Learning long–term dependencies is not as difficult with NARX recurrent neural networks

Tsungnan Lin [1,2], Bill G. Horne [1], Peter Tiňo [1,3] and C. Lee Giles [1,4]

[1] NEC Research Institute, 4 Independence Way, Princeton, NJ 08540
[2] Department of Electrical Engineering, Princeton University, Princeton, NJ 08540
[3] Dept. of Computer Science and Engineering, Slovak Technical University, Ilkovicova 3, 812 19 Bratislava, Slovakia
[4] UMIACS, University of Maryland, College Park, MD 20742

Abstract

It has recently been shown that *gradient descent* learning algorithms for recurrent neural networks can perform poorly on tasks that involve *long–term dependencies*, i.e. those problems for which the desired output depends on inputs presented at times far in the past.

In this paper we explore the long–term dependencies problem for a class of architectures called NARX recurrent neural networks, which have powerful representational capabilities. We have previously reported that gradient descent learning is more effective in NARX networks than in recurrent neural network architectures that have "hidden states" on problems including grammatical inference and nonlinear system identification. Typically, the network converges much faster and generalizes better than other networks. The results in this paper are an attempt to explain this phenomenon.

We present some experimental results which show that NARX networks can often retain information for *two to three times* as long as conventional recurrent neural networks. We show that although NARX networks do not circumvent the problem of long–term dependencies, they can greatly improve performance on long-term dependency problems .

We also describe in detail some of the assumption regarding what it means to latch information robustly and suggest possible ways to loosen these assumptions.

1 Introduction

Recurrent Neural Networks (RNNs) are capable of representing arbitrary nonlinear dynamical systems [27, 29, 30]. However, learning simple behavior can be quite difficult using gradient descent. For example, even though these systems are Turing equivalent, it has been difficult to get them to successfully learn small finite state machines from example strings encoded as temporal sequences.

Recently, it has been demonstrated that at least part of this difficulty can be attributed to long–term dependencies, i.e. when the desired output at time T depends on inputs presented at times $t \ll T$. This was noted by Mozer who reported that RNNs were able to learn short term musical structure using gradient based methods [17], but had difficulty capturing global behavior. These ideas were recently formalized by Bengio *et al.* [2], who showed that if a system is to robustly latch information, then the fraction of the gradient due to information n time steps in the past approaches zero as n becomes large.

Several approaches have been suggested to circumvent the problem of vanishing gradients. For example, gradient–based methods can be abandoned completely in favor of alternative optimization methods [2, 21]. However, the algorithms investigated so far either perform just as poorly on problems involving long–term dependencies, or, when they are better, require far more computational resources [2]. Another possibility is to modify conventional gradient descent by more heavily weighing the fraction of the gradient due to information far in the past, but there is no guarantee that such a modified algorithm would converge to a minima of the error surface being searched [2]. As an alternative to using different learning algorithms, one suggestion has been to alter the input data so that it represents a reduced description that makes global features more explicit and more readily detectable [17, 26, 25]. However, this approach may fail if short term dependencies are equally as important. Finally, it has been suggested that a network architecture that operates on multiple time scales might be useful for approaching this problem [12].

In this paper, we also propose an architectural approach to deal with long–term dependencies. We focus on a class of architectures based upon Nonlinear AutoRegressive models with eXogenous inputs (NARX models), and are therefore called *NARX recurrent neural networks* [3, 18]. This is a powerful class of models which has recently been shown to be computationally equivalent to Turing machines [28]. It has been demonstrated that they are well suited for modeling nonlinear systems such as heat exchangers [3], waste water treatment plants [31, 32], catalytic reforming systems in a petroleum refinery [32], nonlinear oscillations associated with multi–legged locomotion in biological systems [33], time series [4], and various artificial nonlinear systems [3, 18, 22]. Furthermore, we have previously reported that gradient descent learning is more effective in NARX networks than in recurrent neural network architectures with "hidden states" when applied to problems including grammatical inference and nonlinear system identification [11, 13]. Typically, these networks converge much faster and generalize better than other networks. The results in this paper give an explanation of this phenomenon.

2 Vanishing gradients and long–term dependencies

Bengio *et al.* [2] have analytically explained why learning problems with long–term dependencies is difficult. They argue that for many practical applications the goal of the network must be to *robustly latch information*, i.e. the network must be able to store information for a long period of time in the presence of noise. More specifically, they argue that latching of information is accomplished when the states of the network stay within the vicinity of a hyperbolic attractor, and robustness to noise is accomplished if the states of the network are contained in the *reduced attracting set* that attractor, i.e. if the eigenvalues of the Jacobian are contained within the unit circle. In Section 6, we discuss this definition of robustness in more detail and describe how some of the assumptions associated with it might be loosened. In this section we briefly describe some of the key aspects of the results in [2].

A recurrent neural network can be written in the form

$$\mathbf{x}(t+1) = f(\mathbf{x}(t), \mathbf{u}(t); \mathbf{w}) \tag{1}$$

$$\mathbf{y}(t) = g(\mathbf{x}(t)) \tag{2}$$

where $\mathbf{x}$, $\mathbf{u}$, $\mathbf{y}$ and $\mathbf{w}$ are column vectors representing the state, input, output and weights of the network respectively. Almost any recurrent neural network architecture can be expressed in this form [19], where f and g depend on the specific architecture. For example, in simple first–order recurrent neural networks f would be a sigmoid of a weighted sum of the values $\mathbf{x}(t)$ and $\mathbf{u}(t)$ and

g would simply select one of the states as output.

We define $\mathbf{u}_p(t)$, $t = 1...T$ to be an input sequence of length T for the network (for simplicity we shall assume that all sequences are of the same length), and $\mathbf{y}_p(T)$ to be the output of the network for that input sequence.

In what follows we derive the gradient descent learning algorithm in a matrix–vector format which is slightly more compact than deriving it expressly in terms of partial derivatives, and highlights the role of the Jacobian in the derivation.

