Michael C. Mozer
Department of Computer Science and Institute of Cognitive Science, University of Colorado, Boulder, CO 80309–0430

Neural Net Architectures for Temporal Sequence Processing

I present a general taxonomy of neural net architectures for processing time-varying patterns. This taxonomy subsumes many existing architectures in the literature and points to several promising architectures that have yet to be examined. Any architecture that processes time-varying patterns requires two conceptually distinct components: a short-term memory that holds on to relevant past events and an associator that uses the short-term memory to classify or predict. My taxonomy is based on a characterization of short-term memory models along the dimensions of form, content, and adaptability. Experiments on predicting future values of a financial time series (U.S. dollar–Swiss franc exchange rates) are presented using several alternative memory models. The results of these experiments serve as a baseline against which more sophisticated architectures can be compared.

Times Series Prediction: Forecasting the Future and
Understanding the Past, Eds. A. S. Weigend and N. A. Gershenfeld, SFI Studies

INTRODUCTION

Neural networks have proven to be a promising alternative to traditional techniques for nonlinear temporal prediction tasks (e.g., Curtiss, Brandemuehl, & Kreider, 1992; Lapedes & Farber, 1987; and Weigend, Huberman, & Rumelhart, 1992). However, temporal prediction is a particularly challenging problem because conventional neural net architectures and algorithms are not well suited for patterns that vary over time. The prototypical use of neural nets is in *structural pattern recognition*. In such a task, a collection of features—visual, semantic, or otherwise—is presented to a network and the network must categorize the input feature pattern as belonging to one or more classes. For example, a network might be trained to classify animal species based on a set of attributes describing living creatures such as "has tail," "lives in water," or "is carnivorous"; or a network could be trained to recognize visual patterns over a two-dimensional pixel array as a letter in $\{A, B, \ldots, Z\}$. In such tasks, the network is presented with all relevant information simultaneously.

In contrast, *temporal pattern recognition* involves processing of patterns that evolve over time. The appropriate response at a particular point in time depends not only on the current input, but potentially on all previous inputs. This is illustrated in Figure 1, which shows the basic framework for a temporal prediction problem. I assume that time is quantized into discrete steps, a sensible assumption because many time series of interest are intrinsically discrete, and continuous series can be sampled at a fixed interval. The input at time t is denoted $\mathbf{x}(t)$. For univariate series, this input is a scalar but, to allow for a multivariate series, $\mathbf{x}$ should be considered as vector valued. The output of the predictor at time t, $\mathbf{y}(t)$ is based on the input sequence $\mathbf{x}(1) \ldots \mathbf{x}(t)$ and represents some future value of the input or possibly some function of a future value; $\mathbf{y}$ can also be vector valued. A prediction $\mathbf{y}$ can be made at every time step or only at selected times. Prediction involves two conceptually distinct components. The first is to construct a short-term memory that retains aspects of the input sequence relevant to making predictions. The second makes a prediction based on the short-term memory. In a neural net framework, the predictor will always be a feedforward component of the net, while the short-term memory will often have internal recurrent connections. ZhangZhangX. and Hutchinson (this volume) and de Vries and Principe (1992) also make the distinction between these two components.

In designing a neural net model, one must consider the following issues:

- **Architecture**: What is the internal structure of the short-term memory and predictor networks? Answering this question involves specifying the number of layers and units, the pattern of connectivity among units, and the activation dynamics of the units.

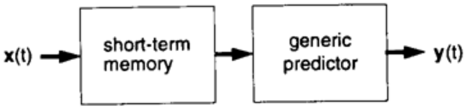

FIGURE 1 Abstract formulation of the temporal prediction task. $\mathbf{x}(t)$ denotes a vectorial representation of the input at time t; $\mathbf{y}(t)$ denotes a prediction at time t based on the input sequence $\mathbf{x}(1)\ldots\mathbf{x}(t)$. The short-term memory retains aspects of the input sequence that are relevant to the prediction task.

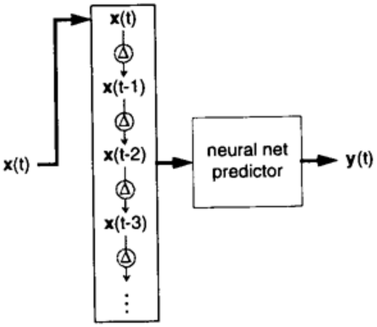

FIGURE 2 Tapped delay-line memory model. Each "Δ" operator introduces a one-time step delay.

- **Training**: Given a set of training examples, how should the internal parameters in the model—the connection *weights*—be adjusted? Each training example i consists of an input series, $\{\mathbf{x}_i(1), \mathbf{x}_i(2), \mathbf{x}_i(3), \ldots, \mathbf{x}_i(t_i)\}$, and an associated series of predictions, $\{\mathbf{p}_i(1), \mathbf{p}_i(2), \mathbf{p}_i(3), \ldots, \mathbf{p}_i(t_i)\}$, where t_i is the number of steps in the example and $\mathbf{p}_i(\tau)$ is a target prediction at time τ. Training a neural net model involves setting its weights such that its predictions, $\mathbf{y}_i(\tau)$, come as close as possible to the target predictions, $\mathbf{p}_i(\tau)$, usually in a least-squares sense.
- **Representation**: How should the time-varying input sequence be represented in the short-term memory? The nature and quantity of information that must go into the memory is domain dependent.

In this chapter, I focus on the representation issue. However, the issues of architecture, network dynamics, training procedure, and representation are intricately related and, in fact, can be viewed as different perspectives on the same underlying problems. A given choice of representation may demand a certain neural net architecture or a particular type of learning algorithm to compute the representation. Conversely, a given architecture or learning algorithm may restrict the class of representations that can be accommodated.

I address the representation issue by characterizing the space of neural network short-term memory models. In the following sections, I present three dimensions along which the memory models vary: *memory form*, *content*, and *adaptability*. This framework points to interesting memory models that have not yet been explored in the neural net literature.

