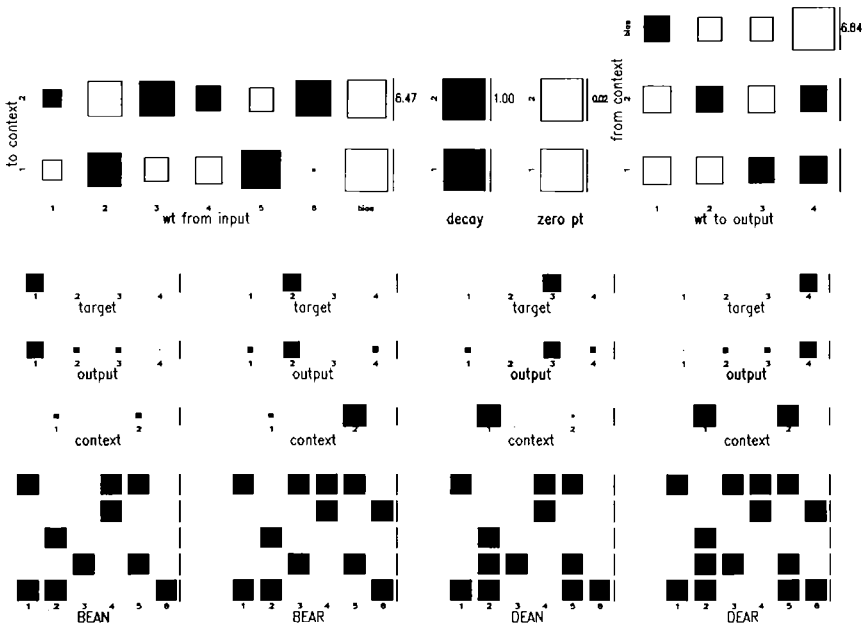

Figure 3. The **DEAR/DEAN/BEAR/BEAN** problem. The upper half of the figure shows learned weights in the network; the lower half activity levels in response to each of the four input sequences.

context connections in the upper-left array, followed by the decay connections (d_i), zero points (z_i), and context-output connections. Each connection is depicted as a square whose *area* indicates the relative weight magnitude, and shading the weight sign—black is positive, white is negative. The sizes of the squares are normalized within each array such that the largest square has sides whose length is equal to that of the vertical bars on the right edge of the array. The absolute magnitude of the largest weight is indicated by the number in the upper-right corner. Among the input-context connections, the largest weight magnitude is 6.47, among the decay values 1.00, the zero points 0.02, and the context-output connections 6.84. Because normalization is performed within each array, weight magnitudes of different connection types must be compared with references to the normalization factors.

The units within each layer are numbered. The weights feeding into and out of context unit 1 have been arranged along a single row, and the weights of context unit 2 in the row above. Bias terms (i.e., weight lines with a fixed input of 1.0) are also shown for the context and output units.

For the activity levels in the lower half of the figure, there are four columns of values, one for each sequence. The input pattern itself is shown in the lowest

array. Time is represented along the vertical dimension, with the first time step at the bottom and each succeeding one above the previous. The input at each time reflects the buffer contents. Because the buffer holds two sequence elements, note that the second element in the buffer at one time step (the activity pattern in input units 4–6) is the same as the first element of the buffer at the next (input units 1–3).

Above the input pattern are, respectively, the context unit activity levels after presentation of the final sequence element, the output unit activity levels at this time, and the target output values. The activity level of a unit is proportional to the area of its corresponding square. If a unit has an activity level of 0.0, its square has no area—an empty space. The squares are normalized such that a "unit square"—a square whose edge is the length of one of the vertical bars—corresponds to an activity level of 1.0. While the input, output, and target activity levels range from 0.0 to 1.0, the context activity levels can lie outside these bounds and are, in fact, occasionally greater than 1.0.

With these preliminaries out of the way, consider what the network has learned. At the completion of each sequence, the context unit activity pattern is essentially binary. Context unit 1 is off for _**BEAN**_ and _**BEAR**_, and on for _**DEAN**_ and _**DEAR**_; thus, it discriminates **B** and **D.** Context unit 2 is off for _**BEAN**_ and _**DEAN**_, and on for _**BEAR**_ and _**DEAR**_; thus it discriminates **N** and **R.** However, the context units do not behave in a straightforward way as Wickelements. If context unit 1 ware sharply tuned to, say, _**D,** the input-context weights should serve as a matched filter to the input pattern _**D.** This is not the case: the weights have signs $-+--+-$ but the _**D** input pattern is 110011. Nor is context unit 1 tuned to the **DE,** whose input pattern is 011010. Instead, the unit appears to be tuned equally to both patterns. By examining the activity of the unit over time, it can be determined that the unit is activated partly by _**D** and partly by **DE** but by no other input pattern. This makes sense: _**D** and **DE** are equally valid cues to the sequence identity, and as such, evidence from each should contribute to the response. To get a feel for why the detector responds as it does, note that _**D** (110011) is distinguished from _**B** (110001) by activity in unit 5; **DE** (011010) from **BE** (001010) by activity in unit 2. The weights from inputs 2 and 5 to context unit 1 are positive, allowing the unit to detect **D** in either context. The other weights are set so as to prevent the unit from responding to other possible inputs. Thus, the unit selects out key features of the Wickelements _**D** and **DE** that are not found in other Wickelements. As such, it behaves as a _**DE** Wickelement detector, and context unit 2 similarly as a **AR**_ detector.

Generalization testing supports the notion that the context units have become sensitive to these Wickelements. If the input elements are permuted to produce sequences like **AR_BE,** which preserves the Wickelements **AR**_ and _**BE,** context unit responses are similar to those of the original sequences. However, with permutations like _**RB**_, _**DAER**_, and **DEAR** (without the end delimiters), which destroy the Wickelements **AR**_ and _**BE,** context unit responses are not

contingent upon the **D, B, N,** and **R.** Thus, the context units are responding to these key letters, but in a context-dependent manner.

I must admit that the example in Figure 3 is fairly easy to interpret in part because the d_i and z_i were initially set to values near 1.0 and 0.0, respectively, and the learning rate for these parameters was turned down, forcing final solutions with values close to these initial ones. This encouraged the context units to produce a more sharply tuned "all-or-none" response to each sequence element.[4] Without biasing the d_i and z_i in this manner, the network was still able to discover solutions; in fact, the solutions were arrived at even more rapidly. These alternative solutions were qualitatively similar to the one described but were somewhat more difficult to interpret.

