PERGAMON

Neural Networks 11 (1998) 861–868

Neural Networks

Contributed article

How embedded memory in recurrent neural network architectures helps learning long-term temporal dependencies

Tsungnan Lin[a,]*, Bill G. Horne[b], C. Lee Giles[c,d]

[a]*EPSON Palo Alto Laboratory, Palo Alto, CA 94034, USA*
[b]*AADM Consulting, Califon, NJ, USA*
[c]*NEC Research Institute, Princeton, NJ, USA*
[d]*UMIACS, University of Maryland, College Park, MD, USA*

Received 25 November 1996; accepted 4 February 1998

Abstract

Learning long-term temporal dependencies with recurrent neural networks can be a difficult problem. It has recently been shown that a class of recurrent neural networks called NARX networks perform much better than conventional recurrent neural networks for learning certain simple long-term dependency problems. The intuitive explanation for this behavior is that the output memories of a NARX network can be manifested as jump-ahead connections in the time-unfolded network. These jump-ahead connections can propagate gradient information more efficiently, thus reducing the sensitivity of the network to long-term dependencies. This work gives empirical justification to our hypothesis that similar improvements in learning long-term dependencies can be achieved with other classes of recurrent neural network axchitectures simply by increasing the order of the embedded memory. In particular we explore the impact of learning simple long-term dependency problems on three classes of recurrent neural network architectures: globally recurrent networks, locally recurrent networks, and NARX (output feedback) networks.

Comparing the performance of these architectures with different orders of embedded memory on two simple long-term dependencies problems shows that all of these classes of network architectures demonstrate significant improvement on learning long-term dependencies when the orders of embedded memory are increased. These results can be important to a user comfortable with a specific recurrent neural network architecture because simply increasing the embedding memory order of that architecture will make it more robust to the problem of long-term dependency learning. © 1998 Elsevier Science Ltd. All rights reserved.

Keywords: Neural networks; Memory; Recurrent; Long-term dependencies; Training; Gradient-descent; Time-delay; Latching; Automata

1. Introduction

Recurrent neural networks (RNNS) are capable of representing arbitrary nonlinear dynamical systems (Seidl and Lorenz, 1991) and can be computationally quite powerful (Siegelmann and Sontag, 1995). However, various empirical studies suggest that sometimes learning even simple behavior can be quite difficult when using gradient-descent learning algorithms. Recently, it has been demonstrated that at least part of this difficulty can be attributed to the problem of *long-term dependencies* (Bengio et al., 1994; Mozer, 1992), i.e. those problems for which the desired output of a system at time T depends on inputs presented at times $t < T$.

In particular (Bengio et al., 1994) showed that if a system is to latch information robustly, then the fraction of the gradient in a gradient-based training algorithm due to information n time steps in the past approaches zero as n becomes large. This effect is called the problem of *vanishing gradiant*. Bengio et al. claimed that the problem of a vanishing gradient is the essential reason why gradiant-descent methods are not sufficiently powerful to learn long-term dependencies. Several appraoches were suggested to circumvent the problem of vanishing gradients in training RNNS. One possible approach is to preset initial weights by using prior knowledge (Frasconi et al., 1995; Giles and Omlin, 1992; Omlin and Giles, 1996) but this is often not available in many applications. Another approach is to use alternative optimization methods instead of gradient-based methods (Bengio et al., 1994), but, those algorithms can perform as poorly as gradient methods, or require far more computational resources. Alternatively, the input data can be altered to represent a reduced description that makes global features more explicit and more readily

* Requests for reprints should be addressed to C. Lee Giles.

detectable (Mozer, 1992). Unfortunately, this approach may fail if short-term dependencies are equally as important. Hochreiter and Schmidhuber (1995) propose a specific architectural approach which utilizes high-order gating units. It was also suggested that a network architecture that operates on multiple time scales might be useful (Gori et al., 1994; Hihi and Bengio, 1996) and so would architectures that find or learn the appropriate memory orders (Lin et al., 1997; Pedersen and Hansen, 1995).

We have shown that a class of recurrent neural networks called NARX networks can perform much better at learning long-term dependencies when using a gradient descent training algorithm than previously reported in the literature (Lin et al., 1996b; Lin et al., 1996a). The intuitive explanation for this behavior is that the output memories of a NARX neural network are manifested as jump-ahead connections in the time-unfolded network that is often associated with algorithms as backpropagation through time (BPTT). These jump-ahead connections provide shorter paths for propagating gradient information, thus reducing the sensitivity of the network to long-term dependencies. We hypothesize that the similar improvement on learning long-term dependencies can be achieved in other classes of recurrent neural network architectures by increasing the orders of embedded memory. It is worth noting that one of the first uses of embedded memory in recurrent network architectures was that of Jordan (1986).