Gradient descent learning is typically based on minimizing the sum–of–squared error cost function

$$C = \frac{1}{2} \sum_p \left(\mathbf{y}_p(T) - \mathbf{d}_p\right)^{\mathrm{T}} \left(\mathbf{y}_p(T) - \mathbf{d}_p\right)$$

where $\mathbf{d}_p$ is the desired (or target) output for the pth pattern[1]. Gradient descent is an algorithm which iteratively updates the weights in proportion to the gradient

$$\Delta \mathbf{w} = \eta \nabla_{\mathbf{w}}^{\mathrm{T}} C \ .$$

where η is a learning rate, and $\nabla_{\mathbf{w}}$ is the row vector operator

$$\nabla_{\mathbf{w}} = \begin{bmatrix} \frac{\partial}{\partial w_1} & \frac{\partial}{\partial w_2} & \ldots & \frac{\partial}{\partial w_n} \end{bmatrix} \ .$$

By using the Chain Rule, the gradient can be expanded

$$\nabla_{\mathbf{w}} C = \sum_p \left(\mathbf{y}_p(T) - \mathbf{d}_p\right)^{\mathrm{T}} \nabla_{\mathbf{x}(T)} \mathbf{y}_p(T) \nabla_{\mathbf{w}} \mathbf{x}(T) \ .$$

We can expand this further by assuming that the weights at different time indices are independent and computing the partial gradient with respect to these weights, which is the trick used to derive algorithms such as Backpropagation Through Time (BPTT) [24, 23]. The total gradient is then

[1] We deal only with problems in which the target output is presented at the *end* of the sequence.

equal to the sum of these partial gradients. Specifically,

$$\nabla_{\mathbf{w}} C = \sum_p \left(\mathbf{y}_p(T) - \mathbf{d}_p\right) \nabla_{\mathbf{x}(T)} y_p(T) \left[\sum_{\tau=1}^{T} \nabla_{\mathbf{w}(\tau)} \mathbf{x}(T)\right] .$$

One more application of the Chain Rule gives

$$\nabla_{\mathbf{w}} C = \sum_p \left(\mathbf{y}_p(T) - \mathbf{d}_p\right) \nabla_{\mathbf{x}(T)} y_p(T) \left[\sum_{\tau=1}^{T} J_{\mathbf{x}}(T, T-\tau) \nabla_{\mathbf{w}(\tau)} \mathbf{x}(\tau)\right] ,$$

where $J_{\mathbf{x}}(T, T-\tau) = \nabla_{\mathbf{x}(\tau)} \mathbf{x}(T)$ denotes the Jacobian of (1) expanded over $T - \tau$ time steps.

Bengio *et al.* [2] showed that if the network satisfies their definition of robustly latching information, i.e. if the Jacobian at each time step has all of its eigenvalues inside the unit circle, then $J_{\mathbf{x}}(T, n)$ is an exponentially decreasing function of n, so that $\lim_{n \to \infty} J_{\mathbf{x}}(T, n) = 0$. This implies that the portion of $\nabla_{\mathbf{w}} C$ due to information at times $\tau \ll T$ is insignificant compared to the portion at times near T. This effect is called the problem of *vanishing gradient*, or *forgetting behavior* [9]. Bengio *et al.* claim that the problem of vanishing gradients is the essential reason why gradient descent methods are not sufficiently powerful to discover a relationship between target outputs and inputs that occur at a much earlier time, which they term the problem of *long–term dependencies.*

3 NARX networks

An important class of discrete–time nonlinear systems is the *Nonlinear AutoRegressive with eXogenous inputs* (NARX) model [3, 15, 16, 31, 32]:

$$y(t) = f\left(u(t - D_u), \ldots, u(t-1), u(t), y(t - D_y), \ldots, y(t-1)\right), \tag{3}$$

where $u(t)$ and $y(t)$ represent input and output of the network at time t, D_u and D_y are the input and output order, and the function f is a nonlinear function. When the function f can be approximated by a Multilayer Perceptron, the resulting system is called a *NARX recurrent neural network* [3, 18].

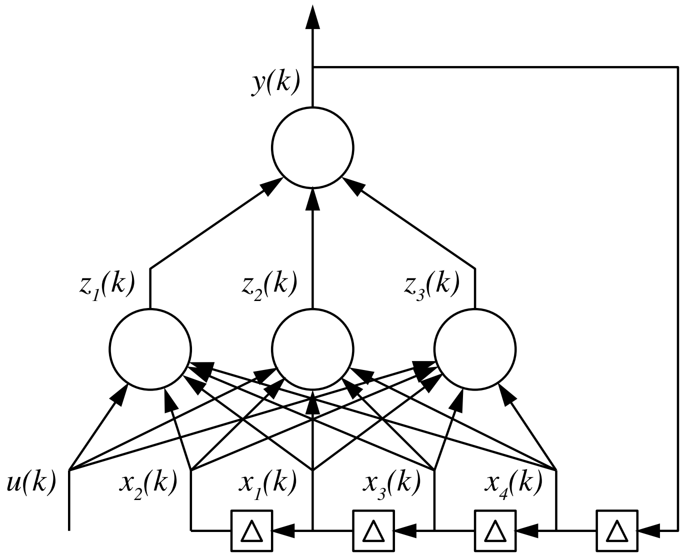

Figure 1: A NARX network with four output delays.

In this paper we shall consider NARX networks with zero input order and a one dimensional output, i.e. those networks which have feedback from the output only. However there is no reason why our results could not be extended to networks with higher input orders. Thus, the operation of the network is defined by

$$y(t) = \Psi\left(u(t), y(t-1), \ldots, y(t-D)\right) \ , \tag{4}$$

where the function Ψ is the mapping performed by the MLP, as shown in Figure 1.

Since the states of a discrete–time dynamical system can always be associated with the unit–delay elements in the realization of the system, we can then describe such a network in the following state space form

$$x_i(t+1) = \begin{cases} \Psi\left(u(t), \mathbf{x}(t)\right) & i = 1 \\ x_{i-1}(t) & i = 2, \ldots, D \end{cases} \tag{5}$$

and

$$y(t) = x_1(t+1) \; .$$

NARX networks are not immune to the problem of long–term dependencies. The Jacobian of the state space map (5) is given by

$$J_{\mathbf{x}}(t+1,1) = \nabla_{\mathbf{x}(t)}\mathbf{x}(t+1) = \begin{bmatrix} \frac{\partial\Psi(t)}{\partial x_1(t)} & \frac{\partial\Psi(t)}{\partial x_2(t)} & \cdots & \frac{\partial\Psi(t)}{\partial x_{D-1}(t)} & \frac{\partial\Psi(t)}{\partial x_D(t)} \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{bmatrix}$$

If the Jacobian at each time step has all of its eigenvalues inside the unit circle, then the states of the network will be in the reduced attracting set of some hyperbolic attractor, and thus the system will be robustly latched at that time. As with any other recurrent neural network, this implies that $\lim_{n\to\infty} J_{\mathbf{x}}(t,n) = 0$. Thus, NARX networks will also suffer from vanishing gradients and the long–term dependencies problem.