Learning the Regularities of Verb Past Tense

In English, the past tense of many verbs is formed according to a simple rule. Examples of these *regular verbs* are shown in Table 2. Each string denotes the phonetic encoding of the verb in italics, and each symbol a single phoneme. The notation of phonemes is the same as that used by Rumelhart and McClelland (1986), from whom the examples were borrowed. Regular verbs can be divided into three classes, depending on whether the past tense is formed by adding /ˆd/ (an "ud" sound, examples of which are shown in the first column in Table 2), /t/ (the second column), or /d/ (the third column). The rule for determining the class of a regular verb is as follows.

If the final phoneme is dental (/d/ or /t/), add /ˆd/;
else if the final phoneme is an unvoiced consonant, add /t/;
else (the final phoneme is voiced), add /d/.

A network was trained to classify the 60 examples in Table 2. Each phoneme was encoded by a set of four trinary acoustic features (see Rumelhart & McClelland, 1986, Table 5). The input layer of the network was a two-element buffer, so a verb like /kamp/ appeared in the buffer over time as **_k, ka, am, mp, p_.** The underscore is a delimiter symbol placed at the beginning and end of each string.

The network had eight input units (two time slices each consisting of four features), two context units, and three output units, one for each verb class. For comparison, both focused and full network architectures were studied. The full architecture was the same as the focused except it had complete connectivity in the context layer and an activation function like Equation 3 instead of Equation

[4]With d_i closer to 0.0, a context unit's activity depends primarily on recent sequence elements, allowing it to be sloppy with its response to earlier elements; likewise, with d_i much larger than 1.0, activity depends primarily on the early sequence elements, and the unit may be sloppy with respect to recent elements. With z_i closer to -0.5, all-or-none responses are not necessary because the effect of spurious activity can be canceled over time.

TABLE 2
Examples of Regular Verbs

+ /ˆd/	+ /t/	+ /d/
/dEpend/ (*depend*)	/ˆprOC/ (*approach*)	/Tretˆn/ (*threaten*)
/gId/ (*guide*)	/bles/ (*bless*)	/Ser/ (*share*)
/inklUd/ (*include*)	/diskˆs/ (*discuss*)	/ansˆr/ (*answer*)
/kˆmand/ (*command*)	/embarˆs/ (*embarrass*)	/dEskrIb/ (*describe*)
/mOld/ (*mold*)	/fAs/ (*face*)	/drI/ (*dry*)
/plEd/ (*plead*)	/help/ (*help*)	/fAr/ (*fare*)
/prOvId/ (*provide*)	/kamp/ (*camp*)	/frItˆn/ (*frighten*)
/rEgord/ (*regard*)	/kuk/ (*cook*)	/kUl/ (*cool*)
/sˆrWnd/ (*surround*)	/mark/ (*mark*)	/kˆntAn/ (*contain*)
/trAd/ (*trade*)	/nˆrs/ (*nurse*)	/krI/ (*cry*)
/SWt/ (*shout*)	/pˆrCˆs/ (*purchase*)	/lˆv/ (*love*)
/ˆtempt/ (*attempt*)	/pas/ (*pass*)	/mIn/ (*mine*)
/dEvOt/ (*devote*)	/pik/ (*pick*)	/prOgram/ (*program*)
/ekspekt/ (*expect*)	/prOdUs/ (*produce*)	/rEfuz/ (*refuse*)
/kˆnsist/ (*consist*)	/puS/ (*push*)	/rEvU/ (*review*)
/nOt/ (*note*)	/rEC/ (*reach*)	/sˆplI/ (*supply*)
/prEzent/ (*present*)	/rok/ (*rock*)	/stˆdE/ (*study*)
/reprEzent/ (*represent*)	/skraC/ (*scratch*)	/trembˆl/ (*tremble*)
/trEt/ (*treat*)	/trAs/ (*trace*)	/yUz/ (*use*)
/want/ (*want*)	/woS/ (*wash*)	/prEvAl/ (*prevail*)

1. The number of connections in each architecture was the same: the focused network requires six connections within the context layer, two for the d_i, two for the z_i, and two for the biases; the full network also requires six, four to connect each unit to each other and two for the biases. Learning rate parameters were adjusted to yield the best possible performance for each architecture.

Figure 4 shows performance on the training set for the two architectures, averaged over 15 runs with different initial random weights. A verb is considered to have been categorized correctly if the most active output unit specifies the verb's class. Both focused and full networks are able to learn the task, although the full network learns somewhat more quickly. Both networks have learned the underlying rule, as indicated by their excellent generalization performance on novel sequences (data points on far right of Figure 4).

Typical weights learned by the focused network are presented in Figure 5, along with the output levels of the two context units in response to 20 verbs. These verbs, though not part of the training set, were all classified correctly.

The response of the context units is straightforward. Context unit 1 has a positive activity level if the final phoneme is a dental (/d/ or /t/), negative otherwise. Context unit 2 has positive activity if the final phoneme is unvoiced, near zero otherwise. These are precisely the features required to discriminate among the three regular verb classes. In fact, the classification rule for regular

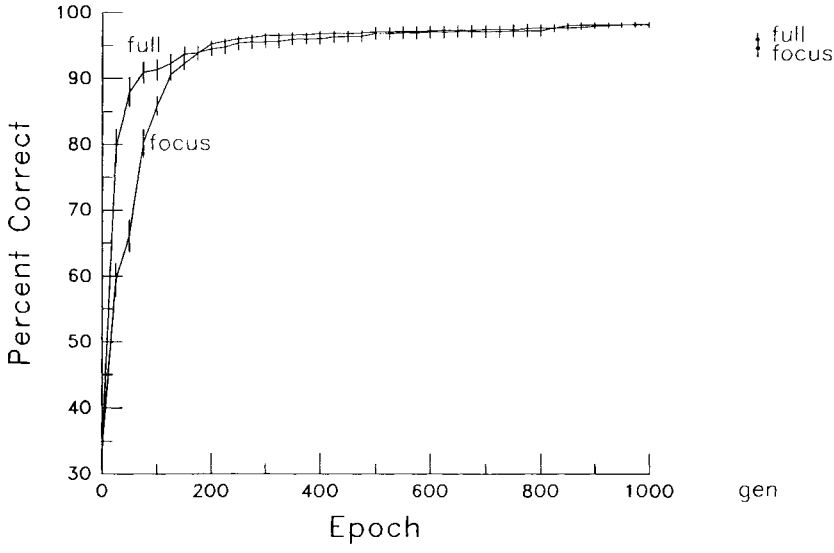

Figure 4. Mean performance on the regular verb task as a function of learning epoch for the focused and full recurrent architectures. The bars indicate one standard error of the mean in each direction. Data points for generalization performance are shown on the far right.

verbs can be observed in the context-output weights (the rightmost weight matrix in Figure 5). Connections are such that output unit 1, which represents the "add /ˆd/" class, is activated by a final dental phoneme; output unit 2, which represents the "add /t/" class, is activated by a final nondental unvoiced phoneme; and output unit 3, which represents "add /d/" class, is activated by a final nondental voiced phoneme.