In this article, we empirically justify this hypothesis by showing the relationship between the memory order of a RNN and its sensitivity to long-term dependencies. In Section 2, we discuss three classes of conventional recurrent neural networks architectures: globally recurrent networks (the architecture, not the training procedure, used by Elman, 1990); locally recurrent networks (in particular the model of Frasconi et al., 1992); NARX networks (Chen et al., 1990; Narendra and Parthasarathy, 1990), and their corresponding models with a high order embedded memory. In Section 3, we provide a empirical comparison of these architectures by investigating their performance on learning two simple long-term dependencies problems: the latching problem and a grammatical inference problem. The simulations show that these classes of recurrent neural network architectures all demonstrate significant improvement on learning long-term dependencies when the embedded memory order is increased and weights remain relatively the same. Thus, a user of one of these recurrent architectures can readily improve their robustness to long-term memory problems simply by increasing the amount of embedded memory, all other variables remaining constant.

2. Embedding high order memory in recurrent neural network architectures

Several recurrent neural network architectures have been proposed; for a collection of papers on the variety see Giles et al. (1994). One taxonomic classification for these architectures can be based on the observability of their states: specifically they can be broadly divided into two groups depending on whether or not the state variables of the network are observable or not (Horne and Giles, 1995). For another taxonomic approach based on memory types, see Mozer (1994). For this study we picked three classes of networks: globally recurrent (GR) networks (Elman, 1990), locally recurrent networks (LR, Frasconi et al., 1992), and NARX networks (Narendra and Parthasarathy, 1990; Chen et al., 1990), and their corresponding architectures with high-order embedded memory. It should be pointed out that our embedded memory simply consists of simple tapped delayed values to various neurons and not more sophisticated embedded memory structures (Mozer, 1994; de Vries and Principe, 1992). NARX networks are a typical model of networks with observable state variables. GR networks are a popular class of network with globally connected hidden state variables, and LR networks belong to locally recurrent network architecture class also with hidden state variables.

2.1. Globally connected rnns

These networks (which we will call GR networks) are a class of recurrent networks in which the feedback connections come from the state vector to the hidden layer, as illustrated in Fig. 1(a). The hidden state variables are sometimes called *context units* in the literature. Suppose such a network with n_u input nodes, n_h hidden nodes, and n_y output nodes, the dynamic equation can be described by:

$$o_i(t) = f\left(\sum_{j=1}^{n_h} w_{ij}^h o_j(t-1) + \sum_{k=1}^{n_u} w_{ik}^u u_k(t) + w_i^b\right) \quad (1)$$

$$y_i(t) = f\left(\sum_{j=1}^{n_h} w_{ij}^y o_j(t) + w_i^b\right) \quad (2)$$

where $o(t)$ and $y(t)$ denote the real valued outputs of the hidden and output neurons at time t, and f is the nonlinear function.

This network with a high order of embedded memory differs from standard globally connected recurrent network in that they have more than one state vector per feedback loop. Specially, for a GR network with embedded memory of order m, the dynamic equations of hidden nodes become:

$$o_i(t) = f\left(\sum_{k=1}^{m}\sum_{j=1}^{n_h} w_{ijm}^h o_j(t-k) + \sum_{k=1}^{n_u} w_{ik}^u u_k(t) + w_i^b\right) \quad (3)$$

Fig. 1(b) illustrates an GR network with embedded memory of order two.

2.2. Locally recurrent networks

In this class of networks, the feedback connections are only allowed from neurons to themselves, and the nodes are

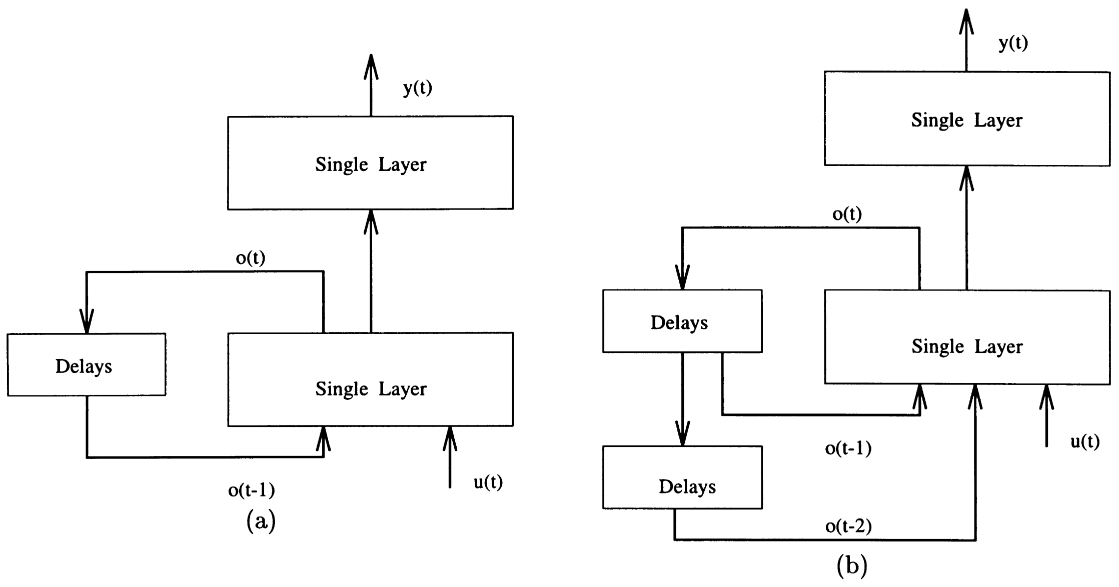


Fig. 1. (a) A standard GR network. (b) A GR network with embedded memory of order 2.