4 When are NARX networks better at discovering long–term dependencies?

In the previous section we saw that NARX networks also suffer from the problem of vanishing gradients, and thus are also prone to the problem of long–term dependencies. However, we find in the simulation results that follow that NARX networks are often much better at discovering long–term dependencies than conventional recurrent neural networks.

An intuitive reason why output delays help can help long–term dependencies can be found by considering how gradients are calculated using the Backpropagation Through Time algorithm.

BPTT involves two phases: unfolding the network in time and backpropagating the error through the unfolded network. When a NARX network is unfolded in time, the output delays will appear as jump–ahead connections in the unfolded network. Intuitively, these jump–ahead connections provide a shorter path for propagating gradient information, thus reducing the sensitivity of the network to long–term dependencies. However, one must keep in mind that this intuitive reasoning is only valid if the total gradient through these jump–ahead pathways is greater than the gradient through the layer–to–layer pathways.

Another intuitive explanation is that since the delays are cascaded together, the proagation of information does not necessarily have to pass through a nonlinearity at each time step, and thus the gradient is not modified by the derivative of the nonlinearity, which is often less than one in magnitude.

It is possible to derive analytical results for some simple toy problems to show that NARX networks are indeed less sensitive to long–term dependencies. Here we give one such example, which is based upon the latching problem described in [2].

Consider the simple one node autonomous recurrent network described by,

$$x(t) = \tanh(wx(t-1))$$

where $w = 1.25$, which has two stable fixed points at ± 0.710 and one unstable fixed point at zero. The following one node, autonomous NARX network

$$x(t) = \tanh\left(w_1 x(t-1) + w_2 x(t-2) + \ldots + w_D x(t-D)\right)$$

with D output delays has the same fixed points as long as $\sum_{i=1}^{D} w_i = w$.

Assume the state of the network has reached equilibrium at the positive stable fixed point and there are no external inputs. In this case we can derive the exact gradient. For simplicity, we only consider the Jacobian $J(t,n) = \frac{\partial x(t)}{\partial x(t-n)}$, which will be a component of the gradient $\nabla_{\mathbf{w}} C$. Figure 2 shows plots of $J(t,n)$ with respect to n for $D = 1$, $D = 3$ and $D = 6$ with $w_i = w/D$. These plots

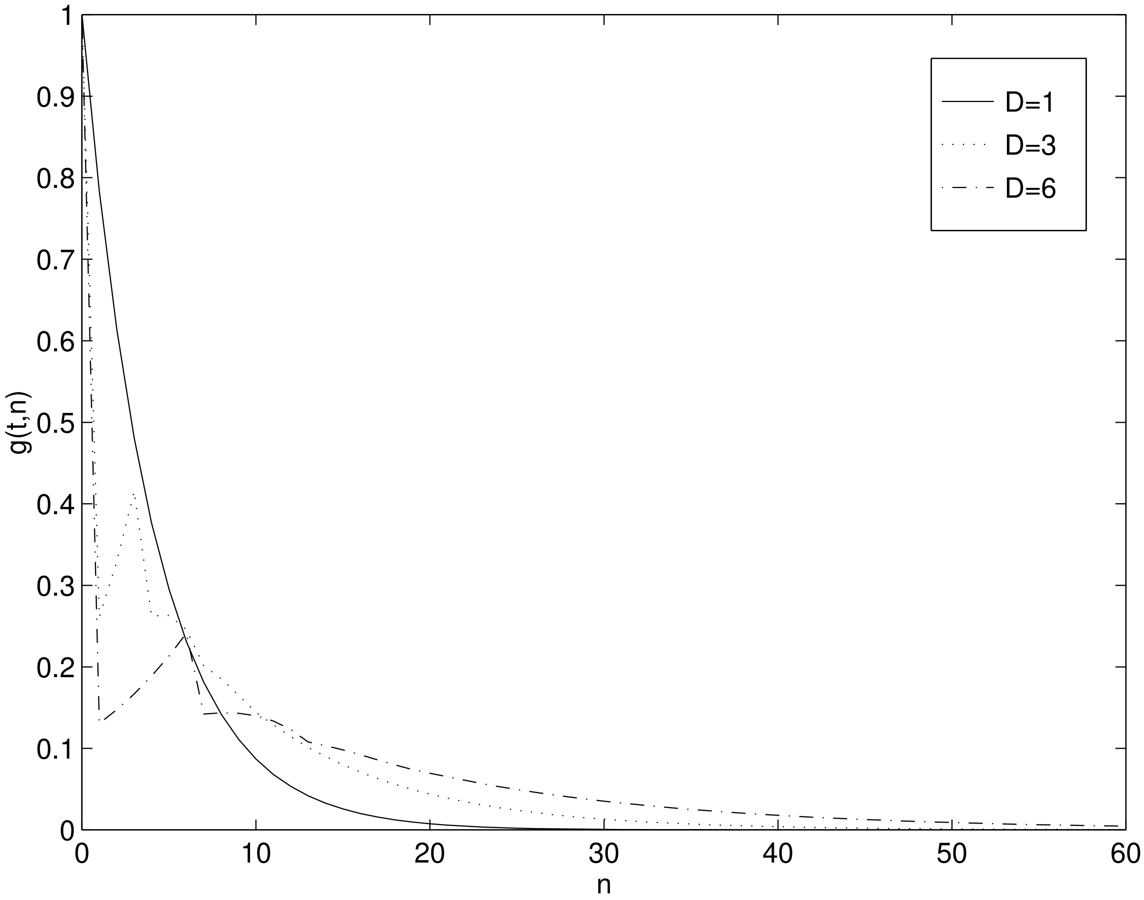

Figure 2: Plots of $J(t, n)$ (the Jacobian expanded over n time steps) as a function of n for different number of output delays ($D = 1$, $D = 3$ and $D = 6$). Although all of these curves can be bounded above by a function that decays exponentially, the values of $J(t, n)$ decay at a slower rate as D becomes larger.