FORMS OF SHORT-TERM MEMORY

TAPPED DELAY-LINE MEMORY

The simplest form of memory is a buffer containing the n most recent inputs. Such a memory is often called a *tapped delay-line* model because the buffer can be formed by a series of delay lines (Figure 2). It is also called a *delay space embedding* and forms the basis of traditional statistical autoregressive (AR) models. Tapped delay-line memories are common in neural net models (e.g., Elman & McClelland, 1986; Elman & Zipser, 1988; Lapedes & Farber, 1987; Plaut & Hinton, 1987; Waibel et al., 1989; Wan, this volume; and Zhang & Hutchinson, this volume). These memories amount to selecting certain elements of a sequence $\mathbf{x}(1)\ldots\mathbf{x}(t)$, say, a total of Ω elements, and forming a state representation $(\bar{\mathbf{x}}_1(t), \bar{\mathbf{x}}_2(t), \bar{\mathbf{x}}_3(t), \ldots \bar{\mathbf{x}}_\Omega(t))$, where $\bar{\mathbf{x}}_i(t) = \mathbf{x}(t-i+1)$.

A minor extension of this formulation to permit nonuniform sampling of past values involves specifying varying delays such that $\bar{\mathbf{x}}_i(t) = \mathbf{x}(t-\omega_i)$, where ω_i is the (integer) delay associated with component i.

Let me present another formalism for characterizing the delay-line memory which is broad enough to encompass the other forms of memories to be discussed. One can treat each $\bar{\mathbf{x}}_i(t)$ as a convolution of the input sequence with a kernel function, c_i:

$$\bar{\mathbf{x}}_i(t) = \sum_{\tau=1}^{t} c_i(t-\tau)\mathbf{x}(\tau),$$

where, for delay-line memories,

$$c_i(t) = \begin{cases} 1, & \text{if } t = \omega_i \; ; \\ 0, & \text{otherwise.} \end{cases}$$

Figure 3(a) shows the kernel function. By substituting a different kernel function, one obtains the forms of memories discussed in the next three sections.

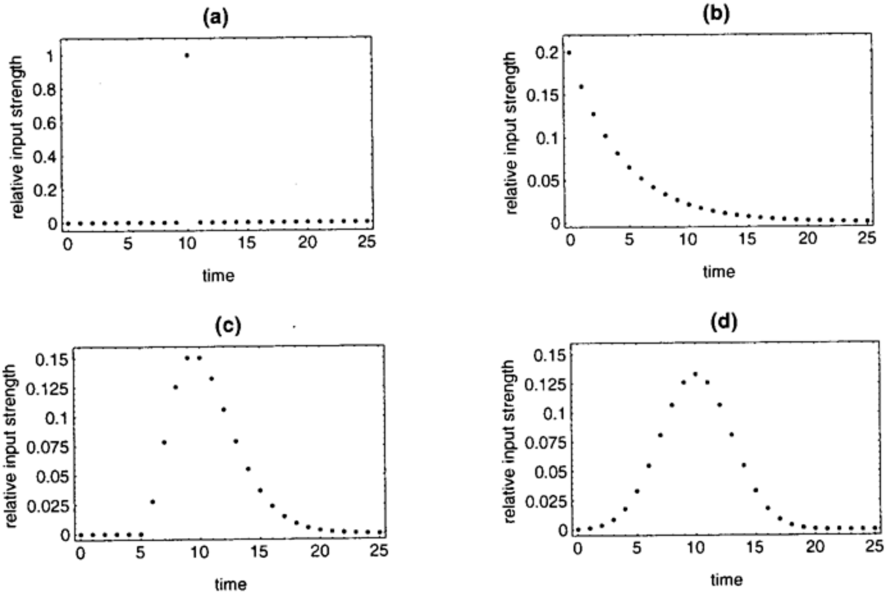

FIGURE 3 The kernel functions for (a) a delay-line memory, $\omega = 10$; (b) an expontial trace memory, $\mu = .8$; (c) a gamma memory, $\omega = 6$; and $\mu = .4$, and (d) a Gaussian memory.

EXPONENTIAL TRACE MEMORY

An exponential trace memory is formed using the kernel function

$$c_i(t) = (1 - \mu_i)\mu_i^t,$$

where μ_i lies in the interval $[-1, 1]$ (Figure 3(b)). This form of memory has been studied by Jordan (1987), Mozer (1989), and Stornetta, Hogg, and Huberman (1988). Unlike the delay-line memory, the exponential trace memory does not sharply drop off at a fixed point in time; rather, the strength of an input decays exponentially. This means that more recent inputs will always have greater strength than more distant inputs. Nonetheless, with no noise, an exponential memory can preserve all information in a sequence. For instance, consider a sequence of binary digits, i.e., $x_i(t) \in \{0, 1\}$ and a memory with $\mu = .5$. The memory becomes a bit string in which the kernel function assigns the input at time $t - \tau$ to bit position $2^{-(\tau+1)}$ of the string. If the memory has finite resolution or is noisy, the least significant bits—the most distant inputs—will be lost. Even with no information loss, however, it is difficult for a neural network to extract and use all the information contained in the memory.

An important property of exponential trace memories is that the $\bar{\mathbf{x}}_i$ can be computed incrementally:

$$\bar{\mathbf{x}}_i(t) = (1 - \mu_i)\mathbf{x}_i(t) + \mu_i \bar{\mathbf{x}}_i(t-1)$$

with the boundary condition $\bar{\mathbf{x}}_i(0) = 0$. As with the delay-line memory, the exponential trace memory consists of Ω state vectors, $(\bar{\mathbf{x}}_1(t), \bar{\mathbf{x}}_2(t), \bar{\mathbf{x}}_3(t), \ldots, \bar{\mathbf{x}}_\Omega(t))$. One can view exponential trace memories as maintaining moving averages (exponentially weighted) of past inputs. The μ_i allow for the representation of averages spanning various intervals of time. The exponential trace memory is a special case of traditional statistical moving-average models, the MA(1) model. The more general case of MA(q) cannot readily be represented using a trace of the input sequence alone and is dealt with below.