connected together in a feed forward architecture (Back and Tsoi, 1991; Frasconi et al., 1992; Sastry et al., 1994; Tsoi and Back, 1994). Specifically, we consider networks proposed by Frasconi et al. (1992) (we will call LR), as shown in Fig. 2(a). The computational power of these networks is discussed in Frasconi and Gori (1996). The dynamic neurons of LR networks can be described by

$$o_i(t) = f\left(w_{ii}^h o_i(t-1) + \sum_j w_{ij}^u u_j(t) + w_i^b\right) \quad (4)$$

where $o_i(t)$ denotes the output of the ith node at time t, and f is the nonlinearity. For a network with embedded memory of order m, the output of the dynamic neurons becomes

$$o_i(t) = f\left(\sum_{n=1}^{m} w_{ii}^h o_i(t-n) + \sum_j w_{ij}^u u_j(t) + w_i^b\right) \quad (5)$$

Fig. 2(b) shows a LR network with embedded memory of order two. Locally recurrent models usually differ in where and how much output feedback is permitted; see Tsoi and Back (1994) for a discussion of architectural differences.

2.3. NARX recurrent neural networks

An important class of discretetime nonlinear systems is the *Nonlinear AutoRegressive with eXogenous inputs* (NARX) model (Chen et al., 1990; Ljung, 1987; Su and McAvoy, 1991; Su et al., 1992):

$$y(t) = f\big(u(t-D_u), \cdots, u(t-1), u(t), y(t-D_y), \cdots, y(t-1)\big) \quad (6)$$

where $u(t)$ and $y(t)$ represent input and output of the network at time t, D_u and D_y are the input-memory and output-memory order, and the function f is a nonlinear function.

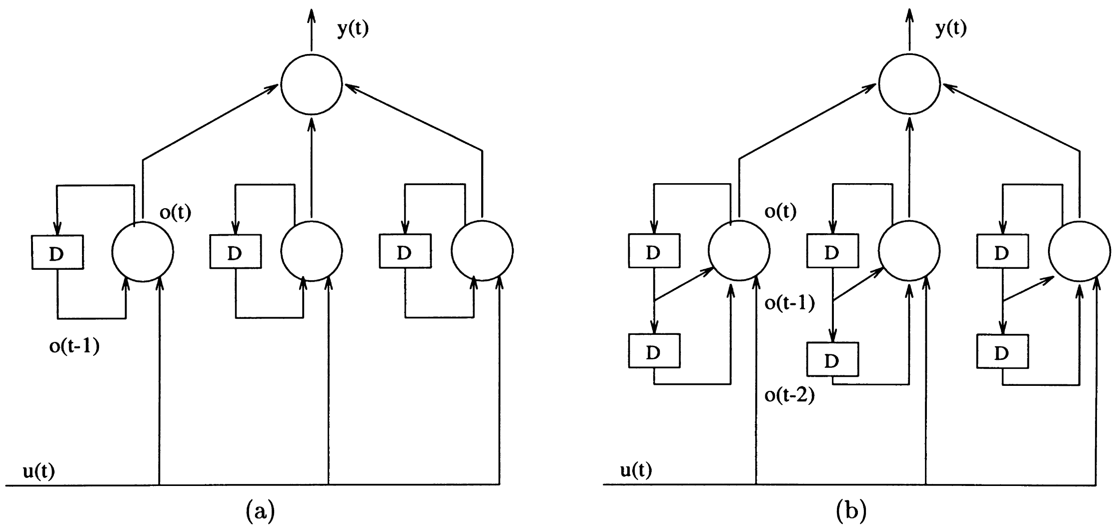


Fig. 2. (a) A standard LR network. (b) A LR network with embedded memory of order 2.

When the function f can be approximated by a multilayer perceptron, the resulting system is called a *NARX recurrent neural network* (Chen et al., 1990; Narendra and Parthasarathy, 1990). Computationally, such networks have been shown to be at least Turing equivalent (Siegelmann et al., 1997). In this paper, we shall consider NARX networks with zero input order. Thus the operation of the network is defined by

$$y(t)=f\big(u(t),y(t-D_y),\cdots,y(t-1)\big) \tag{7}$$

Fig. 3 shows a NARX architecture with output memory of order 3.