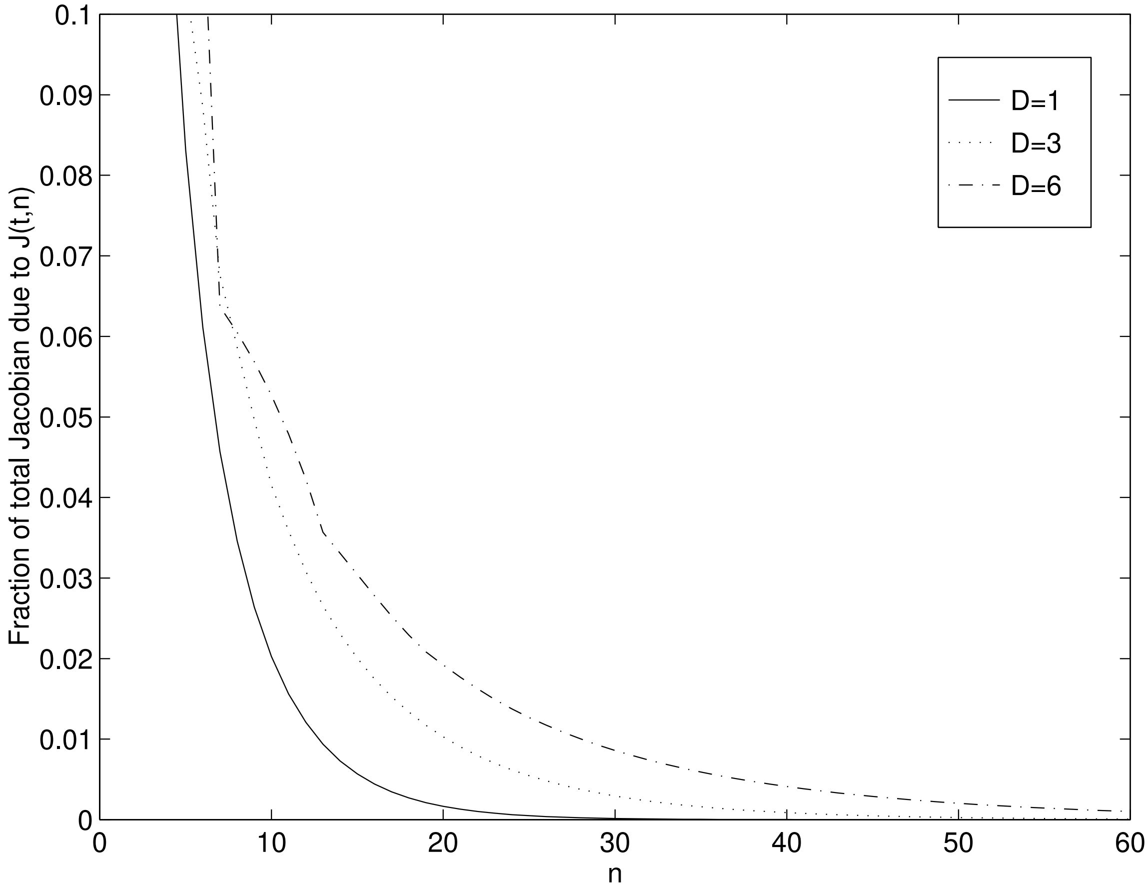

Figure 3: Plots of the ratio $\frac{J(t,n)}{\sum_{\tau=1}^{n} J(t,\tau)}$ as a function of n for different number of output delays ($D = 1$, $D = 3$ and $D = 6$). These curves show that the portion of the gradient due to information n time steps in the past is a greater fraction of the overall gradient as D becomes larger.

show that the effect of output delays is to flatten out the curves and place more emphasis on the gradient due to terms farther in the past. Note that the gradient contribution due to short term dependencies is deemphasized. In Figure 3 we show plots of the ratio

$$\frac{J(t,n)}{\sum_{\tau=1}^{n} J(t,\tau)} ,$$

which illustrates the percentage of the total gradient that can be attributed to information n time steps in the past. These plots show that this percentage is larger for the network with output delays, and thus one would expect that these networks would be able to more effectively deal with long–term dependencies.

5 Experimental results

Simulations were performed to compare the performance of learning long–term dependencies on networks with different number of feedback delays. We tried two different problems: the latching problem and the parity problem.

5.1 The latching problem

We explored a slight modification on the latching problem described in [2]. This problem is a minimal task designed as a test that must necessarily be passed in order for a network to robustly latch information. Bengio *et al.* describe the task as one in which *the input values are to be learned.* Here we give an alternative description of the problem, which allows us to reexpress the problem as one in which only weights are to be learned.

In this task there are three inputs $u_1(t)$, $u_2(t)$, and a noise input $e(t)$, and a single output $y(t)$. Both $u_1(t)$ and $u_2(t)$are zero for all times $t > 1$. At time $t = 1$, $u_1(1) = 1$ and $u_2(1) = 0$ for samples

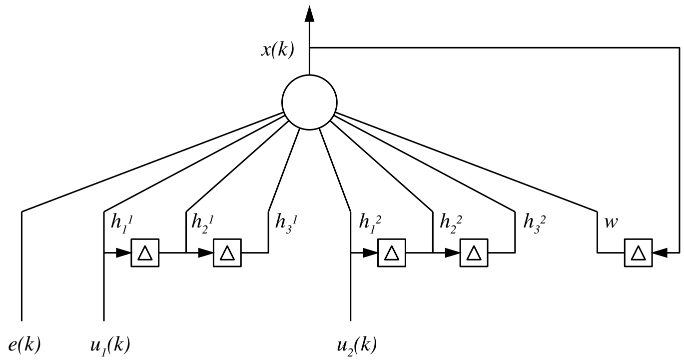

Figure 4: The network used for the latching problem.

from class 1, and $u_1(1) = 0$ and $u_2(1) = 1$ for samples from class 2. The noise input $e(t)$ is given by

$$e(t) = \begin{cases} 0 & t \leq L \\ U(-b, b) & L < t \leq T \end{cases}$$

where $U(-b, b)$ are samples drawn uniformly from $[-b, b]$.

This network used to solve this problem is a NARX network consisting of a single neuron,

$$x(t) = \tanh\left(wx(t-1) + h_1^1 u_1(t) + \ldots + h_L^1 u_1(t) + h_1^2 u_2(t) + \ldots + h_L^2 u_2(t) + e(t)\right)$$

where the parameters h_i^j are adjustable and the recurrent weight w is fixed. In our simulations, we used $L = 3$. The network is shown in Figure 4.