Note that the decay weights in this simulation are small in magnitude; the largest is 0.02. Consequently, context units retain no history of past events, which is quite sensible because only the final phoneme determines the verb class. This fact makes verb classification a simple task: it is not necessary for the context units to hold on to information over time.

Consider now the opposite problem. Suppose the network is given the same verb classification task, but the order of phonemes is reversed; instead of /eksplAn/, /nAlpske/ is presented. In this problem, the relevant information comes at the start of the sequence and must be retained until the sequence is completed. Figure 6 shows performance on reversed regular verbs, averaged over 15 runs. The focused network is able to learn this task, with two context units, although the number of training epochs required is higher than for unreversed verbs. Generalization is as good for reversed as unreversed verbs. The full network, however, does not succeed with reversed verbs. In exploring a wide

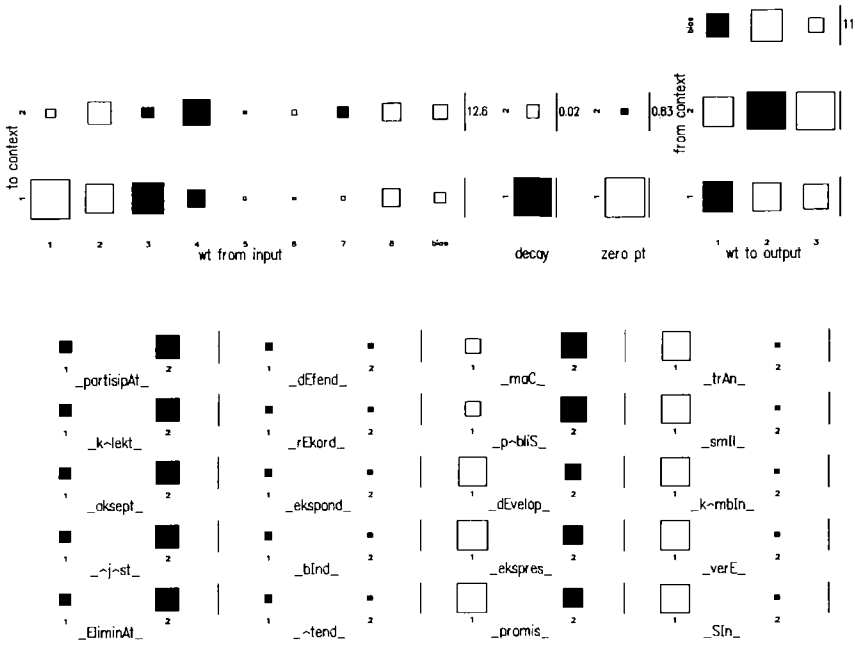

Figure 5. The regular verb problem. The upper half shows learned weights in the network, the lower half shows the final activity levels of the context units in response to a variety of verbs. Verbs in the first column all end with /t/, in the second column with /d/, in the third column with an unvoiced consonant, and the fourth column with a voiced consonant or vowel.

range of learning rate parameters, the highest single-run performance I was able to obtain was 75%. The difference between reversed and unreversed verbs is that the critical information for classification comes at the beginning of the sequence for reversed verbs but at the end of unreversed. In terms of the unfolded architecture of Figure 2b, this corresponds to a low layer of reversed but a high layer for unreversed. These results thus suggest that error signals are lost as they propagate back through the deeply layered full network. I return to this issue after describing several other simulations.

Learning to Reproduce a Sequence

In this task, the network is presented with a three-element input sequence, and then, following a fixed delay, must play back the sequence in time. The training sequences consisted of all permutations of three elements, **A, B,** and **C,** resulting in a total of six sequences; **ABC, ACB, BAC, BCA, CAB,** and **CBA.** An element was encoded by a binary activity pattern; **A** was 100, **B** 010, and **C** 001.

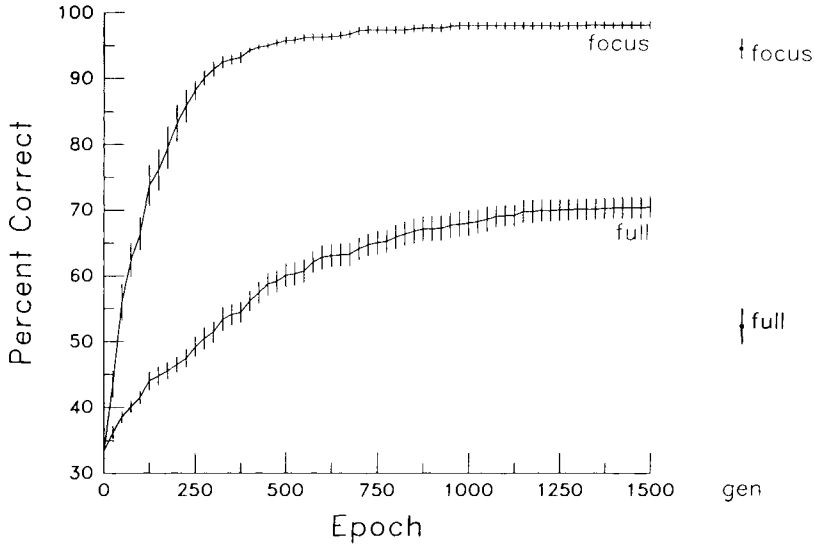

Figure 6. Mean performance on the reversed regular verb task as a function of learning epoch for the focused and full recurrent architectures. The bars indicate one standard error of the mean in each direction. Data points for generalization performance are shown on the far right.

The input layer contained three units on which the sequence was presented, one element per time step. At subsequent times, all inputs were zero. The order of events for **ABC** is presented in Table 3. In this example, there is a one time-step delay between the final element of the input sequence and the start of playback on the output units. Note that the target output levels are zero until playback commences.

The recurrent connections in the context layer allow the network to keep track of the input sequence. To help the network keep track of its position in the output sequence during playback, a set of one-to-one recurrent connections was added from the output units to three additional input units. These three units represented the output at the previous time step (an architecture suggested by Jordan, 1987). During training, these units were set to the *target* output values; for the example in Table 3, these inputs would be zero from times 1–5, 100 at time 6 and 010 at time 7. During testing, the true output values from the previous time step were "quantized" and copied back to the input. Quantization entailed setting all output levels greater than 0.5 to 1.0 and others to 0.0.