3. Experimental results

Simulations were performed to explore the effect of embedded memory on learning long-term dependencies in these three different recurrent network architectures. The long-term dependency problems investigated were the latching problem and a grammatical inference problem. These problems were chosen because they are simple and should be easy to learn but exemplify the long-term dependency issue. For more complex problems involving long-term dependencies see (Hochreiter and Schmidhuber, 1995).

In order to establish some metric for comparison in the experimental results, we gave the recurrent networks sufficient resources (number of weights and training examples, adequate training time) to readily solve the problem but held the the number of weights approximately invariant across all architectures. Also note that in some cases the order of the embedded memory is the same.

3.1. The latching problem

This experiment evaluates the performance of different recurrent network architectures with various orders of embedded memory on a problem already used for studying the difficulty in learning long-term dependencies (Bengio et al., 1994; Hihi and Bengio, 1996; Lin et al., 1996b).

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., 1994). In this two-class problem, the class of a sequence depends only on the first three time steps, the remaining values in the sequence is uniform noise. There are three inputs $u_1(t)$, $u_2(t)$, and a noise input $e(t)$. 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 from class 1, and $u_1(1) = 0$ and $u_2(1) = 1$ for samples from class 2. The class information of each strings is contained in $u_1(t)$ and $u_2(t)$. We used two delay elements for both $u_1(t)$ and $u_2(t)$ in order to hold the class information until $t = 3$. It is not necessary to hold the class information until $t = 3$. Without the delay elements, the networks are still able to learn the problem. The reason to use delay elements is to make our simulation environment identical to (Bengio et al., 1994). The noise input $e(t)$ is given by

$$e(t)\begin{cases}0 & t\le 3\\ U(-b,b) & 3<t\le T\end{cases} \tag{8}$$

where $U(-b, b)$ are samples drawn uniformly from [− 0.155, 0.155]. Target information was only provided at the end of each sequence. For comparison, our training particulars are identical to those of Bengio et al. (1994). For strings from class one, a target value of 0.8 was chosen, for class two, − 0.8 was chosen. The length of the noisy sequence could be varied in order to control the span of long-term depenendencies. Fig. 4 shows the architecture for the latching problem.

For each of these three architectures, several networks with different orders of embedded memory were trained. To compare the effects of different orders of embedded memory in every class of networks on learning long-term dependencies while holding as many other factors as possible constant, particular attention was paid to equalize the number of weights. Table 1 gives a detailed description of all networks used in the latching problem. The noise input $e(t)$ connects to several units, and thus has several weights connected to it. The weight connected to the noise input was fixed as 1.0. In order to learn the task, the networks have to develop two attractors to latch the information and still remain inside the basin of the attractors of being resistant

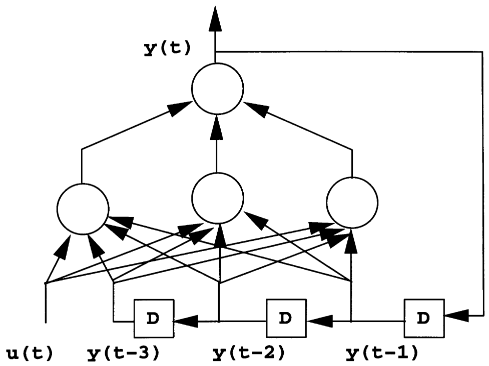


Fig. 3. A NARX network with output memory of order 3.

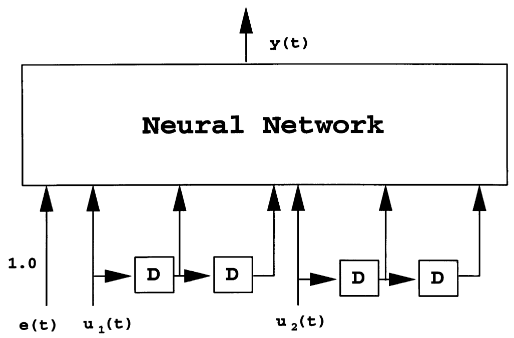


Fig. 4. The network used for the latching problem. The recurrent neural network is to implement the latching system. Different network architectures used in this experiment are described in Table 1. The delay elements connected with $u_1(t)$ and $u_2(t)$ are to hold the class information until $t = 3$.

to noise when $t > 3$. The ability to learn this minimal problem is a measure of the effectiveness of propagating the gradient for different neural network architectures with various memory order.

We varied the length of noisy inputs, T, from 10 to 60 in increments of 2. For each value of T, we ran 50 simulations. For each simulation, we generated 30 strings from each class and the initial weights were randomly distributed in the range $[-0.5, 0.5]$. The network was trained with a MSE cost function using simple BPTT algorithm with a learning rate of 0.1 for a maximum of 200 epochs. Updates occurred at the end of each string and the error was back-propagated the full length of the string. 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 200 epochs and did not correctly classify all strings, then the simulation was ruled a failure. Fig. 5(a)–(c) shows plots of the percentages of those runs that were successful for different classes of networks with different orders of embedded memory. It is clear from these plots that the network architectures with high order embedded memory become increasingly less sensitive to long-term dependencies as the memory order was increased. An interesting comparison between the architectures GR(1) and NARX(6) is shown in Fig. 5(d). Since the two architectures have the exact same number of weights, hidden nodes, and state variables, the only difference is the amount of memory order. Clearly, NARX networks perform far better than the GR networks at learning the latching problem.