Note that the problem as stated is identical to the problem stated by Bengio *et al.* except that here we are using uniform instead of Gaussian random noise. In our formulation the values h_i^j are weights which are connected to tapped delay lines on the input of the network, while Bengio *et al.* describe them as learnable input values.

In our simulation, we fixed the recurrent feedback weight to $w = 1.25$, which gives the autonomous network two stable fixed points at ± 0.710 and one unstable fixed point at zero, as described in Section 4. It can be shown [8] that the network is robust to perturbations in the range $[-0.155, 0.155]$.

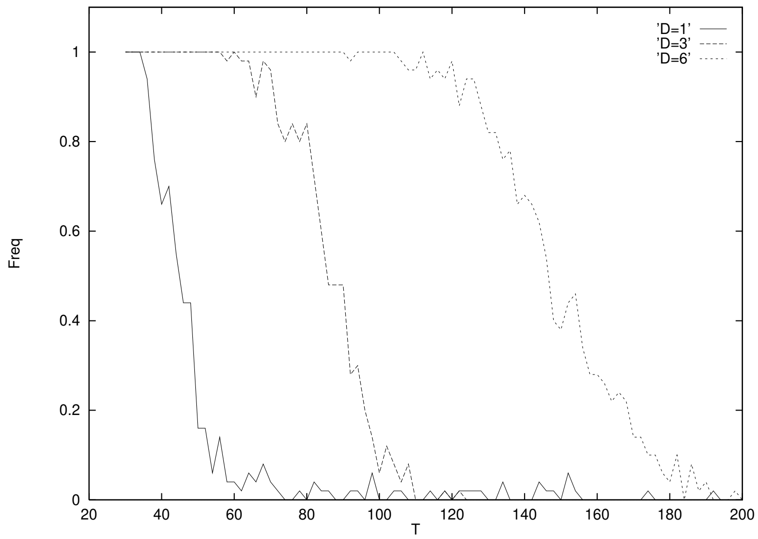

Figure 5: Plots of percentage of successful simulations as a function of T, the length of the input strings, for different number of output delays ($D = 1$, $D = 3$ and $D = 6$).

Thus, the uniform noise in $e(t)$ was restricted to this range. Note that if Gaussian random noise is used, then there is some non–zero probability that the error would be outside of this range regardless of the variance, and thus it is possible for the network to fail to correctly classify all training values due to Gaussian noise. We felt that such effects should be avoided in order to exclusively test the sensitivity of the network to long–term dependencies, and so we chose to use uniform noise instead.

For each simulation, we generated 30 strings from each class, each with a different $e(t)$. The initial values of h_i^j for each simulation were also chosen from the same distribution that defines $e(t)$. For strings from class one, a target value of 0.8 was chosen, for class two -0.8 was chosen.

The network was run using a simple BPTT algorithm with a learning rate of 0.1 for a maximum of 100 epochs. (We found that the network converged to some solution consistently within a few dozen epochs.) If the absolute error between the output of the network and the target value was less than 0.6 on all strings, the simulation was terminated and determined successful. If the simulation exceeded 100 epochs and did not correctly classify all strings then the simulation was ruled a failure.

We varied T from 20 to 200 in increments of 2. For each value of T, we ran 50 simulations.

We then modified the architecture to include output delays of order $D = 3$ and $D = 6$, with all of the recurrent weights $w_i = w/D$. Figure 5 shows a plot of the percentage of those runs that were successful for each case. It is clear from these plots that the NARX networks become increasingly less sensitive to long–term dependencies as the output order is increased.

5.2 Tree Automaton

In previous problem, the inputs to the network consisted of a noise term whose magnitude was restricted in such a way that the network was guaranteed remain within the basin of attraction of the fixed points for a single node. Here we explore two extensions to that problem. First, we consider larger networks, and second we consider inputs which are not as restrictive. In particular, we consider learning an automata problem in which the inputs are boolean valued.

In this example, the class of a string is completely determined by its input symbol at some prespecified time t. For example, Figure 6 shows a five-state automaton, in which the class of each string is determined by the third input symbol. When that symbol is "1", the string is accepted; otherwise, it is rejected. By increasing the length of the strings to be learned, we will be able to create a problem with long term dependencies, in which the output will depend on input values far in the past.

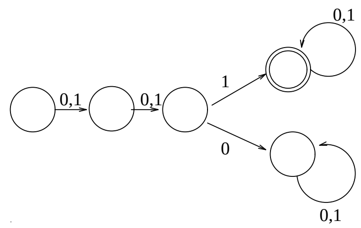

Figure 6: A five state tree automaton.

In this experiment we compared Elman's Simple Recurrent Network [7] against NARX networks. Each network had six hidden nodes. Since the output if each hidden node in an Elman network is fed back, there were six delay elements (states) in the network. The NARX network had six feedback delays from the output node. Thus, the two architectures have the exact same number of weights,

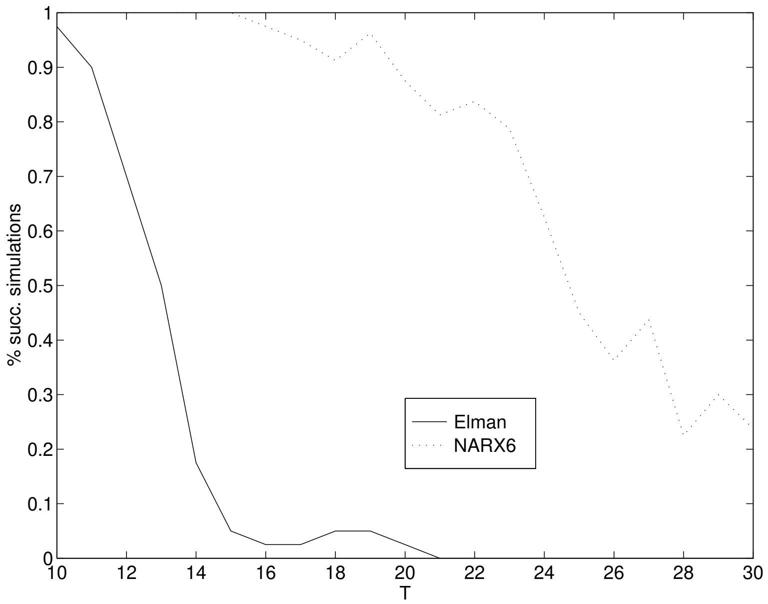

Figure 7: Plots of percentage of successful simulations as a function of T, the length of input strings, for the Elman networks v.s. NARX networks.

