How Embedded Memory in Recurrent Neural Network Architectures Helps Learning Long-term Temporal Dependencies *

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

[1] NEC Research Institute, 4 Independence Way, Princeton, NJ 08540
[2] Department of Electrical Engineering, Princeton University, Princeton, NJ 08540
[3] UMIACS, University of Maryland, College Park, MD 20742

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 architectures 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 to a specific recurrent neural network architecture because simply increasing the embedding memory order will make the architecture more robust to the problem of long-term dependency learning.

KEYWORDS:

discrete-time, memory, long-term dependencies, recurrent neural networks, training, gradient-descent

*Computer Science Technical Report CS-TR-3626 and UMIACS-TR-96-28, University of Maryland, College Park, MD 20742

1 Introduction

Recurrent Neural Networks (RNNs) are capable of representing arbitrary nonlinear dynamical systems [24] and can be computationally quite powerful [25]. 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* [2, 18], i.e. those problems for which the desired output of a system at time T depends on inputs presented at times $t \ll T$.

In particular Bengio *et al.* [2] 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 gradient*. Bengio *et al.* claimed that the problem of a vanishing gradient is the essential reason why gradient-descent methods are not sufficiently powerful to learn long-term dependencies.

Several approaches have been suggested to circumvent the problem of vanishing gradients in training RNNs. One possible approach is to preset initial weights by using prior knowledge [6, 9] but this is often not available in many applications. Another approach is to use alternative optimization methods instead of gradient-based methods [2]. 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 detectable [18, 22, 23]. Unfortunately, this approach may fail if short-term dependencies are equally as important. Hochreiter and Schmidhuber [12] propose a specific architectural approach which utilizes high-order gating units. Recently, it has been suggested that a network architecture that operates on multiple time scales might be useful [10, 11].

We have shown that a class of recurrent neural networks called NARX networks long-term dependencies when using a gradient descent training algorithm than previously reported in the literature [16, 15]. 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 [14]. In this paper, we empirically justify this hypothesis by showing the relationship between 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) [5]; locally recurrent networks (in particular the Frasconi, Gori and Soda's model) [7]; NARX networks [3, 20], 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. These 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.

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 [8]. One taxometric 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 states of the network are observable or not [13]. For another taxometric approach based on memory types, see Mozer [19]. For this study we picked three classes of networks: globally recurrent (GR) networks [5], locally recurrent networks (LR) [7], and NARX networks [3, 20]; 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 [19, 4]. NARX networks are a typical model of networks with observable states. GR networks are a popular class of network with globally connected hidden states, and LR networks belong to locally recurrent network architecture class also with hidden states.

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 Figure 1 (a). These hidden states are sometimes called *context units* in the literature. Suppose such a network with n_u input nodes, n_h hidden nodes of, and n_y output nodes, the dynamic equation can be described by:

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

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

,where $o(t)$ and $y(t)$ denotes 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:

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

Figure 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 connected together in a feed forward architecture [1, 7, 21, 28]. Specifically, we consider networks proposed by Frasconi *et al.* [7] (we will call LR), as shown in Figure 2 (a). The dynamic neurons of LR networks can be described by

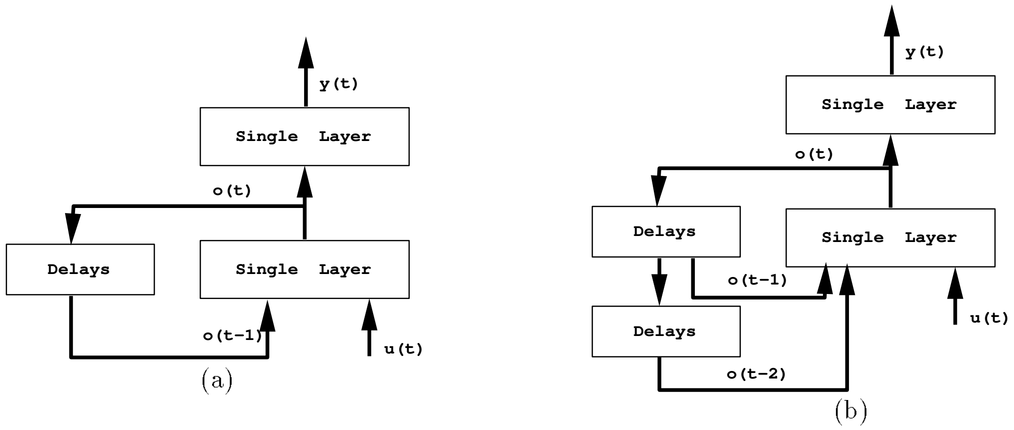


Figure 1: (a) A standard GR network. (b) A GR network with embedded memory of order two.