3.2. Grammatical inference (tree automata) problem

In previous problem, the inputs to the network were followed by a noise term. In this experiment, we consider learning to classify strings of boolean values, which are labeled according to some prespecified automata.

In this example, the class of a string is completely determined by its input symbol at some prespecified time t. For instance, Fig. 6 shows a five-state automaton, in which the class of each string is determined by the third input symbol. Therefore the decision time is $t = 3$. 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 control the span of long-term dependencies, in which the output will depend on input values far in the past.

For this experiment all inputs were encoded into one input neuron with the two alphabets encoded, respectively, as 0.0 and 1.0. 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. All other experimental parameters were the same as the previous experiment.

Because memory order LR(1) networks were experimentally unable to learn sequences of length greater than 10, different LS networks were used. Table 2 shows all the architectures used in this experiment. The network was trained by 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 would consistently perform perfectly on the testing set as well. For each value of T, we ran 80 simulations. Fig. 7(a)–(c) shows plots of the percentage of the runs that were successful in each case. A comparison between NARX networks and GR networks was shown in Fig. 7(d). Again, we note the same improvement on learning long-term dependencies obtained by increasing the order of embedded memory in each class of recurrent neural network architectures.

4. Conclusion

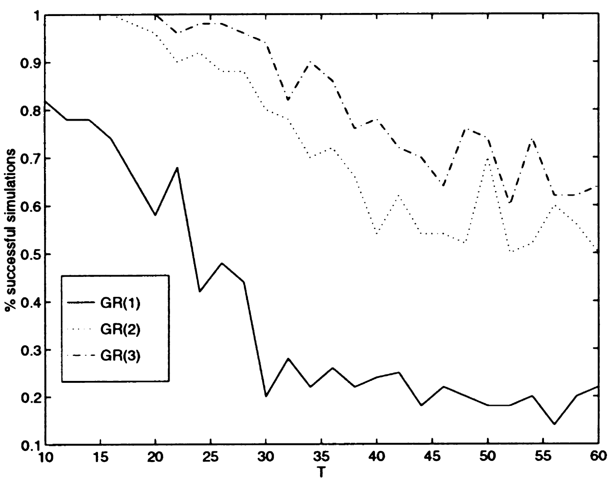
In this article, we explore the impact of embedded memory on various recurrent neural networks architectures for

Table 1
Architecture description of different recurrent networks used for the latching problem. We used the hyperbolic tangent function as the nonlinear function for each neuron

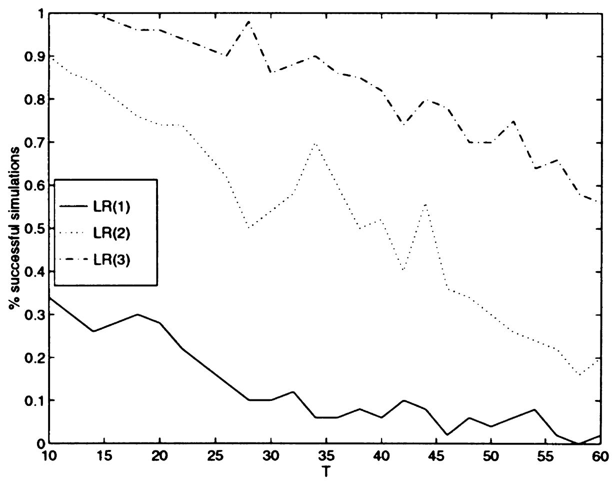
Architecture	Network description				No. weights
	Memory order	No. states	No. hidden neurons	In-hid-out	
GR(1)	1	6	6 nodes	3-6-1	85
GR(2)	2	10	5 nodes	3-5-1	91
GR(3)	3	12	4 nodes	3-4-1	81
NARX(2)	2	2	11 nodes	3-11-1	111
NARX(4)	4	4	8 nodes	3-8-1	97
NARX(6)	6	6	6 nodes	3-6-1	85
LR(1)	1	14	14 nodes	3-14-1	109
LR(2)	2	22	11 nodes	3-11-1	110
LR(3)	3	27	9 nodes	3-9-1	111

Table 2
Architectural description of different recurrent network architecture used for the tree automata problem. We used the hyperbolic tangent function in each neuron