GAMMA MEMORY

Two dimensions, *depth* and *resolution* (de Vries & Principe, 1991, 1992), can be used to characterize the delay-line and exponential-trace memories. Roughly, "depth" refers to how far into the past the memory stores information relative to the memory size, quantified perhaps by the ratio of the first moment of the kernel function to the number of memory state variables. A low-depth memory only holds recent information; a high-depth memory holds information distant in the past. "Resolution" refers to the degree to which information concerning the individual elements of the input sequence is preserved. A high-resolution memory can reconstruct the actual elements of the input sequence; a low-resolution memory holds coarser information about the sequence. In terms of these dimensions, the delay-line memory is low depth but high resolution, whereas the exponential-trace memory is high depth but low resolution.

de Vries and Principe (1991, 1992) have proposed a memory model that generalizes across delay lines and exponential traces, allowing a continuum of memory forms ranging from high resolution and low depth to low resolution and high depth. In continuous time, the kernel for this memory is a gamma density function, hence the name *gamma memory*. Although de Vries and Principe deal primarily with continuous time dynamics, there is a discrete equivalent of the gamma density function, the negative binomial, which can serve as the corresponding kernel for discrete-time gamma memories:

$$c_i(t) = \begin{cases} \begin{pmatrix} t \\ \omega_i \end{pmatrix} (1-\mu_i)^{\omega_i+1} \mu_i^{t-\omega_i} & \text{if } t \geq \omega_i; \\ 0 & \text{if } t < \omega_i \,. \end{cases}$$

As before, ω_i is an integer delay and μ_i lies in the interval $[0, 1]$. Figure 3(c) shows a representative gamma kernel. With $\omega_i = 0$, the gamma kernel reduces to an exponential trace kernel. In the limit as $\mu_i \rightarrow 0$, it reduces to a delay-line kernel.

As with the exponential trace kernel, the convolution of the gamma kernel with the input sequence can be computed incrementally (de Vries & Principe, 1991, 1992). However, to compute $\bar{\mathbf{x}}$ for a given μ and ω, denoted here as $\bar{\mathbf{x}}_{\mu,\omega}$, it is also necessary to compute $\bar{\mathbf{x}}_{\mu,j}$ for $j = 0 \ldots \omega - 1$. The recursive update equation is

$$\bar{\mathbf{x}}_{\mu,j}(t) = (1-\mu)\bar{\mathbf{x}}_{\mu,j-1}(t-1) + \mu\bar{\mathbf{x}}_{\mu,j}(t-1)$$

with boundary conditions

$$\bar{\mathbf{x}}_{\mu,-1}(t) = \mathbf{x}(t+1) \text{ for all } t \geq 0, \text{ and}$$
$$\bar{\mathbf{x}}_{\mu,j}(0) = 0 \text{ for all } j \geq 0.$$

In constructing a gamma memory, the designer must specify the largest ω, denoted Ω. The total number of state vectors is then $\Omega + 1$: $\bar{\mathbf{x}}_{\mu,0} \ldots \bar{\mathbf{x}}_{\mu,\Omega}$. Given Ω, the tradeoff between resolution and depth is achieved by varying μ. A large μ yields a high-depth, low-resolution memory; a small μ yields low-depth, high-resolution memory.

Each set of Ω+1 state vectors with a given μ forms a *gamma family*. The gamma memory can consist of many such families, with family i characterized by μ_i and Ω_i. Of course, this leads to a large and potentially unwieldy state representation. Based on assumptions about the domain, one can trim down the state representation, selecting only certain vectors within a family for input to the neural net predictor, e.g., the subset $\{\bar{\mathbf{x}}_{\mu_i,\Omega_i}\}$. One can view this as a direct extension of the exponential trace memory, which is of this form with $\Omega_i = 0$ for all i. The nonselected vectors must be maintained as part of the recursive update algorithm but need not be passed to the neural net predictor.

OTHER FORMS OF MEMORY

The three memories just described are not the only possibilities. Any kernel function results in a distinct memory form. For instance, a Gaussian kernel (Figure 3(d)) can be used to obtain a symmetric memory around a given point in time. What makes the gamma memory and its special cases particularly useful is that they can be computed by an incremental update procedure, whereas forms such as the Gaussian require evaluating the convolution of the kernel with the input sequence at each time step. Such convolutions, while occasionally used (Bodenhausen & Waibel, 1991; Tank & Hopfield, 1987; and Unnikrishnan, Hopfield, & Tank, 1991), are not terribly practical because of the computational and storage requirements.

Radford Neal (personal communication, 1992) has suggested a class of kernels that are polynomial functions over a fixed interval of time beginning at a fixed point in the past; i.e.,

$$c_i(t) = \begin{cases} \sum_{j=0}^{\Omega} \mu_{ij} t^j, & \text{if } t_i^- \leq t \leq t_i^+; \\ 0, & \text{otherwise.} \end{cases}$$

He has proposed an incremental update procedure for memories based on this kernel. The number of memory state vectors, Ω, is the order of the polynomial; the update time is independent of the interval width $t_i^+ - t_i^-$. This appears to be a promising but as yet unexplored alternative to the gamma memory.

For any kernel function, c, which can be expressed as a weighted sum of gamma kernels,

$$c(t) = \sum_{j=0}^{\Omega} q_j c_{\mu,j}(t) . \tag{1}$$

Where q_j are the weighting coefficients, the resulting memory can be expressed in terms of the gamma memory state vectors:

$$\bar{\mathbf{x}}(t) = \sum_{j=0}^{\Omega} q_j \bar{\mathbf{x}}_{\mu,j} .$$

de Vries and Principe (1991) show that the gamma kernels form a basis set, meaning that, for an arbitrary kernel function c that decays exponentially to 0 as $\tau \to \infty$, there exists a value of Ω and a set of coefficients $\{q_j\}$ such that Eq. (1) holds. This result is easy to intuit for the case of delay kernels, but one can see from this simple case that the required computation amounts essentially to convolving a complex kernel with the input sequence at each time step and, consequently, is not of great practical utility.

CONTENTS OF SHORT-TERM MEMORY

Having described the *form* of the memory, I now turn to the *content*. Although the memory must hold information pertaining to the input sequence, it does not have to be a memory of the raw input sequence, as was assumed previously. The memory encoding process can include an additional step in which the input sequence, $\mathbf{x}(1) \ldots \mathbf{x}(t)$, is transformed into a new representation $\mathbf{x}'(1) \ldots \mathbf{x}'(t)$, and it is this transformed representation that is encoded in the memory. Thus, $\bar{\mathbf{x}}_i$ are defined in terms of $\mathbf{x}'$, not $\mathbf{x}$.