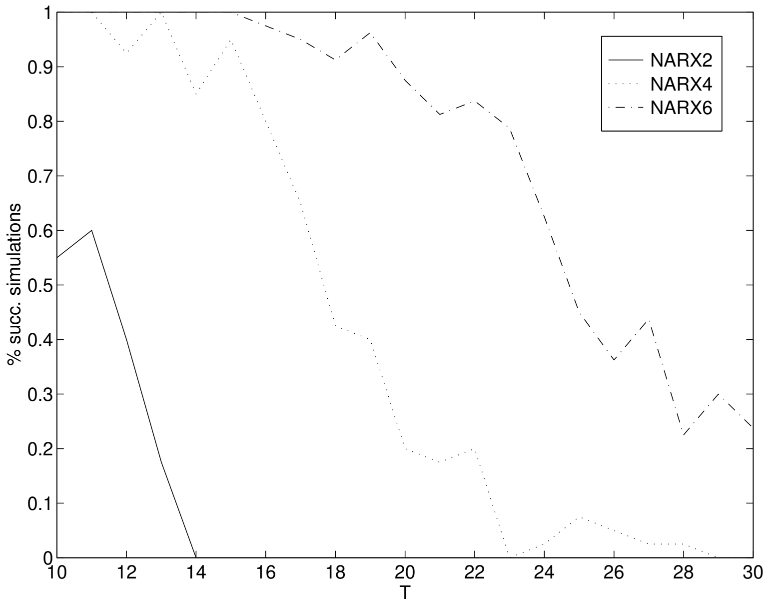

Figure 8: Plots of percentage of successful simulations as a function of T, the length of the input strings, for NARX networks with different of output delays($D = 2$, $D = 4$, $D = 6$).

hidden nodes, and states. The initial weights were randomly distributed in the range $[-0.5, 0.5]$.

For each simulation, we randomly generated a training set and an independent testing set, each consisting of 500 strings of length T such that there were an equal number of positive and negative strings. We varied T from 10 to 30. For the accepted strings, a target value of 0.8 was chosen, for the rejected strings -0.8 was chosen.

The network was trained using a simple BPTT algorithm with a learning rate 0.01 for a maximum of 200 epochs. If the simulation exceeded 200 epochs and did not correctly classify all strings in the training set, then the simulation was ruled a failure. We found that when the network learned the training set perfectly, then it consistently performed perfectly on the testing set as well. For each value of T, we ran 80 simulations.

Figure 7 shows a plot of the percentage of the runs that were successful in each case. It is clear from this plot that the NARX network performs far better than the Elman network at learning long-term dependencies.

We also wanted to see how the performance varied due to different numbers of output delays. We chose three different networks in which the size of the output tapped delay line was chosen to be either 2, 4, or 6. To make the total number of trainable weights comparable, the networks had 11, 8, and 6 hidden nodes respectively, giving 56, 57, and 55 weights.

Figure 8 shows the result of the experiment. It is clear that the sensitivity to the long-term dependencies decreases as the number of output delays increases.

6 A closer look at robust information latching

In this section we make a critical examination of the definition of robust latching given by Bengio *et al.* [2]. Specifically, they assume that if a network is to be robust to noise, then the states must always be in the reduced attracting set of the hyperbolic attractor. While such a condition is sufficient to robustly latch information, it is not necessary. In this section we show how robustness may be redefined to be both necessary and sufficient.

First, Bengio *et al.* assume the existence of a "class–determining" subsystem that computes information about the class of an input sequence v. If, say, only the first L values in the input sequence (to be classified) are relevant for determining the class of v, the output of the subsystem is some valuable signal of length L, coding the class, whereas the outputs at times greater than L are unimportant and can be considered minor fluctuations. In their experiments, the fluctuations are modeled as a zero–mean Gaussian noise with a small variance.

The outputs $u(t)$ of the class–determining subsystem feed a latching subsystem,

$$\mathcal{S}: \quad \tilde{\mathbf{x}}(t) = M(\tilde{\mathbf{x}}(t-1)) + u(t) \ .$$

It will be useful to consider the corresponding autonomous dynamical system

$$\mathcal{S}_A: \quad \mathbf{x}(t) = M(\mathbf{x}(t-1)) \ .$$

The key role in latching the class information of $\{\tilde{\mathbf{x}}(t)\}$ in $\mathcal{S}$ is played by the hyperbolic attractors of $\{\mathbf{x}(t)\}$ in $\mathcal{S}_A$. It is assumed that the important class information is coded in the first L time steps of $u(t)$; inputs at times $t > L$ are unimportant and can be considered as noise. Note that this is the key reason why Bengio *et al.* needed to assume the existence of a class–determining subsystem, which will somehow "highlight" the important information at times $t \leq L$, but suppress the information in the succeeding times steps.

The important inputs at times $t \leq L$, cause the states $\tilde{\mathbf{x}}(t)$ to move to the "vicinity" of a hyperbolic attractor X of $\mathcal{S}_A$. If the values of $u(t)$ for $t > L$ are sufficiently small, then the states of $\mathcal{S}$ will not move away from X, thus latching the information coded in $u(1), \ldots, u(L)$ for an arbitrary long time.

Having established this scenario for latching information of possibly long input sequences, Bengio *et al.* discuss what it means for the system to be robust. Specifically, they allow the input to be noisy but bounded, i.e. $\|u(t)\| < b(t)$ such that the latching system $\mathcal{S}$ initiated in a state from $\Gamma(X)$, receiving additive inputs bounded by $b(t)$, will stay in a vicinity of X.

They conclude that $\Gamma(X)$ is a subset of the basin of attraction $\beta(X)$ of X (in $\mathcal{S}_A$), such that for all $\mathbf{x} \in \Gamma(X)$ and $l \geq 1$, the eigenvalues of $J_{\mathbf{x}}(t,l)$ are contained within the unit circle. Such a set is called *the reduced attracting set of* X. Specific bounds of $b(t)$ are given so that $\tilde{\mathbf{x}}(t)$ are asymptotically guaranteed to stay within a prescribed neighborhood of X.