The network was made up of six input units, three for the current sequence element and three for the previous output state, three context units, and three

TABLE 3
Sequence of Input and Target Output Patterns for ABC

Time Step	*Input*	*Target Output*
1	100 (A)	000
2	010 (B)	000
3	001 (C)	000
4	000	000
5	000	100 (A)
6	000	010 (B)
7	000	001 (C)

output units. The task of the context units was to learn a static representation of the sequence that could be used in regenerating the sequence.

Fifteen replications of the simulation were run with random initial weights for both focused and full network architectures. The focused network had two-thirds as many adjustable parameters within the context layer as the full, six instead of nine.

Performance was judged using the quantized outputs. The task was successfully learned on all runs. The mean number of training epochs required for perfect performance was 767 for the focused network and 620 for the full network. Although the focused network took a bit longer to learn, this difference was not statistically reliable ($t(28) = 0.958$, $p > 0.3$). Figure 7 shows a typical weight configuration obtained by the focused network and its response to **ABC.** The weights are quite interesting. It appears that each context unit handles a particular symbol. For example, context unit 3 (the top row of the weight arrays) is excited by both **A** in the input and **A** as the previous output, and it has the effect of inhibiting **A** on the output. Similarly, context unit 2 is tuned to **C** and unit 1 to **B.**

The sequence reproduction task becomes more difficult to learn as the delay between input and playback is increased. In the above example, the delay was one time step. Simulations were also run at a four-time-step delay. Training continued until performance was perfect, up to a maximum of 15,000 epochs. The focused network was able to learn the task perfectly on 12 of 15 runs, the full network on only 2 of 15. Mean performance over all runs following training was 98.5% for the focused network, but only 72.9% for the full. This difference was significant ($t(28) = 8.42$, $p < 0.001$).

Increasing the playback delay increases the time lag between the critical input information and the start of the response. The full network appears able to learn only when the critical input shortly precedes the response, whereas the focused network is able to learn with extended time lags. This conclusion was also suggested by the regular verb results.

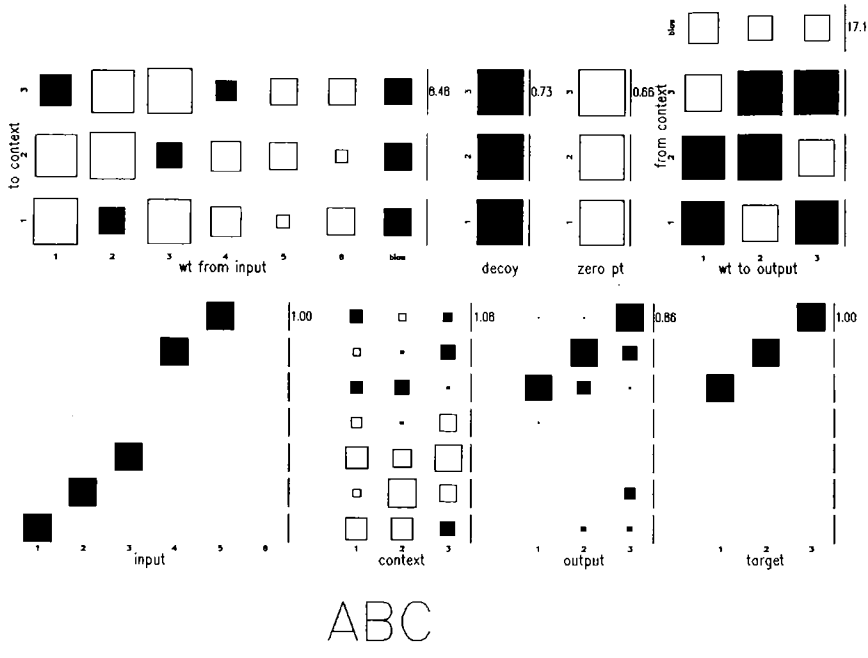

Figure 7. The sequence reproduction problem. The upper half of the figure shows learned weights in the focused network, the lower half shows input, context, output, and target activity over time for the sequence **ABC**. The sequence-to-be-reproduced is encoded on input units 1–3; the quantized output from the previous time step is encoded on input units 4–6.

Large Verb Simulation

To study a more difficult task, the regular-verb categorization problem was extended to a larger corpus of verbs. As before, the task was to classify each verb according to the manner in which its past tense is formed. The complexity of the task was increased by including both regular and irregular verbs, 136 training instances altogether, and a total of 13 response categories, 3 for regular forms and 10 for irregular. The response categories and number of training instances in each category are listed in Table 4. The categories are based loosely on a set suggested by Bybee and Slobin (1982).

The corpus of verbs was borrowed from a psychological model of Rumelhart and McClelland (1986) designed to account for children's acquisition of verb past tenses. This model would produce the past tense of a verb given its infinitive form as input. The representation used as both input and output from the model is a Wickelement encoding of the verb, each Wickelement encoding a particular

TABLE 4
Verb Classification

Category Number	*Instances in Category*	*Examples*	*Category (How Past Tense is Formed)*
1	20	*explain (explained)* *cry (cried)*	regular verb, add /d/
2	20	*dance (danced)* *pack (packed)*	regular verb, add /t/
3	20	*reflect (reflected)* *guide (guided)*	regular verb, add /ˆd/
4	7	*beat (beat)* *put (put)*	no change
5	3	*send (sent)* *build (built)*	change a final /d/ to /t/
6	8	*deal (dealt)* *mean (meant)*	internal vowel change and add a final /t/
7	6	*do (did)* *sell (sold)*	internal vowel change and add a final /d/
8	5	*bring (brought)* *teach (taught)*	internal vowel change, delete final consonant, and add a final /t/
9	5	*have (had)* *make (made)*	internal vowel change, delete final consonant, and add a final /d/
10	4	*swim (swam)* *ring (rang)*	internal vowel change of /i/ to /a/
11	17	*feed (fed)* *get (got)*	internal vowel change and stem ends in a dental
12	20	*begin (begun)* *break (broke)*	other internal vowel change
13	1	*go (went)*	*go* in a category by itself

phonetic feature in the context of two neighboring phonetic features. Because this static representation is built into the model, the model did not require temporal dynamics. My interest in studying this problem was to see whether the focused recurrent network could, given time-varying inputs, learn the task. Because the focused architecture is tailored to learning Wickelement representations, if it is able to learn the task then it must have learned a static representation somewhat like the Wickelement representation presupposed by Rumelhart and McClelland's model.