The case considered in previous sections is an identity transformation,

$$\mathbf{x}'(\tau) = \mathbf{x}(\tau).$$

Memories utilizing this transformation will be called *input* or *I* memories. Models in the neural net literature make use of three other classes of transformation. First, there is a transformation by a nonlinear vector function f,

$$\mathbf{x}'(\tau) = f(\mathbf{x}(\tau)).$$

This transformation results in what I will call a *transformed input* or *TI* memory. Generally f is the standard neural net activation function, which computes a weighted sum of the input elements and passes it through a sigmoidal nonlinearity:

$$f_{\mathbf{w}}(\mathbf{v}) = \frac{1}{1+\mathbf{e}^{-\mathbf{w}\cdot\mathbf{v}}}. \tag{2}$$

Second, the nonlinear transformation can be performed over not only the current input but also the current internal memory state:

$$\mathbf{x}'(\tau) = f(\mathbf{x}(\tau), \bar{\mathbf{x}}_1(\tau), \ldots, \bar{\mathbf{x}}_\Omega(\tau)). \tag{3}$$

This leads to a *transformed input and state* or *TIS* memory. Such memories can be implemented in a recurrent neural net architecture in which $\bar{\mathbf{x}}_i$ and $\mathbf{x}'$ correspond to activities in two layers of hidden units (Figure 4(a)). Note that this architecture is quite similar to a fairly common, recurrent, neural net sequence-processing architecture (Figure 4(b); e.g., Elman, 1990; Mozer, 1989), which I'll refer to as the *standard* architecture.

Third, for *autopredictive* tasks in which the target output, $\mathbf{p}(\tau)$, is a one-step prediction of the input; i.e.,

$$\mathbf{p}(\tau) = \mathbf{x}(\tau+1),$$

one can consider an alternative content to the memory. Rather than holding the actual sequence value, the preceding prediction can be used instead:

$$\mathbf{x}'(\tau) = \mathbf{p}(\tau-1).$$

This transformation will be called an *output* or *O* memory. Of course, *TO* and *TOS* memories can be constructed by analogy to the TI and TIS memories, suggesting a characterization of memory content along two subdimensions, one being the transformation applied and the other being whether the transformation is applied to the input or previous output. Since it does not make a great deal of sense to ignore input sequence information when it is available, I will include in the category of output memories those that combine input and output information, e.g., by taking a difference between input and output, $\mathbf{x}(\tau+1) - \mathbf{p}(\tau)$, or by using the input information when it is available, such as during training, and the output information otherwise.

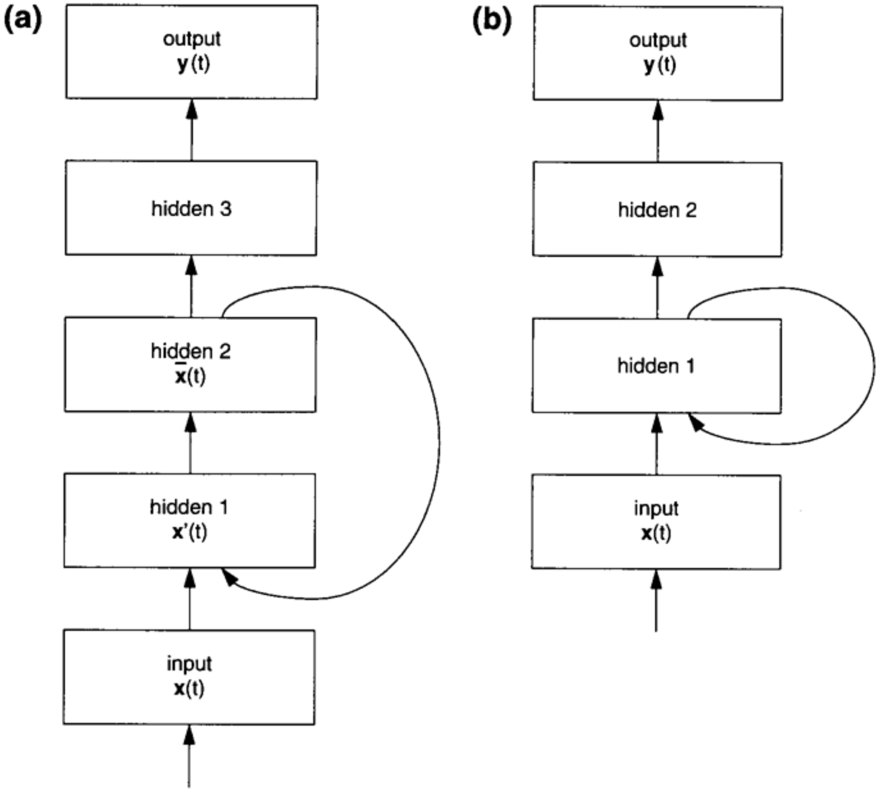

FIGURE 4 (a) A TIS memory in a neural net architecture. Each rectangle represents a layer of processing units, and each arrow represents complete connectivity from one layer to another. The input layer corresponds to the current element of the time series, $\mathbf{x}(t)$, the first hidden layer to the TIS representation, $\mathbf{x}'(t)$, and the second hidden layer to the memory state, $\{\bar{\mathbf{x}}_i(t)\}$. (b) A standard recurrent neural net architecture, which is like the TIS architecture except that the first two hidden layers are collapsed together; i.e., $\mathbf{x}'(t) = [\bar{\mathbf{x}}_1(t) \ldots \bar{\mathbf{x}}_\Omega(t)]$.

MEMORY ADAPTABILITY

The final dimension that characterizes short-term memories is the *adaptability* of the memory. A memory has various parameters—$\{\mu_i\}$, $\{\Omega_i\}$, and the $\mathbf{w}$ of Eqs. (2) and (3)—that must be specified. If the parameters are fixed in advance, one can call the memory *static* because the memory state, $\{\bar{\mathbf{x}}_i(t)\}$, is a predetermined function of the input sequence, $\mathbf{x}(1) \ldots \mathbf{x}(t)$. The task of the neural net is make the best predictions possible given the fixed representation of the input history. In contrast, if the neural net can adjust memory parameters, the memory representation is *adaptive*. Essentially, by adjusting memory parameters, the neural net selects, within limits on the capacity of the memory, what aspects of the input sequence are available for making predictions. In addition to learning to make predictions, the neural net must also learn the characteristics of the memory—as specified by its parameters—that best facilitate the prediction task.

The memory model in Figure 2 is an example of a static memory. The model sketched in Figure 4 is intrinsically an adaptive memory because without training the hidden unit responses, the network seems unlikely to preserve sufficient relevant information.