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

where $o_i(t)$ denotes the output of the i^{th} 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

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

Figure 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 [28] for a discussion of architectural differences.

2.3 NARX recurrent neural networks

An important class of discrete–time nonlinear systems is the *Nonlinear AutoRegressive with eXogenous inputs* (NARX) model [3, 17, 26, 27]:

$$y(t) = f\left(u(t-D_u), \ldots, u(t-1), u(t), y(t-D_y), \ldots, y(t-1)\right), \tag{6}$$

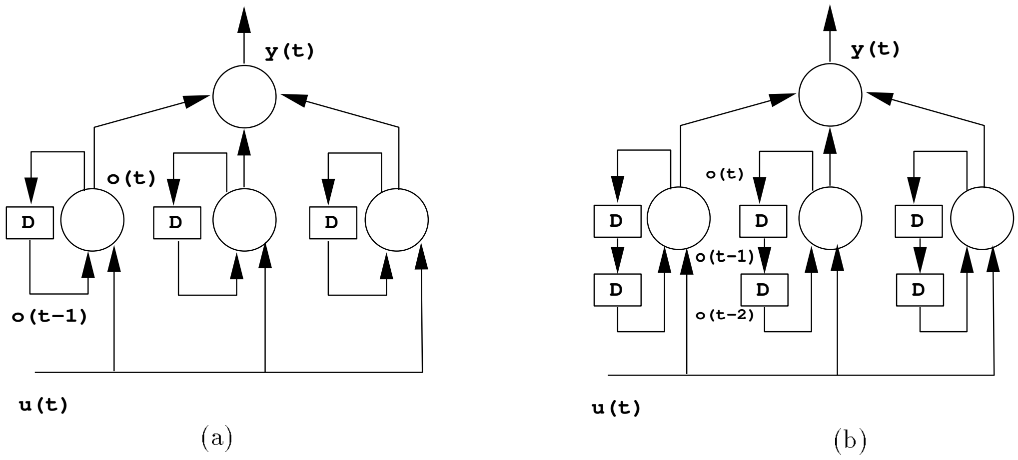


Figure 2: (a) A standard LR network. (b) A LR network with embedded memory of order two.

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. When the function f can be approximated by a Multilayer Perceptron, the resulting system is called a *NARX recurrent neural network* [3, 20].

In this paper, we shall consider NARX networks with zero input order and . Thus the operation of the network id defined by

$$y(t) = f\left(u(t), y(t - D_y), \ldots, y(t-1)\right), \tag{7}$$

Figure 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 typlify the long-term dependency issue. For more complex problems involving long-term dependencies see [12].

In order to establish some metric for comparison in the experimental results, we gave the recurrent

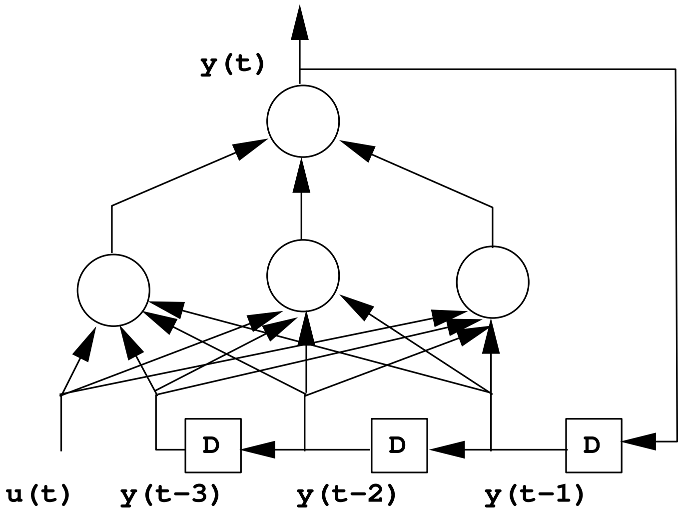


Figure 3: A NARX network with output memory of order 3.

Architecture	*Network Description*				*# weights*
	Memory order	# states	# 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 1: Architecture description of different recurrent networks used for the latching problem.

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 order of embedded memory on a problem already used for studying the difficulty in learning long-term dependencies [2, 11, 16].