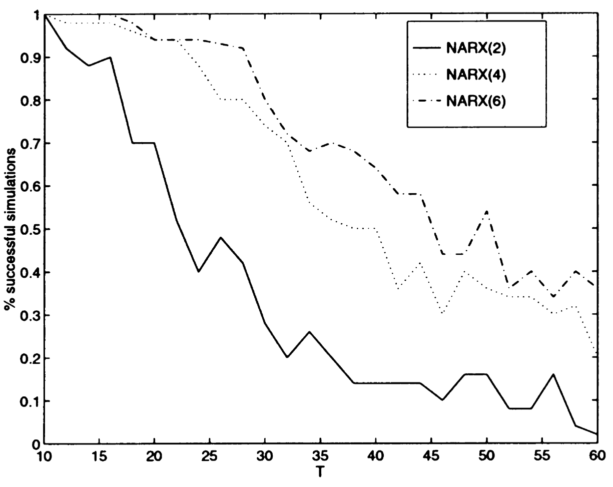
Architecture	Network description				No. weights
	Memory order	No. states	No. hidden neurons	In-hid-out	
GR(1)	1	6	6 nodes	1-6-1	55
GR(2)	2	10	5 nodes	1-5-1	66
GR(3)	3	12	4 nodes	1-4-1	61
NARX(2)	2	1	1 nodes	1-11-1	56
NARX(4)	4	4	8 nodes	1-8-1	57
NARX(6)	6	6	6 nodes	1-6-1	55
LR(2)	2	22	11 nodes	1-11-1	56
LR(4)	4	32	8 nodes	1-8-1	57
LR(6)	6	36	6 nodes	1-6-1	55



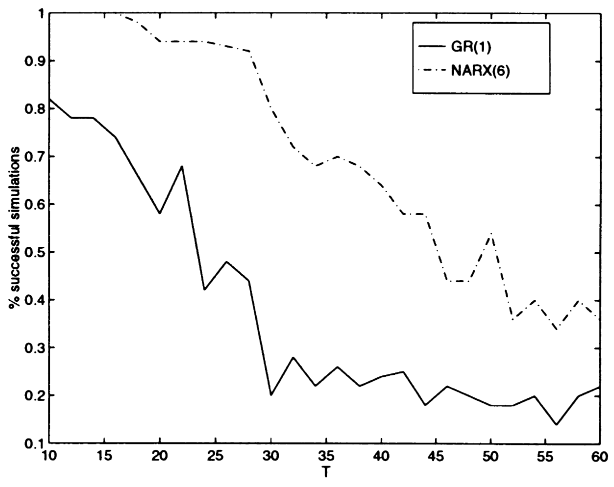
(a)



(b)



(c)



(d)

Fig. 5. Performance on the latching problem. Plots of percentage of successful simulations from 50 runs as a function of *T*, the length of input strings, for different classes of network architectures with different orders of embedded memory: (a) globally connected RNN (GR); (b) locally connected RNN (LR); (c) NARX; (d) NARX vs GR(1).

learning long-term dependency problems, 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. Motivated by the analysis of the problem of learning long-term dependencies and the success of NARX networks on problems including grammatical inference and nonlinear system identification (Horne and Giles, 1995), we explored the ability of other recurrent neural networks with a high order of embedded memory on problems that involve long-term dependencies. We chose three classes of recurrent neural network architectures based on state observability: hidden state globally recurrent and locally recurrent networks, and observable state NARX networks.

We tested this approach of extending memory in conventional recurrent neural networks on two simple long-term dependency problems. However, a specific architecture can perform well if signals and noisy inputs can be separated in high dimensional input space (Hochreiter and Schmidhuber, 1995). Our experimental results show that in each of these classes of recurrent neural networks architectures can demonstrate significant improvement on learning long-term dependencies when the memory order of the network is increased. We speculate such improvement will occur for any recurrent neural network architecture. The intuitive explanation for this behavior is that the embedded memories are manifested as jump-ahead connections in the unfolded network that is often used to describe algorithms like Back-propagation Through Time. These jump-ahead connections provide a shorter path for propagating gradient information, thus reducing the sensitivity of the network to long-term

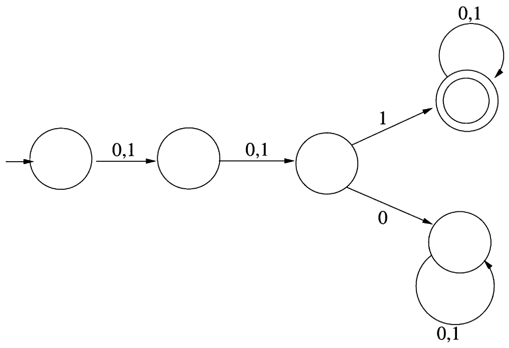


Fig. 6. A five-state tree automaton. The unlabeled arrow is the start state and the double circled state is the the acceptance state.

dependencies. Another explanation is that the states do not necessarily need to propagate through nonlinearities at every time step, which may avoid a degradation in the gradient caused by the partial derivative of the nonlinearity. We speculate that using increased memory order will also help other recurrent network architectures on learning long-term dependency problems. Although specific architectures can be constructed for this problem, the approach of increasing memory order can easily be applied to any recurrent architecture already in use, of course at the cost of increased numbers of weights. It is an open question as to how other embedded memory models will affect the problem of learning long-term dependencies.