Interesting cases of adaptive memory models in the neural net literature include: the learning of delays (Bodenhausen & Waibel, 1991; and Unnikrishnan, Hopfield, & Tank, 1991), the learning of exponential decay rates (Bachrach, 1988; Frasconi, Gori, & Soda, 1992; and Mozer, 1989), and the learning of gamma memory parameters (de Vries & Principe, 1992). Every sequence-processing, recurrent, neural net architecture trained with *backpropagation through time* (or *BPTT*; Rumelhart, Hinton, & Williams, 1986) or *real-time recurrent learning* (or *RTRL*; Robinson & Fallside, 1987; Schmidhuber, 1992; and Williams & Zipser, 1989) is also an adaptive memory model in that the training adjusts the $\mathbf{w}$ of Eq. (2). The Elman SRN algorithmSRN (Elman, 1990) lies somewhere between an adaptive and static memory because the training procedure—which amounts to backpropagation one step in time—is not as powerful as full-blown backpropagation through time or RTRL.

Static memory models can be a reasonable approach if there is adequate domain knowledge to constrain the type of information that should be preserved in the memory. For example, Tank and Hopfield (1987) argue for a memory that has high resolution for recent events and decreasing resolution for more distant events. The argument is based on a statistical model of temporal distortions in the input. If there is local temporal uncertainty in the occurrence of an input event, then the uncertainty relative to the present time increases with the expected temporal lag of the event. The decreasing-resolution-with-increasing-depth memory suggested by this argument can be achieved by a gamma family, $\{\bar{\mathbf{x}}_{\mu,\omega} : 0 \leq \omega \leq \Omega\}$. As ω increases, the state vector $\bar{\mathbf{x}}_{\mu,\omega}$ represents an increasing-depth, decreasing-resolution trace of the input.

MEMORY TAXONOMY

I have described three dimensions along which neural net memory models vary: memory form (delay line, exponential trace, gamma trace), content (I, TI, TIS, O, TO, TOS), and adaptability (static, adaptive). The Cartesian product of these dimensions yields 36 distinct classes.

Table 1 presents some combinations of memory form and content, along with existing models in the literature of each class.[1] The I-delay memory is the simplest class and corresponds to a feedforward network with a delay space embedding of the input sequence. TI-delay memories are the basis of the TDNN (Waibel et al., 1989) and the FIR neural net (Wan, this volume). In these models, each hidden unit—which is a nonlinear transformation of the input—maintains a history of its n most recent values, and all these values are available to the next layer. TI-delay models have also been studied extensively within the more physically oriented neural net community (Kleinfeld, 1986; Sompolinksy & Kanter, 1986; and for an overview, see Kühn & van Hemmen, 1991). Herz (1991) proposes a TIS-delay memory in a network whose dynamics are governed by a Lyapunov function under certain symmetry conditions on the time-delayed weights. Connor, Atlas, and Martin (1992) study a nonlinear, neural network ARMA model, whose MA component is constructed from an O-delay memory that retains the difference between the predicted and actual outputs (the latter being identical to the next input). In general, nonlinear MA(q) models can be based on O-delay memories.

Moving to the second column in Table 1, Bachrach (1988), Frasconi, Gori, and Soda (1992), and Mozer (1989) have proposed an TI-exponential memory model and described a computationally efficient algorithm for adapting the μ_i parameters. Mozer's (1992) multiscale integration model is a TIS-exponential memory. The motivation underlying this work was to design recurrent hidden units that had different time constants of integration, the slow integrators forming a coarse but global sequence memory and the fast integrators forming a fine-grained but local memory. This memory was adaptive in that the hidden-unit response functions were learned but static in that the μ_i were fixed. Jordan (1987) has incorporated an O-exponential memory in a sequence production network. During training, the network is given target output values, and these are fed back into the output memory; during testing, however, the actual outputs are used instead.

As the third column of the table suggests, the gamma memory is the least studied form. de Vries and Principe's initial simulations have been confined to an I-gamma memory, although they acknowledge other possibilities.

Beyond the simple memory classes, hybrid schemes also can be considered. The TDNN, for example, is an I-delay memory attached to a series of TI-delay memory modules. I have proposed architectures combining an I-delay memory with a TIS-exponential memory (Mozer & Soukup, 1991) and an I-delay memory with a TI-exponential memory (Mozer, 1989). The possibilities are, of course, diverse, but the certain conclusion is that the best class of memory depends on the domain and the task.

[1] The TO and TOS varieties have been omitted because they have received little study.

TABLE 1 Taxonomy of neural net architectures for temporal processing

	memory form		
memory contents	delay	exponential	gamma
I	Elman & Zipser, 1988; Lapedes & Farber, 1987; Zhang & Hutchinson, this volume	?	de Vries & Principe, 1991, 1992
TI	Waibel et al., 1989; Wan, this volume; Kleinfeld, 1986; Sompolinsky & Kanter, 1986	Bachrach, 1988; Frasconi, Gori, & Soda, 1992; Mozer, 1989	?
TIS	Herz, 1991	Mozer, 1992	?
O	Connor, Atlas, & Martin, 1992	Jordan, 1987	?

While this taxonomy represents a reasonable first cut at analyzing the space of memory models, it is not comprehensive and does not address all issues in the design of a memory model. Although some models in the literature can be pigeonholed into a certain cell in the taxonomy, they have critical properties that are ignored by the taxonomy. For example, Schmidhuber (1992; Schmidhuber, Prelinger, & Mozer, in preparation) has proposed a type of I-delay memory in which the memory size and the ω_i delay parameters are dynamically determined based on the input sequence as it is processed. The idea underlying this approach is to discard redundant input elements in a sequence. (See Myers, 1990, and Ring, 1991, for variants of this idea.)

For the most part, I also have sidestepped the issue of network dynamics, assuming instead the typical backpropagation-style activation dynamics. There is an interesting literature on temporal associative networks that seek energy minima (e.g., Herz, 1991; Kleinfeld, 1986; and Sompolinsky & Kanter, 1986) and networks that operate with continuous time dynamics (Pearlmutter, 1989). However, this work more directly addresses the issue of network architecture, not representation, and as I indicated at the outset, I have focused on representational issues.

THE INADEQUACY OF STANDARD, RECURRENT, NEURAL NET ARCHITECTURES

The standard, recurrent, neural net architecture for sequence-processing tasks in Figure 4(b) amounts to a trivial case of a TIS-exponential memory with $\mu_i = 0$ or, equivalently, a TIS-delay memory with $\Omega = 0$. I will refer to this limiting case as a *TIS-0* memory. Although I and others have invested many years exploring TIS-0 architectures, the growing consensus seems to be that the architecture is inadequate for difficult temporal-processing and prediction tasks.