This problem is a minimal task designed as a test that must necessarily be passed in order for a network to robustly latch information [2]. In this two-class problem, the class of a sequence depends only on the first 3 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 $t > 3$. 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 noise input $e(t)$ is given by

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

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 [2]. 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 dependencies. For our experiment, the input sequences were 1 and -1 and were one-hot encoded into two input neurons with trainable weights. The noise input weights were 0 until after 3 time steps, then 1. Figure 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 weight connected the noisy 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 to noise when $t > 3$. The ability of learning this minimal problem is a measure of the effectiveness of propagating the gradient for different neural network architectures with various memory orders.

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

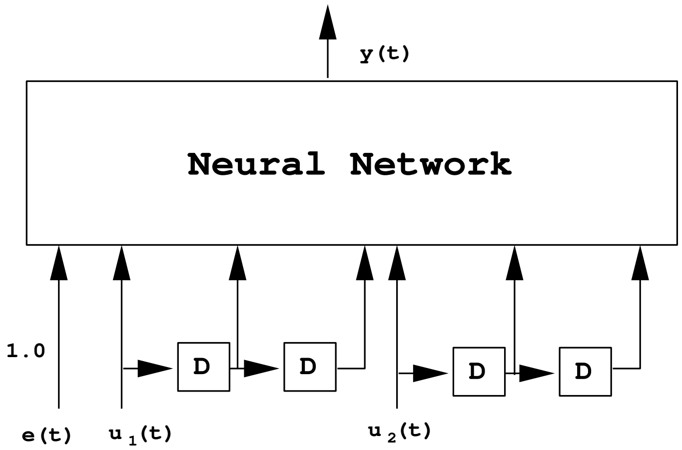


Figure 4: The network used for the latching problem.

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.

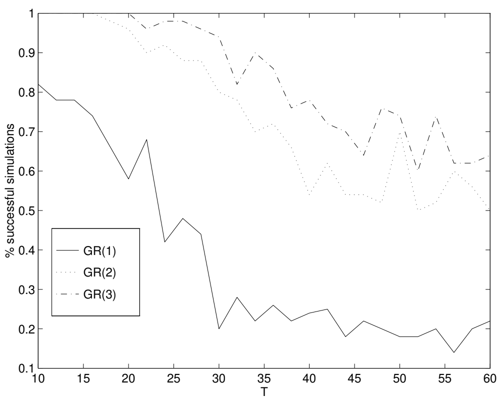
Figures 5 (a) to (c) show plots of the percentage of those runs that were successful for different classes of networks with different orders of embedded memory. It is clear from these plots that the network architectures with high order embedded memory become increasingly less sensitive to long–term dependencies as the memory order was increased.

An interesting comparison between the architectures GR(1) and NARX(6) is shown in Figure 5 (d). Since the two architectures have the exact same number of weights, hidden nodes, and states, the only difference is the amount of memory order. Clearly, NARX networks perform far better than the GR networks at learning the latching problem.

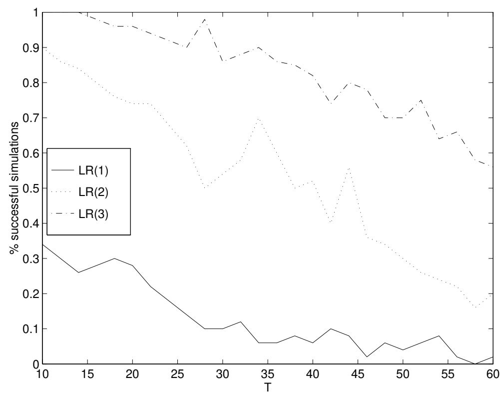
3.2 Grammatical Inference (Tree Automata) Problem

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 labelled according to some prespecified automata.

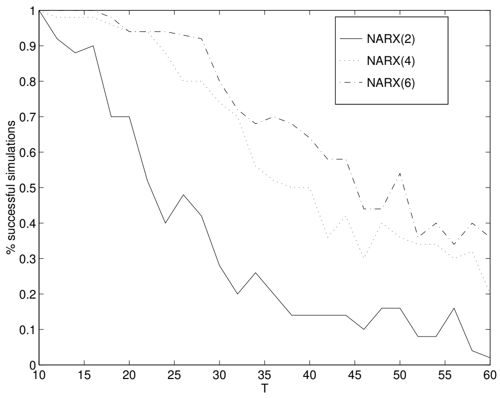
In this example, the class of a string is completely determined by its input symbol at some prespecified time t. For instance, Figure 6 shows a five-state automaton, in which the class of each string is determined by the third input symbol. When that symbol is "1", the string is accepted; otherwise,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