They point out that if the network is to robustly latch information, then it must necessarily suffer from the problem of vanishing gradients, i.e. $\mathbf{x}(t) \in \Gamma(X)$ implies $\|J_{\mathbf{x}}(\tau,1)\| = \|\nabla_{\mathbf{x}(\tau)}\mathbf{x}(\tau+1)\| < 1$, for $t \leq \tau < T$ and therefore when $t \ll T$, we have $\|\nabla_{\mathbf{x}(t)}\mathbf{x}(T)\| \to 0$.

While their analysis is valuable for pointing out problems associated with learning long–term dependencies using gradient descent methods, their definition of robustness is too strong. In the remainder of this section we discuss conditions that are both necessary and sufficient for the network to be robust to noise.

Bengio *et al.* require that $\Gamma(X)$ be the reduced attracting set of X, but it is sufficient to find a set of possible states in the basin of attraction of X such that the system $\mathcal{S}$, fed with sufficiently small inputs $u(t)$, does not diverge from X.

A useful formalization of this idea in dynamical systems' theory is stated in terms of the *shadowing lemma* [5, 10]. Given a number $b > 0$, a b–pseudo–orbit of the system $\mathcal{S}_A$ is a sequence $\{\tilde{\mathbf{x}}(t)\}$ such that $\|M(\tilde{\mathbf{x}}(t)) - \tilde{\mathbf{x}}(t+1)\| < b$, for all $t \geq 0$. Pseudo–orbits arise as trajectories of the autonomous system $\mathcal{S}_A$ contaminated by a noise bounded by b. One may ask a question to what extent are such "corrupted" state trajectories $\{\tilde{\mathbf{x}}(t)\}$ informative about the "real" trajectories $\{\mathbf{x}(t)\}$ of the autonomous system $\mathcal{S}_A$. It turns out that in systems having the so called *shadowing property*, corrupted state trajectories are "shadowed" by real trajectories within a distance depending on the level of the input noise. Bigger noise implies looser shadowing of the corrupted trajectory by an uncorrupted one. Formally, system $\mathcal{S}_A$ has a shadowing property if for every $\epsilon > 0$, there exists a $b > 0$, such that any b–pseudo–orbit $\{\tilde{\mathbf{x}}(t)\}$ is ϵ–approximated by an actual orbit of $\mathcal{S}_A$ initiated in some state $\mathbf{x}(0)$, i.e. $\|\tilde{\mathbf{x}}(t) - M^t(\mathbf{x}(0))\| < \epsilon$, where M^t means the composition of M with itself t times, and M^0 is the identity map.

It is proved in [10] that except possibly for small exceptional sets, discrete–time analog neural networks do have the shadowing property. In particular, they show that that the shadowing property holds for networks with sigmoidal (i.e. strictly increasing, bounded form above and below, and continuously differentiable) activation functions.

As long as $\mathcal{S}_A$ has the shadowing property, it is sufficient to pick arbitrary small $\epsilon > 0$ and start in a point $\tilde{\mathbf{x}}(0) \in \beta(X)$ whose distance from the border of $\beta(X)$ is at least ϵ. Then there exists a bound b on additive noise $u(t)$ such that a "corrupted" trajectory $\{\tilde{\mathbf{x}}(t)\}$ of $\mathcal{S}_A$ (i.e. a trajectory of $\mathcal{S}$) will be "shadowed" by a real trajectory $\{\mathbf{x}(t)\}$ of $\mathcal{S}_A$ originating in some $\mathbf{x}(0)$ from the ϵ–neighborhood of $\tilde{\mathbf{x}}(0)$. Since $\mathbf{x}(0) \in \beta(X)$, $\{\mathbf{x}(t)\}$ converges to X and so $\tilde{\mathbf{x}}(t)$ will not move away from X. Smaller ϵ results in tighter bounds b.

Hence, to achieve a "robust" latch of an information to an attractor X, it is not strictly necessary for the states to be in the reduced attracting set of X. In fact, for every state $\tilde{\mathbf{x}}(0)$ from the basin of attraction $\beta(X)$ of X, there exists a bound on additive inputs $u(t)$ such that $\{\tilde{\mathbf{x}}(t)\}$ will asymptotically stay in an ϵ–neighborhood of X.

7 Conclusion

In this paper we considered an architectural approach to dealing with the problem of learning long–term dependencies, i.e. when the desired output depends on inputs presented at times far in the past, which has been shown to be difficult for gradient based algorithms. We explored the ability of a class of architectures called NARX recurrent neural networks to solve such problems. We found that although NARX networks do not circumvent this problem, it is easier to discover long–term dependencies with gradient descent in these architectures than in architectures without output delays. This has been observed previously, in the sense that gradient descent learning appeared to be more effective in NARX networks than in recurrent neural network architectures that have "hidden states" on problems including grammatical inference and nonlinear system identification [11, 13].

The intuitive explanation for this behavior is that the output delays are manifested as jump–ahead connections in the unfolded network that is often used to describe algorithms like Backpropagation Through Time. Another explanation is that the states do not necessarily need to propagate through nonlinearities at every time step, which may avoid a degradation in gradient due to the partial derivative of the nonlinearity.

We presented an analytical example that showed that the gradients do not vanish as quickly in NARX networks as they do in networks without multiple delays when the network is contained in a fixed point. We also presented two experimental problems which show that NARX networks can outperform networks with single delays on some simple problems involving long–term dependencies.

We speculate that similar results could be obtained for other networks. In particular we hypothesize that any network that uses tapped delay feedback [1, 14] would demonstrate improved performance on problems involving long–term dependencies. It may also be possible to obtain similar results for the architectures proposed in [6, 9, 20, 34].

We have also described in detail some of the assumptions presented in [2] regarding what it means to latch information robustly. Based on shadowing lemma from dynamical systems' theory we have shown that information can potentially be robustly latched to an attractor X in every state $\mathbf{x}$ of the basin of attraction $\beta(X)$ of X. The closer $\mathbf{x}$ is to the border of $\beta(X)$ the smaller unimportant outputs from the "class–determining" subsystem must be in order to successfully latch the information to X.

Acknowledgements

We would like to thank Andrew Back for many useful discussions on this material.

References

[1] A.D. Back and A.C. Tsoi. FIR and IIR synapses, a new neural network architecture for time series modeling. *Neural Computation*, 3(3):375–385, 1991.