The TIS-0 architecture seems sufficiently powerful *in principle* to handle arbitrary tasks. One is tempted to argue that, with enough hidden units and training, the architecture should be capable of forming a memory that retains the necessary task-relevant information. This is achieved by adjusting the $\mathbf{w}$ parameters in Eqs. (2) and (3) using a gradient-descent procedure. *In practice*, however, many have found that gradient descent is not sufficiently powerful to discover the sort of relationships that exist in temporal sequences, especially those that span long temporal intervals and that involve extremely high order statistics. Bengio, Frasconi, and Simard (1993) present theoretical arguments for inherent limitations of learning in recurrent networks. Mozer (1989, 1992) illustrates with examples of relatively simple tasks that cannot be solved by TIS-0 memories. Although TIS-0 memories were inadequate for these tasks, TI-exponential or TIS-exponential memories yielded significantly improved performance. In both cases, the exponential trace components allowed the networks to bridge larger intervals of time. As further evidence, consider the active neural net speech-recognition literature. I know of no serious attempts at speech recognition using TIS memories. However, the TDNN architecture—a TI-delay memory—is viewed as quite successful. There have even been interesting explorations using TI-exponential memories (Bengio, De Mori, & Cardin, 1990; and Watrous & Shastri, 1987).

With TIS-0 architectures, gradient descent often cannot find meaningful values for the recurrent-connection strengths. Gradients computed by backpropagation, using either BPTT or RTRL, tend to be quite small.[2] For an intuition as to why this is so, consider the network "unfolded" in time, which is necessary for computing the gradients with BPTT. In the unfolded network, there is a copy of the original network (units and connections) for each time step of the temporal sequence. This turns the recurrent network into a deeply layered feedforward network. Now consider the consequence of changing a weight embedded deep in the unfolded network, say, a weight connecting a hidden unit at time 1 to a hidden unit to time 2. Although it will have some consequence at a later time t, the effect is likely to be small, hence so will the gradient. Further, the impact of a connection weight—even

[2] With careful selection of the weights, one can prevent the gradients from shrinking through time (Hochreiter, 1991); however, this imposes serious restrictions on the sorts of memories the network can form.

if appropriate—will be masked by other weights if their values are inappropriate. This situation is true for feedforward nets as well as recurrent nets, but the problem is exacerbated by the deeply layered structure of the unfolded net. The result is an error surface fraught with local optima.

While this situation may hold for TIS-0 memories, it may not be the case for other recurrent architectures that have a more specialized connectivity, e.g., TI-exponential memories. Further, used *in conjunction with* other types of memory that may help bootstrap learning, TIS-0 memories are a promising possibility. It also seems well worth investigating classes of recurrent network memory in the taxonomy that have yet to be explored. To the best of my knowledge, these classes include TIS-delay, TI-gamma, TIS-gamma, nonlinear MA models of the O-delay variety, as well as MA models based on O-gamma and O-exponential and the TO and TOS varieties.

EXPERIMENTS WITH A FINANCIAL DATA SERIES

In the remainder of this chapter, I present experiments examining a prediction task from financial data series using three memory types: I-delay, TIS-0, and a hybrid approach combining I-delay and TIS-0. Although these experiments do not explore the more sophisticated memory classes described earlier, they provide a baseline against which future experiments can be compared.

The data series that I studied was Data Set C of the Santa Fe Time Series Prediction and Analysis Competition, the tickwise bids for the exchange rate from Swiss francs to U.S. dollars, recorded by a currency trading group from August 1990 to April 1991. "Tickwise" means that the samples come at irregular intervals of time, as demanded by the traders. Each sample is thus indexed by a time of day. There were six prediction tasks in the Competition, but I focused on three: predicting the value of the series 1, 15, and 60 minutes in the future.

The data set appears challenging because the average sampling rate is high relative to the rate of trends in the data. Although the nature of the data is that it is obviously noisy, smoothing is inappropriate because there may be information in the high frequency changes. Consequently, a high-depth memory would seem necessary. This data set seems ideal for a TIS architecture which, in principle, does not have a fixed memory depth.

The training data consisted of ten contiguous spans of time with unspecified gaps between them, in total 114 days of data. I treated each day's data as an independent sequence, i.e., I assumed no useful information was carried over from one day to the next. For each day, I massaged the data to transform tickwise samples to fixed interval samples, one minute apart, operating on the assumption that the value of the series remained constant from the time of a sample until the time of the next sample. This sequence was subsampled further, at an interval of Δ_s

minutes. For the 1-, 15-, and 60-minute prediction tasks, I used $\Delta_s = 2$, 5, and 10, respectively. The average trading day spanned 575 minutes, resulting in daily series of lengths 57.5 to 287.5. Of the 114 days of training data, 6 complete days—chosen at random—were set aside to be used for validation testing. It was on the basis of this validation data that many parameter values, such as the Δ_s, were selected.

The input to the neural net predictor consisted of three values: the day of the week, the time of day, and the change in the series value from one sample to the next, $s(\Delta_s t) - s(\Delta_s(t-1))$, where $t = 1 ... T$ is an index over the input sequence and $s(\tau)$ denotes the sample at minute τ from the first sample of a day. The target output of the neural net is a prediction of the change in the series value Δ_p minutes in the future, i.e., $s(\Delta_s t + \Delta_p) - s(\Delta_s t)$.

Each of the input and output quantities is represented using a *variable encoding* (Ballard, 1986), which means that the input and output activities are monotonic in the represented quantity. This is the obvious encoding; I mention it simply because connectionist approaches to representation (e.g., Ballard, 1986; and Smolensky, 1990) offer a range of alternatives that could be fruitfully explored, but I have not done so in the present work.

The activity of each input unit was normalized to have a mean of 0.0 and variance 1.0 over the training set. This leads to a better-conditioned search space for learning (le Cun, Kanter, & Solla, 1991; and Widrow & Stearns, 1985). The target output activity was normalized to have mean 0.0 and variance 100.0 over the training set. Despite this large variance, the observed variance in the unit was much closer to 1.0 because strong predictions could not be made due to the uncertainty in the data.

The general architecture studied, shown in Figure 5, includes two memories—types TIS-0 and I-delay—which feed, along with the inputs directly, into a layer of hidden units, and then map to an output prediction. A symmetric activation function (activation in range −1 to +1) was used for the hidden units, including the hidden units of the TIS-0 memory. (Recall that the TIS-0 memory is implemented as a recurrent network with internal hidden units; see Figure 4(b).) The output unit had a linear activation function.