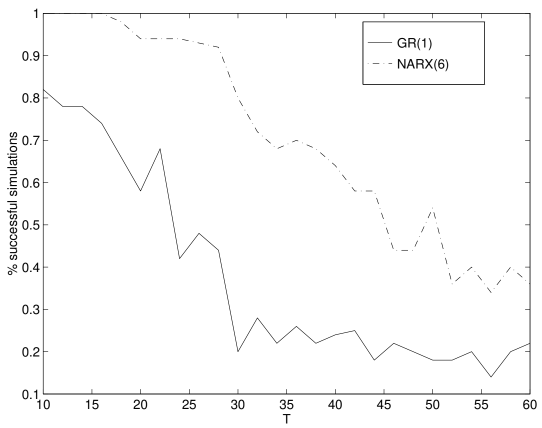
(a)



(b)



(c)



(d)

Figure 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 v.s. GR(1).

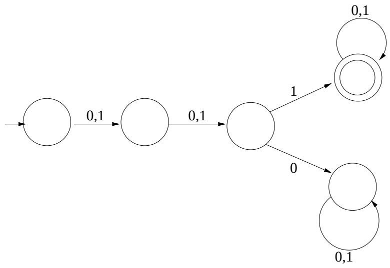


Figure 6: A five-state tree automaton. The unlabeled arrow is the start state and the double circled state is the the acceptance state.

Architecture	*Network Description*				*# weights*
	Memory order	# states	# 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	2	11 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

Table 2: Architectural description of different recurrent network architecture used for the tree automata problem.

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 2 alphabets encoded respectively as 0 and 1. 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