The task is difficult. The verb classes contain some internal regularities, but these regularities are too weak to be used to uniquely classify a verb. For instance, all verbs in category 3 end in a /d/ or /t/, but so do verbs in categories 4, 5, and 11. Whether a verb ending in /d/ or /t/ belongs in category 3 or one of the other categories depends on whether it is regular, but there are no simple

features signaling this fact. Further, fine discriminations are necessary because two outwardly similar verbs can be classified into different categories. *Swim* and *sing* belong to category 10, but *swing* to category 12; *ring* belongs to category 10, but *bring* to category 8; *set* belongs to category 4, but *get* to category 11. Finally, the task is difficult because some verbs belong in multiple response categories; for example, *sit* could go in either category 10 or 11. The lowest category number was chosen in these cases.

Because the category to which a verb belongs is somewhat arbitrary, the network must memorize a large number of special cases. (Indeed, an earlier version of these simulations were run in which the target responses were incorrect for about 15% of the items. The network learned the task just as well, if not a bit faster than in the simulations reported here.)

The network architecture was similar to that used in the regular verb example. The input layer was a two-phoneme buffer, and the encoding of phonemes was the same as before. The output layer consisted of 13 units, one for each verb class. Both focused and full network architectures were simulated. To match the two networks on number of connections, 25 context units were used in the focused network, 16 in the full; this resulted in 613 weights for the focused network and 621 for the full network.

Figure 8 shows performance on the training set for the two architectures,

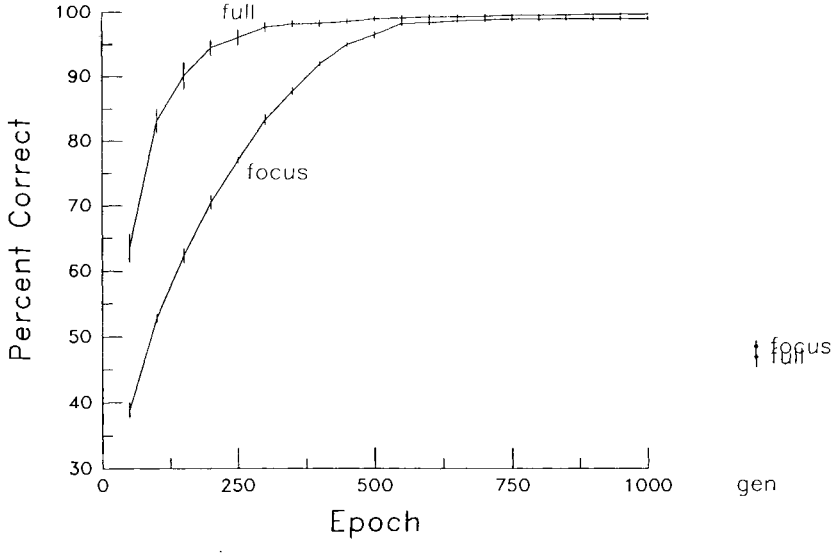

Figure 8. Mean performance on the large verb problem as a function of learning epoch for the focused and full recurrent architectures. The bars indicate one standard error of the mean in each direction. Data points for generalization performance are shown on the far right.

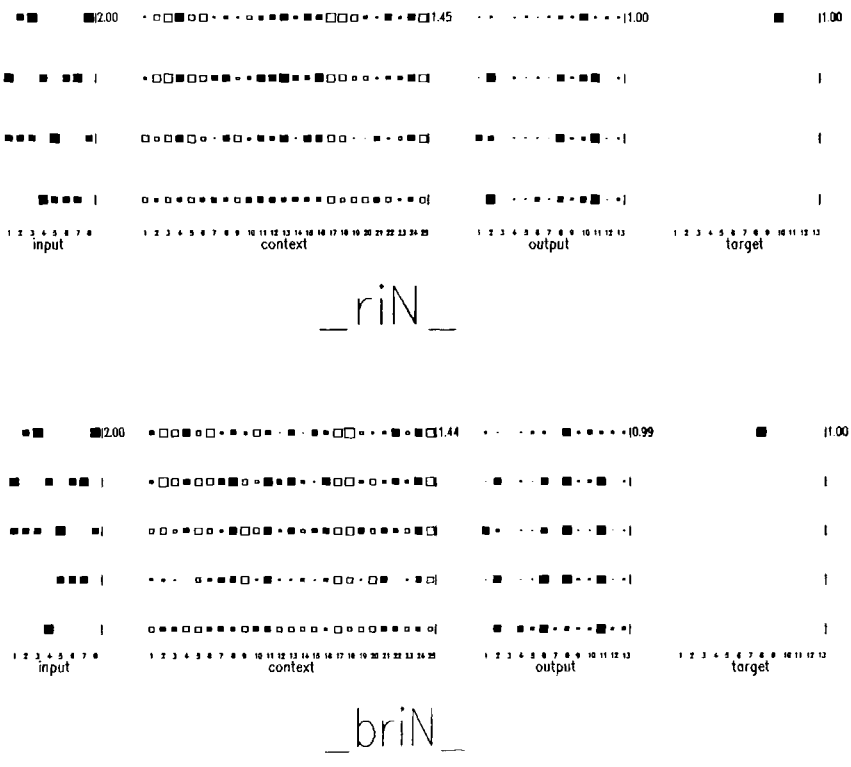

Figure 9. The large verb task. The upper portion of the figure shows input, context, output, and target activity over time for the sequence _**riN**_ (*ring*), the lower portion for the sequence _**briN**_ (*bring*).

averaged over 10 runs with different initial random weights. A verb is considered to have been categorized correctly if the most active output unit specifies the verb's class. Both focused and full networks are able to learn the task, although the full network learns somewhat more quickly. Errors observed during training seemed quite reasonable. Verbs are sometimes "overregularized," as when *eat* becomes *eated*. Overgeneralization occurs in other respects, as when *sit* was misclassified in category 4—verbs whose past tense is the same as the root—presumably by analogy to *hit* and *fit* and *set*. Surprisingly, neither the full nor focused net had difficulty learning category 13, although it contained only a single verb—*go*.

Generalization performance on novel sequences is poor for both networks (data points on far right of Figure 8), but this is readily explained. The corpus provided by Rumelhart and McClelland had 420 verbs altogether. To normalize across categories, at most 20 verbs from each category were used in the training

set. Consequently, the regular verb classes were approximately the same size as the irregular classes, eliminating any a priori bias toward classifying an unfamiliar verb as regular. The verbs from the corpus not used in training were used for generalization testing; these verbs were almost exclusively from the three regular verb categories. Thus, the network attempted to classify the unfamiliar regular verbs without any expectation that the verbs would be regular. Most all errors involved mistaking the verbs to be irregular.