The actual architectures tested were three special cases of this general architecture. The first case eliminated the TIS-0 memory and used a six-step I-delay memory. The number of hidden units was varied. When the number of hidden units is 0, this architecture reduces to a linear autoregressive model. The second case eliminated the I-delay memory in favor of the TIS-0 memory. The TIS-0 memory consisted of 15 internal hidden units, plus 5 hidden units in the penultimate layer. The third case—a hybrid architecture—included both the TIS-0 and I-delay memories, with parameters as above. The choice of number of hidden units and number of delays was based on validation testing, as described below.

The various networks were trained using stochastic gradient descent, whereby the weights were updated following presentation of an entire day's data. Backpropagation through time (Rumelhart, Hinton, & Williams, 1986) was used to train

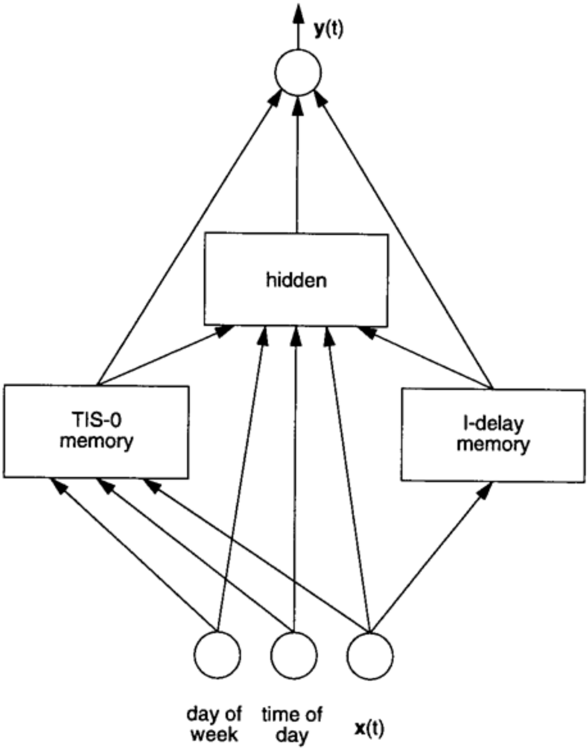

FIGURE 5 General architecture used for time series experiments. The figure depicts three inputs feeding into two types of memory which then map, through a layer of hidden units, to a single output—the prediction unit. Arrows represent connections between units or sets of units. The architecture includes direct connections from "day of week" and "time of day" to the output, which are not depicted.

the recurrent connections in the TIS-0 memory. Sufficiently small learning rates were chosen such that the training error decreased reliably over time; generally, I set the learning rate to .0002 initially and decreased it gradually to .000005 in the first 500 passes through the training data. The networks were initialized with weights chosen randomly from a normal distribution, mean zero, and were normalized such that the sum of the absolute values of the weights feeding into a unit totaled 2.0.

Following each pass through the training data, the errors in the training and validation sets were computed. Rather than training the network until the training set error converged, as is commonly done, training was stopped when the validation error began to rise, a method suggested by Weigend, Rumelhart, and Huberman (1990) and Geman, Bienenstock, and Doursat (1992). In actuality, this involved training the network until convergence, observing the point at which the validation error began to rise, and then restoring the saved network weights at this critical point. At this point, ten final passes were made through the training data, annealing the learning rate to 0. The purpose of this step was to fine tune the weights, removing the last bit of jitter.

I explored many variations of the architecture and training procedure, including: varying the number of hidden units, varying the number of delays in the I-delay memory, trying different values for the Δ_s, removing the day of week and time of day inputs, and generating a larger training set by transforming each day's data into Δ_s different streams, each offset from the next by one minute. Generalization performance was estimated for each variation using the validation set.[3] The parameters and choices as previously described yielded performance as good as any.

For each architecture studied, 25 replications were performed with different random initial weights. The ten models that yielded the best validation performance were selected to make predictions for the competition data, which were averaged together to form the final predictions. The purpose of this procedure was to obtain predictions that were less sensitive to the initial conditions of the networks. Similar procedures have been suggested by Lincoln and Skrzypek (1990) and Rosenblatt (1962).

COMPETITION TEST DATA

I did not submit an official entry to the Competition. However, shortly after the close of the competition deadline, I began experimenting with Data Set C, with no knowledge of the series beyond what was available to competition entrants. I used the TIS-0 version of the architecture to obtain 15- and 60-minute predictions for the competition data. Performance on this series is measured in units of *normalized mean squared error* or NMSE. The normalization involved is to divide the mean squared error by the error that would have been obtained assuming that the future value of the series is the same as the last observed value. This normalization term is the least-squares prediction assuming a random walk model for the series. NMSE values less than 1.0 thus indicate that structure has been extracted from the series. After submitting my predictions to Andreas Weigend, he reported back that I'd done a bit better than entrants to the Competition: the 15-minute prediction yielded an NMSE of .859 and the 60-minute prediction an NMSE of .964.

At the Time Series Workshop, Xiru Zhang quite validly argued that the competition data set simply did not contain sufficient test points to ascertain reliability of the predictions. We agreed to perform additional experiments using larger test sets.

[3] Because the validation set was also used to determine when to stop training, a bias is introduced in using the validation set for selecting the architecture. In retrospect, two distinct validation sets should have been used.

EXTENDED COMPETITION TEST SET

The first set of experiments utilized the competition training set and, hence, the models developed for the Competition, but test data came from the gaps in the training set. Recall that the training set consisted of ten contiguous time spans with gaps in between. The gaps covered 69 days, more than sufficient for an extended test set. All data in the gaps was used for testing, with the exception of the first hour of each day, in order to provide an initialization period for the recurrent TIS-0 architecture.

The results for 1-, 15-, and 60-minute predictions are shown in Table 2. The I-delay architecture was tested with different numbers of hidden units and the TIS-0 model in the configuration described earlier. The I-delay architecture with zero hidden units is equivalent to a linear autoregressive model, AR(6). To a first order, all architectures performed the same, slightly better than a random walk model and also better than Zhang and Hutchinson (this volume). I attempted to determine the reliability of differences in the NMSEs but did not get very far. A straight t-test on the squared errors for the individual predictions yielded no significant differences. Noting that the data distribution strongly violated the assumption of normality, I applied a log transform to the squared errors, which still did not achieve a normal distribution but came a bit closer. After the log transform, reliable differences between NMSEs were obtained for pairs of conditions whose NMSEs differed by roughly .001 or more. No matter, performance was discouragingly poor in all conditions.