[2] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[3] S. Chen, S.A. Billings, and P.M. Grant. Non–linear system identification using neural networks. *International Journal of Control*, 51(6):1191–1214, 1990.

[4] J. Connor, L.E. Atlas, and D.R. Martin. Recurrent networks and NARMA modeling. In J.E. Moody, S.J. Hanson, and R.P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 301–308, 1992.

[5] E. Coven, I. Kan, and J.Yorke. Pseudo-orbit shadowing in the family of tent maps. *Transactions AMS*, 308:227–241, 1988.

[6] B. de Vries and J.C. Principe. The gamma model — A new neural model for temporal processing. *Neural Networks*, 5:565–576, 1992.

[7] J.L. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.

[8] P. Frasconi, M. Gori, M. Maggini, and G. Soda. Unified integration of explicit rules and learning by example in recurrent networks. *IEEE Transactions on Knowledge and Data Engineering*, 7(2):340–346, 1995.

[9] P. Frasconi, M. Gori, and G. Soda. Local feedback multilayered networks. *Neural Computation*, 4:120–130, 1992.

[10] M. Garzon and F. Botelho. Observability of neural network behavior. In J.D. Cowen, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems 6*, pages 455–462. Morgan Kaufmann, 1994.

[11] C.L. Giles and B.G. Horne. Representation and learning in recurrent neural network architectures. In *Proceedings of the Eigth Yale Workshop on Adaptive and Learning Systems*, pages 128–134, 1994.

[12] M. Gori, M. Maggini, and G. Soda. Scheduling of modular architectures for inductive inference of regular grammars. In *ECAI'94 Workshop on Combining Symbolic and Connectionist Processing, Amsterdam*, pages 78–87. Wiley, August 1994.

[13] B.G. Horne and C.L. Giles. An experimental comparison of recurrent neural networks. In *Advances in Neural Information Processing Systems 7*, 1995. To appear.

[14] R.R. Leighton and B.C. Conrath. The autoregressive backpropagation algorithm. In *Proceedings of the International Joint Conference on Neural Networks*, volume 2, pages 369–377, July 1991.

[15] I.J. Leontaritis and S.A. Billings. Input–output parametric models for non–linear systems: Part I: deterministic non–linear systems. *International Journal of Control*, 41(2):303–328, 1985.

[16] L. Ljung. *System identification : Theory for the user.* Prentice-Hall, Englewood Cliffs, NJ, 1987.

[17] M. C. Mozer. Induction of multiscale temporal structure. In J.E. Moody, S. J. Hanson, and R.P. Lippmann, editors, *Neural Information Processing Systems 4*, pages 275–282. Morgan Kaufmann, 1992.

[18] K.S. Narendra and K. Parthasarathy. Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, 1:4–27, March 1990.

[19] O. Nerrand, P. Roussel-Ragot, L. Personnaz, G. Dreyfus, and S. Marcos. Neural networks and nonlinear adaptive filtering: Unifying concepts and new algorithms. *Neural Computation*, 5(2):165–199, 1993.

[20] P. Poddar and K.P. Unnikrishnan. Non–linear prediction of speech signals using memory neuron networks. In B.H. Juang, S.Y. Kung, and C.A. Kamm, editors, *Neural Networks for Signal Processing: Proceedings of the 1991 IEEE Workshop*, pages 1–10. IEEE Press, 1991.

[21] G.V. Puskorius and L.A. Feldkamp. Recurrent network training with the decoupled extended Kalman filter. In *Proceedings of the 1992 SPIE Conference on the Science of Artificial Neural Networks,* Orlando, Florida, April 1992.

[22] S.-Z. Qin, H.-T. Su, and T.J. McAvoy. Comparison of four neural net learning methods for dynamic system identification. *IEEE Transactions on Neural Networks*, 3(1):122–130, 1992.

[23] R.J.Williams and D. Zipser. Gradient-based learning algorithms for recurrent networks and their computational complexity. In Y. Chauvin and D. E. Rumelhart, editors, *Back-propagation: Theory, Architectures and Applications*, chapter 13, pages 433–486. Lawrence Erlbaum Publishers, Hillsdale, N.J., 1995.

[24] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning internal representations by error propagation. In D.E. Rumelhart and J.L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, pages 318–362. MIT Press, Cambridge, MA, 1986.

[25] J. Schmidhuber. Learning complex, extended sequences using the principle of history compression. *Neural Computation*, 4(2):234–242, 1992.

[26] J. Schmidhuber. Learning unambiguous reduced sequence descriptions. In J. E. Moody, S. J. Hanson, and R. P. Lippman, editors, *Advances in Neural Information Processing Systems 4*, pages 291–298. San Mateo, CA: Morgan Kaufmann, 1992.

[27] D.R. Seidl and D. Lorenz. A structure by which a recurrent neural network can approximate a nonlinear dynamic system. In *Proceedings of the International Joint Conference on Neural Networks*, volume 2, pages 709–714, July 1991.

[28] H.T. Siegelmann, B.G. Horne, and C.L. Giles. Computational capabilities of NARX neural networks. Technical Report UMIACS-TR-95-12 and CS-TR-3408, Institute for Advanced Computer Studies, University of Maryland, 1995.

[29] H.T. Siegelmann and E.D. Sontag. On the computational power of neural networks. *Journal of Computer and System Science*, 50(1):132–150, 1995.

[30] E.D. Sontag. Systems combining linearity and saturations and relations to neural networks. Technical Report SYCON–92–01, Rutgers Center for Systems and Control, 1992.

[31] H.-T. Su and T.J. McAvoy. Identification of chemical processes using recurrent networks. In *Proceedings of the American Controls Conference*, volume 3, pages 2314–2319, 1991.

[32] H.-T. Su, T.J. McAvoy, and P. Werbos. Long–term predictions of chemical processes using recurrent neural networks: A parallel training approach. *Industrial Engineering and Chemical Research*, 31:1338–1352, 1992.

[33] S.T. Venkataraman. On encoding nonlinear oscillations in neural networks for locomotion. In *Proceedings of the Eigth Yale Workshop on Adaptive and Learning Systems*, pages 14–20, 1994.

[34] E.A. Wan. Time series prediction by using a connectionist network with internal delay lines. In A.S. Weigend and N.A. Gershenfeld, editors, *Time Series Prediction*, pages 195–217. Addison–Wesley, 1994.