Typical responses of the network are presented in Figure 9. The two sequences shown, _**riN**_ (*ring*) and _**briN**_ (*bring*), have quite similar input patterns yet produce different outputs: _**riN**_ belongs in category 10 and _**briN**_ in category 8. Due to the size of the network, interpreting the behavior of individual context units and how they serve to distinguish two inputs like _**riN**_ and _**briN**_ is extremely difficult.

EVALUATION OF THE FOCUSED ARCHITECTURE

The simulations reported above are typical of results I have obtained with the full and focused architectures. For both architectures, learning becomes more difficult as the delay between the critical input and the response is increased. This was observed in two simulations: the regular verbs and the sequence reproduction task. While this difficulty is manifested in slowed learning for the focused architecture, its effect on the full architecture is far more devastating. The full architecture is simply unable to learn tasks that involve long intervals between critical input and response. Not all tasks are of this nature, however. For tasks in which the information contained in the input is more evenly distributed across time (e.g., the large verb simulation), the full network appears to learn in fewer training cycles when full and focused networks are matched on total number of connections.

Nonetheless, the focused architecture shows advantages over two competing approaches: the back propagation unfolding-in-time procedure and the real-time recurrent learning (RTRL) algorithm of Williams and Zipser (1989, this volume). Learning in the focused architecture is less computation intensive than the unfolding-in-time procedure because back propagation of the error signal in time is avoided. The focused architecture requires about two-thirds as many floating-point operations per training cycle as the unfolding-in-time procedure. This savings is achieved whether the network is implemented in serial or parallel hardware. Although the focused architecture turns out to be a special case of the RTRL algorithm, the space requirements of the more general RTRL algorithm are far worse. In RTRL, the number of internal state variables grows with the cube of the number of units, whereas in the focused architecture the number of internal state variables grows with only the product of the numbers of input and context units.

Scaling Properties

A critical question to be asked of any network architecture is how well its performance will scale as the problem size increases. The focused architecture promises to scale better than the full architecture with respect to the sequence length. The reasoning is as follows. As I discussed previously, any recurrent architecture (e.g., Figure 2a) can be unfolded in time to obtain a computationally equivalent feedforward network (Figure 2b). The *depth* of this unfolded network increases with sequence length. However, an unfolded version of the focused architecture can be constructed with a fixed depth and a *breadth* that increases with sequence length (Figure 10). The input units and context units are replicated for each time step of the sequence. Each set of context units is activated solely by the input at the corresponding time step. In the third layer of the network, the net activity of context unit i is computed by taking a weighted sum of unit i's activity at each time τ from τ - 1, . . . , t. This simple summation is possible because the integration of context unit activity over time is linear. That is, the context unit activation equation

$$c_i(t) = d_i c_i(t - 1) + s[net_i(t)] \quad \text{(I)}$$

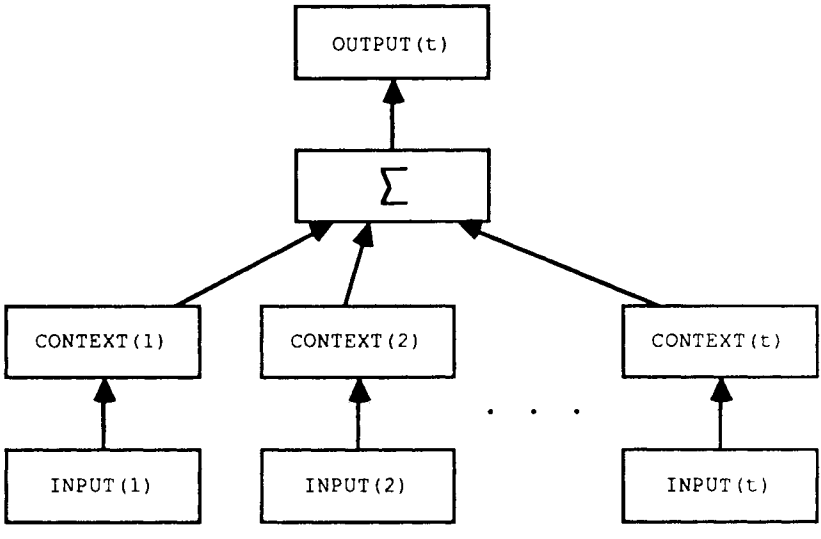

Figure 10. An unfolded version of the focused architecture having four layers. Input and context units are replicated for each time step of the input sequence. The activity of each context unit is summed over time in the third layer, weighted by a time-dependent decay factor.

can be rewritten in closed form as

$$c_i(t) = \sum_{\tau=1}^{t} d_i^{t-\tau}\, s[net_i(\tau)].$$

Each set of units in the second layer of Figure 10 computes $s[net_i(\tau)]$. The third layer then sums the $s[net_i(\tau)]$ across τ, weighted by the decay factor $d_i^{t-\tau}$, to obtain $c_i(t)$.

Consider the situation when all d_i are near 1.0, as they are set at the start of a simulation. Information from all times will be integrated with equal weight; no matter when in time an input appears, it will be treated uniformly. If the desired response of the network depends on a critical input at a particular time, it will not matter when in time this input occurs. Further, increasing the length of a sequence serves only to add background noise against which the critical input must be detected.[5]

To recap, longer sequences translate to a greater effective depth of the full architecture, but a greater effective breadth of the focused architecture. As I argued earlier, one has reason to suspect that deep and narrow networks are more troublesome for back propagation than shallow and wide ones. If so, the focused network should scale better with respect to sequence length.

Indeed, comparisons of the full and focused architectures reported here can be interpreted as support for this claim. Consider the regular verb example. When the verbs are presented unreversed, only the final sequence element is critical for classification. Thus, although the unfolded full network may have as many layers as sequence elements, the *effective* depth to which the network must attend is quite shallow. Reversing the verbs increases the effective depth. The comparison of unreversed and reversed verbs in the full network is therefore a test of scaling as the effective depth increases; the same comparison in the focused network is a test of scaling as the effective breadth increases. In this case, greater depth is clearly more detrimental than greater breadth.