Finnoff, Hergert, and Zimmermann (1992) report performance comparisons across a broad spectrum of tasks for a variety of different training techniques. While their explorations included variations on the techniques I used, they found that a relatively simple technique—stopping training based on a validation set and then training further using weight decay—appeared to work very well across tasks.

TABLE 2 NMSE for extended competition test set

architecture	1-minute prediction (18,465 data points)	15-minute prediction (7,246 data points)	60-minute prediction (3,334 data points)
I-delay, 0 hidden	.998	1.001	.999
I-delay, 1 hidden	.998	1.001	.996
I-delay, 3 hidden	.998	1.000	.998
I-delay, 5 hidden	.998	1.000	.998
I-delay, 10 hidden	.998	.999	.999
TIS-0	.998	.999	.997

TABLE 3 NMSE for 1985–87 test data

architecture	1-minute prediction (57,773 data points)
I-delay, 0 hidden	.999
I-delay, 5 hidden	.985
I-delay, 10 hidden	.985
I-delay, 20 hidden	.985
TIS-0	.986
hybrid TIS-0 and I-delay	.986

I experimented with this method, but obtained results essentially the same as those reported in Table 2. They also used a much larger validation set. Being concerned that my small validation set was responsible for performance, I ran further experiments in which the validation corpus was increased to 25% of the training set, but this manipulation also had little effect on performance.

THE 1985–1987 CORPUS

The extended competition test set was drawn from the same time period as the training data. Xiru Zhang suggested an additional data set in which the testing corpus came from a different time period than the testing corpus. He proposed training data from the same exchange rate series spanning the time period 6/85–5/86 and testing data spanning the time period 6/86–5/87. This encompassed 248 days of training data and a comparable amount of testing data. I selected 37 days of the training data—roughly 15%—for validation testing. Following Zhang and Hutchinson, I explored only the one-minute prediction task. The first hour of each day's data was excluded from testing, for reasons indicated earlier.

The results are summarized in Table 3 for a variety of architectures. Here, the I-delay architecture with 0 hidden units—equivalent to a linear AR(6) model—performed significantly worse than the other architectures, but there was no reliable difference among the other architectures.

Zhang and Hutchinson report an NMSE of .964 on the same data set, better than the .985 I obtained. However, it is important to emphasize that the two performance scores are *not* comparable. Zhang and Hutchinson threw out cases—from both training and test sets—for which a tickwise series value was not available 60 ± 30 seconds from the time at which a prediction was made. Hence, their predictions were *conditional on assumptions about the time of occurrence of future ticks.* At the point at which a prediction is made, they cannot possibly know whether the assumptions are valid. Therefore, their approach cannot be used as a predictive model without a secondary component that predicts whether their prediction criteria will be satisfied. Alternatively, they could retrain and retest their network on all data, not just the data satisfying their criterion.[4]

Making such a conditional prediction is likely a simpler task than making an unconditional prediction, because it ensures some homogeneity among the cases. In effect, it throws out a large number of the "no change" predictions; in my test set, the fraction of "no change" future values was 71% versus only 41% in Zhang and Hutchinson's set.

A perhaps more interesting measure of performance is whether the network can correctly predict the change in direction of the series: down, no change, or up. To explore this measure, it is necessary to quantize the network outputs, i.e., to specify a criterion, call it c^+, such that all predictions greater than c^+ are classified as "up," and a second criterion, c^-, such that all predictions less than c^- are classified as "down." Predictions between c^- and c^+ are classified as "no change." The principled approach to selecting c^- and c^+ involves selecting the values that yield the best classification performance on the training set; this involves an exhaustive search through all possible values. Instead, I used a simple technique to determine c^+: c^+ was chosen such that the fraction of training set predictions greater than c^+ equaled the fraction of training cases whose target prediction was "up," and likewise for c^-. Using these values of c^- and c^+ on the test set, the 10-hidden I-delay network yielded a performance of 63.2% correct predictions. This sounds pretty good, but in reality is quite poor because 71% of all cases were "no change" and the net could have obtained this performance level simply by always responding "no change." The network has a much greater ability to predict the direction of change *conditional upon a change having occurred*: It correctly predicts the direction on 58.5% of test cases, whereas always predicting the most frequent category in the training set obtains a performance of only 49.9%. If one is seriously interested in predicting direction of change, it seems sensible to train a network with three output units, one for each mutually exclusive prediction, perhaps using a normalized exponential output function (Bridle, 1990; Rumelhart et al., 1993) to obtain a probability distribution over the alternative responses.

CONCLUSIONS

In this work, I have begun an exploration of alternative architectures for temporal sequence processing. In the difficult time series prediction problem studied, a nonlinear extension of autoregressive models—the basic I-delay architecture—fared no worse than a more complex architecture, the hybrid I-delay and TIS-0. However, it is impossible to determine at present whether the prediction limitations I found

[4] In defense of Zhang and Hutchinson's decision, they believed that the competition set-up implied there would be a value near the prediction time, and thus it was to their advantage to make use of this information.

are due to not having explored the right architecture, the right preprocessing or encoding of the input sequence, the right training method, or are simply due to the absence of further structure in the data that can be extracted from the series alone without the aid of other financial indicators. The studies of Zhang and Hutchinson (this volume) of the same data series using an I-delay memory but with a different input/output encoding and training method yielded comparable results, supporting the hypothesis that the encoding and method are not responsible for the poor predictability. At very least, my results can serve as a baseline against which researchers can compare their more sophisticated architectures, representations, and training method. I look forward to reports showing improvements over my work.

My aim in presenting a taxonomy of architectures is to suggest other possibilities in the space to be explored. I don't claim that the taxonomy is all encompassing or that it is necessarily the best way of dividing up the space of possibilities. However, it seems like a reasonable first attempt in characterizing a set of models for temporal processing commonly used in the field.

ACKNOWLEDGMENTS

My thanks to Andreas Weigend and Neil Gershenfeld for organizing the Santa Fe workshop, and in particular to Andreas for his abundant feedback at all stages of the work and suggestions for extensions. Jim Hutchinson, Radford Neal, Jürgen Schmidhuber, and Fu-Sheng Tsung, and an anonymous reviewer provided very helpful comments on an earlier draft of the chapter. This research was supported by NSF Presidential Young Investigator award IRI–9058450, grant 90–21 from the James S. McDonnell Foundation, and DEC external research grant 1250.