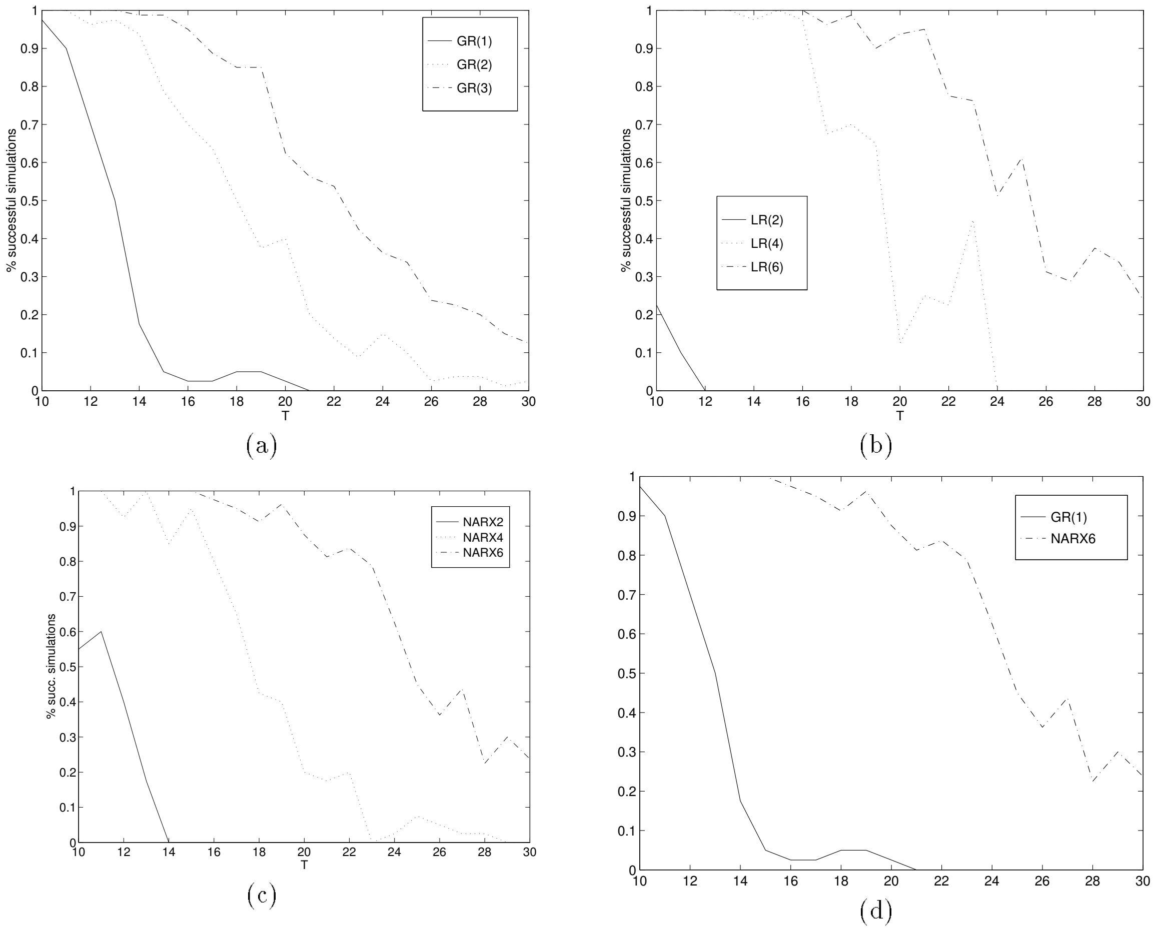


Figure 7: Tree Automata Problem. Plots of percentage of successful simulations out of 80 as a function of T, the length of input strings, for different classes of networks with different orders of embedded memory: (a) Globally connected RNN (GR), (b) Locally connected RNN (LR), (c) NARX, (d) NARX v.s. GR.

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.

Figures 7 (a) to (c) show plots of the percentage of the runs that were successful in each case. A comparison between NARX networks and GR networks was shown in Figure 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 paper, we explore the impact of embedded memory on various recurrent neural networks architectures for 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 [13], we explore 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-observerability: hidden state globally recurrent and locally recurrent networks, and observeable state NARX networks.

We tested this approach of extending memory in conventional recurrent neural networks on two simple long-term dependency problems. Our experimental results show that 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.

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 Backpropagation Through Time. These jump-ahead connections provide a shorter path for propagating gradient information, thus reducing the sensitivity of the network to long-term 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 gradient due to 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. Though specific architectures can be constructed for this problem, the approach of increasing memory order can be easily be applied to any recurrent architecture already in use of course at the cost of increased numbers of weights.

Acknowledgments

We would like to thank Jürgen Schmidhuber 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 modelling. *Neural Computation*, 3(3):337–350, 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] B. de Vries and J. C. Principe. The gamma model — A new neural model for temporal processing. *Neural Networks*, 5:565–576, 1992.

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

[6] 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.

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

[8] C.L. Giles, G.M. Kuhn, and R.J. Williams. Dynamic recurrent neural networks: Theory and applications. *IEEE Transactions on Neural Networks*, 5(2), 1994. Special Issue.

[9] C.L. Giles and C.W. Omlin. Inserting rules into recurrent neural networks. In S.Y. Kung, F. Fallside, J. Aa. Sorenson, and C.A. Kamm, editors, *Neural Networks for Signal Processing II, Proceedings of The 1992 IEEE Workshop*, pages 13–22, Piscataway, NJ, 1992. IEEE Press.

[10] 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.

[11] S. El Hihi and Y. Bengio. Hierarchical recurrent neural networks for long-term dependencies. In *Advances in Neural Information Processing Systems 8*. MIT Press, Cambridge, MA, 1996.

[12] S. Hochreiter and J. Schmidhuber. Long short term memory. Technical Report FKI-207-95, Fakultat fur Informatik, Technische Universitat Munchen, Munchen, 1995.

[13] B.G. Horne and C.L. Giles. An experimental comparison of recurrent neural networks. In *Advances in Neural Information Processing Systems 7*, pages 697–704. MIT Press, 1995.

[14] M. I. Jordan. Attractor dynamics and parallelism in a connectionist sequential machine. In *Proceedings of the Eighth Conference of the Cognitive Science Society*, pages 531–546. Erlbaum, 1986.

[15] Tsungnan Lin, B.G. Horne, P. Tino, and C.L. Giles. Learning long-term dependencies in narx recurrent neural networks. *IEEE Transactions on Neural Networks*. Accepted.

[16] Tsungnan Lin, B.G. Horne, P. Tino, and C.L. Giles. 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, 1996. In press.

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

[18] 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.

[19] Michael C. Mozer. Neural net architectures for temporal sequence processing. In A.S. Weigend and N.A. Gershenfeld, editors, *Time Series Prediction*, pages 243–264. Addison–Wesley, 1994.

[20] K.S. Narendra and K. Parthasarathy. Identification and control of dynamical systems using neural networks. *IEEE Trans. on Neural Networks*, 1(1):4, 1990.

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

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

[23] 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.

[24] D.R. Seidl and R.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 1991*, volume II, pages 709–714, July 1991.

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

[26] 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.

[27] 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.

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