The focused and full networks differ along two dimensions. The focused network has 1-1 connections in the context layer and the context unit activation function is linear; the full network has complete connectivity in the context layer and a nonlinear context unit integration function (one in which the recurrent connections are contained within the squashing function). The depth-versus-breadth result is contingent on linear integration, not on 1-1 connections within the context layer. As discussed earlier, several researchers have examined a third architecture with 1-1 connections and nonlinear integration. This architecture does not seem to perform well, as one might predict on the basis of the nonlinear

[5]Note that if the decay terms become much less or greater than 1.0, there becomes a bias toward recent or distant information, respectively. It is thus important to start the system with initial decay terms near 1.0 and to change them slowly.

integration function. Note that we have not yet explored a fourth and potentially promising architecture, one with complete connectivity in the context layer and a linear integration function.

Problems with the Approach

Despite reasonable success with the focused architecture, some difficulties should be pointed out. First, instability problems arise if the decay values become larger than 1.0 because such values allow a unit's activity to grow exponentially over time. In practice, this is not a serious problem as long as learning rates for the decay connections are kept small. Nonetheless, the final activity levels of the context units can become ridiculously large, particularly on generalization testing if the novel patterns are longer than the training patterns. For example, in the reversed regular verb problem, generalization testing occasionally produced context unit activity levels above 25. One possible solution is to constrain the allowed values of the decay connections. I have tried restricting the allowed values to the interval 0–1. Generally, this restriction increases the number of learning trials required, but does improve stability and generalization performance.

A second criticism of the focused architecture is that it uses an input buffer. This buffer was motivated by the desire to train context units to respond to Wickelements, but is not strictly necessary. Without a buffer, though, the context units are unable to obtain nonlinear interactions across time. For instance, a single unit cannot be tuned to respond sharply to input **A** followed by input **B** but not to either **A** or **B** in isolation. No matter, the buffer used in the focused architecture is altogether different from that required by the naive buffer model presented in the introduction. The buffer in the buffer model specifies a temporal window over which information integration can occur, whereas the focused architecture's buffer specifies a temporal window over which nonlinear interactions can occur. The focused architecture will generally not need as large a buffer as the buffer model. For example, the past-tense network had only a two-slot buffer, whereas a buffer model would likely require as many slots as there are phonemes in the longest verb.

A final difficulty with the focused architecture is that, while it may be appropriate for relatively short sequences, it is unclear how well the approach will work on long sequences in which very little information is contained in a single sequence element, such as a speech recognition task with the time-domain waveform as input. Of course, this sort of problem is difficult for the full architecture as well. One solution is to extend the buffer size to capture significant segments of the input. It would seem a more promising solution, however, to preprocess the input in some manner, perhaps using unsupervised learning mechanisms, to obtain higher-order features which could then be fed into the recognition system.

APPENDIX: DERIVATION OF THE FOCUSED BACK-PROPAGATION ALGORITHM

Assume the following situation: a t-time-step sequence has been presented and at time t a desired output is specified that allows for the computation of an error signal. The problem is to determine two quantities: the error gradient with respect to the recurrent connections ($\partial E/\partial d_i$) and with respect to the input-context connections ($\partial E/\partial w_{ji}$).

Beginning with the recurrent connections, the chain rule can be used to expand $\partial E/\partial d_i$:

$$\frac{\partial E}{\partial d_i} = \frac{\partial E}{\partial c_i(t)} \frac{\partial c_i(t)}{\partial d_i},$$

and $\partial E/\partial c_i(t)$ can be computed directly by back-propagating from the output layer to the context layer at time t. Thus, the problem is to determine $\partial c_i(t)/\partial d_i$. Given that

$$c_i(t) = d_i c_i(t-1) + s[net_i(t)] \tag{A.1}$$

and

$$net_i(\tau) \equiv \sum_k w_{ki} x_k(\tau) \tag{A.2}$$

(Equation 1 from the main text), and assuming the initial condition $c_i(0) = 0$, the difference equation (A.1) can be rewritten in closed form as

$$c_i(t) = \sum_{\tau=1}^{t} d_i^{t-\tau}\, s[net_i(\tau)]. \tag{A.3}$$

Defining

$$\alpha_i(t) \equiv \frac{\partial c_i(t)}{\partial d_i}.$$

by substituting $c_i(t)$ from Equation A.3 and computing the partial derivative, we obtain

$$\alpha_i(t) = \frac{\partial}{\partial d_i}\left[\sum_{\tau=1}^{t} d_i^{t-\tau}\, s[net_i(\tau)]\right]$$

$$= \sum_{\tau=1}^{t-1} (t-\tau)\, d_i^{t-\tau-1}\, s[net_i(\tau)]. \tag{A.4}$$

Regrouping the terms gives

$$\alpha_i(t) = \sum_{k=1}^{t-1} \sum_{\tau=1}^{k} d_i^{t-\tau-1}\, s(net_i(\tau)]$$

$$= \sum_{k=1}^{t-1} d_i^{t-k-1} \sum_{\tau=1}^{k} d_i^{k-\tau}\, s[net_i(\tau)]. \tag{A.5}$$

Combining Equations A.3 and A.5, we have

$$\alpha_i(t) = \sum_{k=1}^{t-1} d_i^{t-k-1}\, c_i(\mathrm{k}).$$

Removing the $k = t - 1$ term from the summation and factoring out d_i, we obtain

$$\alpha_i(t) = c_i(t-1) + d_i \sum_{k=1}^{t-2} d_i^{t-k-2}\, c_i(k). \tag{A.6}$$

From Equation A.4, the summation in Equation A.6 can be replaced by $\alpha_i(t-1)$ to yield the incremental expression:

$$\alpha_i(t) = c_i(t-1) + d_i\alpha_i(t-1).$$

Following a similar derivation for the input-context connections, we can expand $\partial E/\partial w_{ji}$:

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial c_i(t)} \frac{\partial c_i(t)}{\partial w_{ji}}.$$

As stated, $\partial E/\partial c_i(t)$ can be computed directly by back-propagating from the output layer to the context layer at time t. Thus, the problem is to determine $\partial c_i(t)/\partial w_{ji}$. Defining

$$\beta_{ji}(t) \equiv \frac{\partial c_i(t)}{\partial w_{ji}},$$

by substituting $c_i(t)$ from Equation A.3 and computing the partial derivative, we obtain

$$\beta_{ji}(t) = \frac{\partial}{\partial w_{ji}} \left[\sum_{\tau=1}^{t} d_i^{t-\tau}\, s[net_i(\tau)] \right].$$

Using Equation A.2 to compute the derivative of $s[net_i(\tau)]$ gives

$$\beta_{ji}(t) = \sum_{\tau=1}^{t} d_i^{t-\tau}\, s'[net_i(\tau)]x_j(\tau). \tag{A.7}$$

Removing the $\tau = t$ term from the summation and factoring out d_i, we obtain

$$\beta_{ji}(t) = s'[net_i(t)]x_j(t) + d_i \sum_{\tau=1}^{t-1} d_i^{t-\tau-1} s'[net_i(\tau)]x_j(\tau). \qquad (A.8)$$

From Equation A.7, the summation in Equation A.8 can be replaced by $\beta_{ji}(t-1)$ to yield the incremental expression

$$\beta_{ji}(t) = s'[net_i(t)]x_j(t) + d_i\beta_{ji}(t-1).$$

ACKNOWLEDGMENTS

Thanks to Jeff Elman, Yoshiro Miyata, Yves Chauvin, and Geoff Hinton for their insightful comments and assistance. The graphical displays of network states are due to Yoshiro's code. Dave Rumelhart and Jay McClelland were kind enough to provide me with the phonological encoding and classification of verbs from their simulation work. This research was supported by NSF Presidential Young Investigator award IRI-9058450 and grant 90-21 from the James S. McDonnell Foundation, as well as grant 87-2-36 from the Alfred P. Sloan Foundation to Geoffrey Hinton.