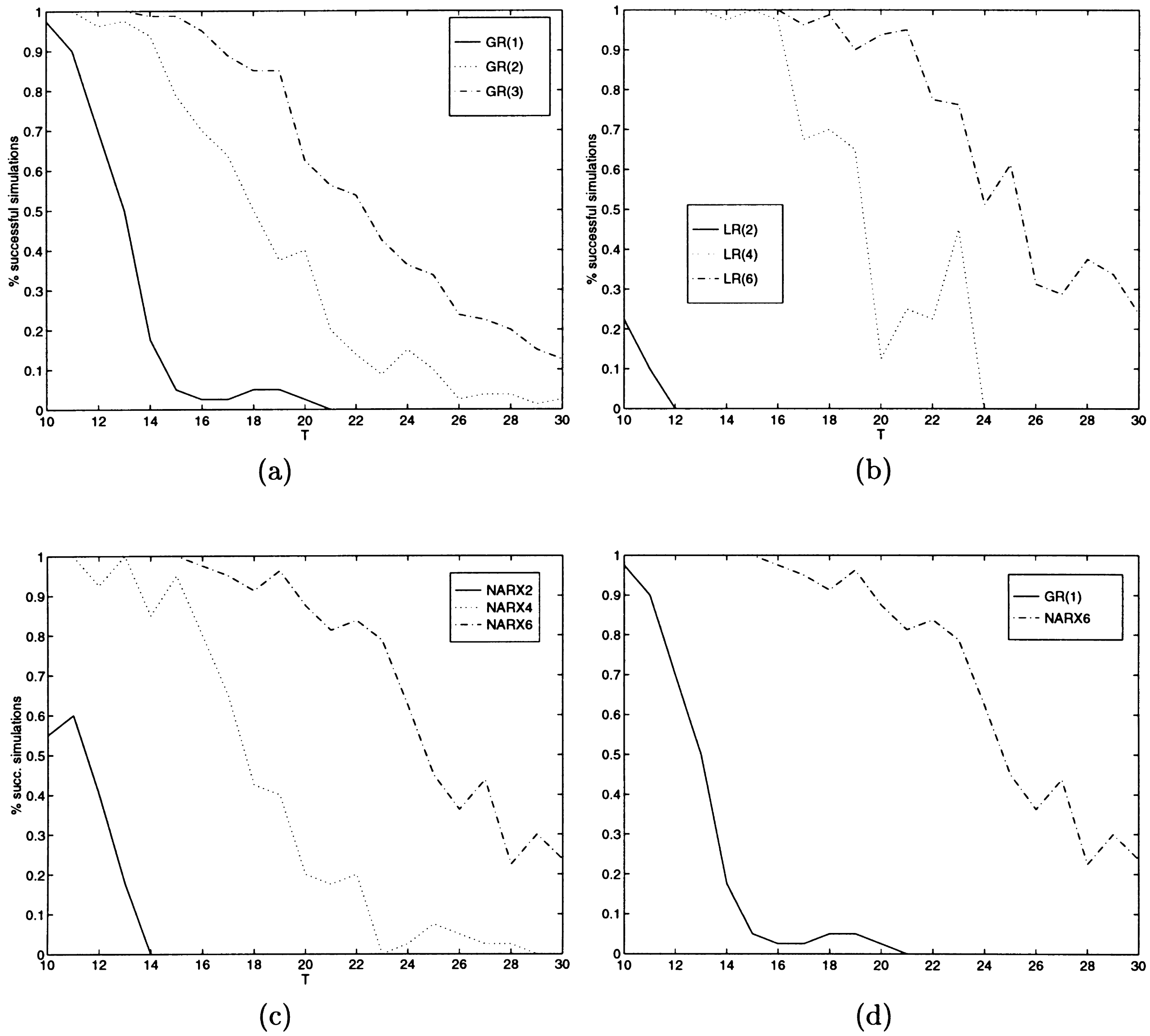


Fig. 7. Tree automata problem. Plots of percentage of successful simulations out of 80 as a function of *T*, the length of input strings, for different classes of networks with different orders of embedded memory: (a) globally connected RNN (GR); (b) locally connected RNN (LR); (c) NARX; (d) NARX vs GR.

Acknowledgements

We would like to thank L. Almeida, Y. Bengio, R. Salomon, H. Stone and the reviewers for useful suggestions.

References

Back A., & Tsoi A. (1991). FIR and IIR synapses, a new neural network architecture for time series modeling. *Neural Computation*, *3 (3)*, 337–350.

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

Chen S., Billings S., & Grant P. (1990). Non-linear system identification using neural networks. *International Journal of Control*, *51 (6)*, 1191–1214.

de Vries B., & Principe J.C. (1992). The gamma model—a new neural model for temporal processing. *Neural Networks*, *5*, 565–576.

Elman J. (1990). Finding structure in time. *Cognitive Science*, *14*, 179–211.