REFERENCES

Almeida, L. (1987). A learning rule for asynchronous perceptions with feedback in a combinatorial environment. In M. Caudill & C. Butler (Eds.), *Proceedings of the IEEE First Annual International Conference on Neural Networks* (Vol. 2, pp. 609–618). San Diego, CA: IEEE Publishing Services.

Bachrach, J. (1988). *Learning to represent time.* Unpublished master's thesis, University of Massachusetts, Amherst.

Bybee, J. L., & Slobin, D. I. (1982). Rules and schemas in the development and use of the English past tense. *Language, 58,* 265–289.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science, 14,* 179–212.

Elman, J. L., & McClelland, J. L. (1986). Exploiting lawful variability in the speech wave. In J. S. Perkell & D. H. Klatt (Eds.), *Invariance and variability in speech processes* (pp. 360–380). Hillsdale, NJ: Lawrence Erlbaum Associates.

Elman, J. L., & Zipser, D. (1988). Learning the hidden structure of speech. *Journal of the Acoustical Society of America, 83,* 1615–1625.

Freyd, J. (1987). Dynamic mental representations. *Psychological Review, 94,* 427–438.

Gori, M., Bengio, Y., & Mori, R. de (1989). BPS: A learning algorithm for capturing the dynamic nature of speech. In *Proceedings of the First International Joint Conference on Neural Networks* (Vol. 2, pp. 417–423).

Hinton, G. (1987). Learning distributed representations of concepts. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society* (pp. 1–12). Hillsdale, NJ: Lawrence Erlbaum Associates.

Hinton, G. (1989). Connectionist learning procedures. *Artificial Intelligence, 40,* 185–234.

Jordan, M. I. (1987). Attractor dynamics and parallelism in a connectionist sequential machine. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society* (pp. 531–546). Hillsdale, NJ: Lawrence Erlbaum Associates.

Landauer, T. K., Kamm, C. A., & Singhal, S. (1987). Teaching a minimally structured back propagation network to recognize speech. In *Proceedings of the Ninth Annual Conference of the Cognitive Science Society* (pp. 531–536). Hillsdale, NJ: Lawrence Erlbaum Associates.

Lang, K. (1987). *Connectionist speech recognition.* Unpublished Ph.D. thesis proposal, Carnegie-Mellon University, Pittsburgh, PA.

Lapedes, A., & Farber, R. (1987). *Nonlinear signal processing using neural networks* (Report No. LA-UR-87-2662). Los Alamos, NM.

McClelland, J. L., & Elman, J. L. (1986). Interactive processes in speech perception: The TRACE model. In J. L. McClelland & D. E. Rumelhart (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition. Volume II: Psychological and biological models* (pp. 58–121). Cambridge, MA: MIT Press.

Miyata, Y. (1988). *The learning and planning of actions* (ICS Technical Report 8802). La Jolla, CA: University of California, San Diego, Institute for Cognitive Science.

Mozer, M. C. (1990). Discovering faithful "Wickelfeature" representations in a connectionist network. In *Proceedings of the 12th Annual Conference of the Cognitive Science Society* (pp. 356–363). Hillsdale, NJ: Lawrence Erlbaum Associates.

Mozer, M. C. (1991). *The perception of multiple objects: A connectionist approach.* Cambridge, MA: MIT Press/Bradford Books.

Pineda, F. (1987). Generalization of back propagation to recurrent neural networks. *Physical Review Letters, 19,* 2229–2232.

Plaut, D. C., Nowlan, S., & Hinton, G. E. (1986). *Experiments on learning by back propagation* (Technical report CMU-CS-86-126). Pittsburgh, PA: Carnegie-Mellon University, Department of Computer Science.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error propagation. In D. E. Rumelhart & J. L. McClelland (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition. Volume I: Foundations* (pp. 318–362). Cambridge, MA: MIT Press/Bradford Books.

Rumelhart, D. E., & McClelland, J. L. (1986). On learning the past tenses of English verbs. In J. L. McClelland & D. E. Rumelhart (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition. Volume II: Psychological and biological models* (pp. 216–271). Cambridge, MA: MIT Press/Bradford Books.

Seidenberg, M. S. (1990). Word recognition and naming: A computational model and its implications. In W. D. Marslen-Wilson (Ed.), *Lexical representation and process.* Cambridge, MA: MIT Press.

Sejnowski, T. J., & Rosenberg, C. R. (1987). Parallel networks that learn to pronounce English text. *Complex Systems, 1,* 145–168.

Smolensky, P. (1983). Schema selection and stochastic inference in modular environments. In *Proceedings of the Sixth Annual Conference on Artificial Intelligence AAAI-83* (pp. 109–113).

Stornetta, W. S., Hogg, T., & Huberman, B. A. (1987). A dynamical approach to temporal pattern processing. In *Proceedings of the IEEE Conference on Neural Information Processing Systems.*

Tank, D., & Hopfield, J. (1987). *Proceedings of the National Academy of Sciences, 84,* 1896.

Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., & Lang, K. (1987). *Phoneme recognition using time-delay neural networks* (Technical report TR-1-0006). Japan: ATR Interpreting Telephony Research Labs.

Watrous, R. L., & Shastri, L. (1987). Learning acoustic features from speech data using connectionist networks. In *Proceedings of the Ninth Annual Conference of the Cognitive Science Society* (pp. 518–530). Hillsdale, NJ: Lawrence Erlbaum Associates.

Wickelgren, W. (1969). Context-sensitive coding, associative memory, and serial order in (speech) behavior. *Psychological Review, 76,* 1–15.

Williams, R. J., & Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Computation, 1,* 270–280.