Frasconi P., & Gori M. (1996). Computational capabilities of local-feedback recurrent networks acting as finite-state machines. *IEEE Transactions on Neural Networks*, *7 (6)*, 1521–1524.

Frasconi P., Gori M., & Soda G. (1992). Local feedback multilayered networks. *Neural Computation*, *4*, 120–130.

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

Giles, C., Omlin, C., 1992. Inserting rules into recurrent neural networks. In: Kung, S., Fallside, F., Sorenson, J. A., Kamm, C. (Eds), Neural Network for Signal Processing II, Proceedings of the 1992 IEEE Workshop, Piscataway, NJ. IEEE Press, pp. 13–22.

Giles, C., Kuhn, G., Williams, R., 1994. Dynamic Recurrent Neural Networks: Theory and Applications. IEEE Transactions on Neural Networks, 5(2), Special Issue.

Gori, M., Maggini, M., Soda, G., 1994. Scheduling of modular architectures for inductive inference of regular grammars. In: ECAI'94 Workshop on Combining Symbolic and Connectionist Processing, Amsterdam. Wiley, Chichester, pp. 78–87.

Hihi, S. E., Bengio, Y., 1996. Hierarchical recurrent neural networks for long-term dependencies. In: Advances in Neural Information Processing Systems 8. MIT Press, Cambridge, MA.

Hochreiter, S., Schmidhuber, J., 1997. Long short-term memory. Neural Computation, 9, p. 1681.

Horne, B., Giles, C., 1995. An experimental comparison of recurrent neural networks. In: Advances in Neural Information Processing Systems 7. MIT Press, Cambridge, MA, pp. 697–704.

Jordan, M., 1986. Attractor dynamics and parallelism in a connectionist sequential machine. In: Proceedings of the Ninth Annual conference of the Cognitive Science Society. Lawrence Erlbaum, pp. 531–546.

Lin T., Horne B., Tino P., & Giles C. (1996). Learning long-term dependencies in narx recurrent neural networks. *IEEE Transactions on Neural Networks*, *7 (6)*, 1329–1338.

Lin, T., Horne, B., Tino, P., Giles, C., 1996b. Learning long-term dependencies is not as difficult with narx recurrent neural networks. In: Advances in Neural Information Processing Systems 8. MIT Press, Cambridge, MA.

Lin T., Giles C., Horne B., & Kung S. (1997). A delay damage model selection algorithm for narx neural networks. *IEEE Transactions on Signal Processing*, *45 (11)*, 2719–2730. (Special Issue on Neural Networks).

Ljung, L., 1987. System Identification: Theory for the User. Prentice-Hall, Englewood Cliffs, NJ.

Mozer, M.C., 1992. Induction of multiscale temporal structure. In: Moody, J., Hanson, S.J., Lippmann, R. (Eds.), Neural Information Processing Systems 4. Morgan Kaufmann, pp. 275-282.

Mozer, M.C., 1994. Neural net architectures for temporal sequence processing. In: Weigend, A., Gershenfeld, N. (Eds.), Time Series Prediction. Addison-Wesley, Reading, MA, pp. 243–264.

Narendra K., & Parthasarathy K. (1990). Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, *1 (1)*, 4–27.

Omlin C., & Giles C. (1996). Rule revision with recurrent neural networks. *IEEE Transactions on Knowledge and Data Engineering*, *8 (1)*, 183–188.

Pedersen, M., Hansen, L., 1995. Recurrent networks: second order properties and pruning. In: Tesauro, G., Touretzky, D., Leen, T. (Eds.), Advances in Neural Information Processing Systems 7. MIT Press, Cambridge, MA.

Sastry P., Santharam G., & Unnikrishnan K. (1994). Memory neuron networks for identification and control of dynamical systems. *IEEE Transactions on Neural Networks*, *5 (2)*, 306–319.

Seidl, D., Lorenz, R., 1991. A structure by which a recurrent neural network can approximate a nonlinear dynamic system. In: Proceedings of the International Joint Conference on Neural Networks 1991, Vol. II, pp. 709–714.

Siegelmann H., & Sontag E. (1995). On the computational power of neural nets. *Journal of Computer and System Sciences*, *50 (1)*, 132–150.

Siegelmann H., Horne B., & Giles C. (1997). Computational capabilities of recurrent narx neural networks. *IEEE Transactions on Systems, Man and Cybernetics—Part B*, *27 (2)*, 208.

Su, H.-T., McAvoy, T., 1991. Identification of chemical processes using recurrent networks. In: Proceedings of the American Controls Conference, Vol. 3, pp. 2314-2319.

Su H.-T., McAvoy T., & Werbos P. (1992). Long-term predictions of chemical processes using recurrent neural networks: a parallel training approach. *Industrial Engineering and Chemical Research*, *31*, 1338–1352.

Tsoi A., & Back A. (1994). Locally recurrent globally feedforward networks, a critical review of architectures. *IEEE Transactions on Neural Networks*, *5 (2)*, 229–239