TRTRL: A Localized Resource-Efficient Learning Algorithm for Recurrent Neural Networks

Danny Budik, *Student Member, IEEE,* Itamar Elhanany, *Senior Member, IEEE**

***Abstract*— This paper introduces an efficient, low-complexity online learning algorithm for recurrent neural networks. The approach is based on the real-time recurrent learning (RTRL) algorithm, whereby the sensitivity set of each neuron is reduced to weights associated either with its input or ouput links. As a consequence, storage requirements are reduced from $O(N^3)$ to $O(N^2)$ and the computational complexity is reduced to $O(N^2)$. Despite the radical reduction in resource requirements, it is shown through simulation results that the overall performance degradation is rather minor. Moreover, the scheme lends itself to parallel hardware realization by virtue of the localized property that is inherent to the approach.**

***Index Terms*— Recurrent neural networks, real-time recurrent learning (RTRL), constraint optimization.**

I. INTRODUCTION

Recurrent neural networks (RNNs) are widely acknowledged as an effective tool that can be used by a wide range of applications that store and process temporal sequences. The ability of RNNs to capture complex, nonlinear system dynamics has served as a driving motivation for their study. RNNs have the potential to be effectively used in a wide range of modeling, system identification and control applications, where other techniques may fall short. Consequently, a variety of learning algorithms have been proposed. Most learning algorithms rely on the calculation of error gradients with respect to the network weights. What distinguishes recurrent neural networks from static, or feedforward, networks is the fact that the gradients are time-dependent or dynamic. This implies that the current error gradient does not only depend on the current input, output and targets, but rather on its possibly infinite past. How to effectively train RNNs remains a challenging and an active research topic.

Practical constraints often guide the selection of one learning algorithm over another. The learning problem consists of adjusting the parameters (weights) of the network, so that the trajectories have certain specified properties. There are two primary classes of algorithms that have been proposed for computing dynamic gradients in RNNs: Backpropagation Through Time (BPTT) [1][2] and Real-Time Recurrent Learning (RTRL) [3][4][5]. Due to its inherent structure, BPTT must be used in batch mode. BPTT involves two phases. First, input stimuli are fed in the forward path through the network, thereby updating the internal states. Second, backpropagation is used to update the weights with respect to the performance error. The key advantage of BPTT, which can be viewed as an extension to the classical Elman recurrent network [6], is that the training algorithm is identical to those that are used for feedforward networks. However, BPTT has several fundamental drawbacks, two of which being the fact that it is not a real-time algorithm in the sense that batch data must be applied and the extensive memory requirements that are dictated by the need to store significant amounts of state information.

The second class of learning algorithms are variants of the RTRL algorithm that calculate gradients in real-time. The gradients at time k are obtained in terms of those at time instant $k-1$. Once the gradients are evaluated, weight updates can be calculated in a straightforward manner. These algorithms are very attractive in that they are applicable to real-time applications. However, the two main drawbacks of RTRL are the large computational complexity ($O(N^4)$) and, even more critical, the storage requirements of ($O(N^3)$), where N denotes the number of neurons (PEs) in the network. It is argued in this paper, that although the computational complexity has been at the center of the discussion in the literature, storage requirements are potentially the bottleneck for scalability.

Many methods have been proposed to reduce the computational complexity of RTRL. Hybrid BPTT/RTRL schemes have received attention in [7], reducing the complexity to $O(N^3)$. These methods take blocks of BPTT and use RTRL to encapsulate the history before the start of each block. Other methods, that similarly reduce computational complexity to $O(N^3)$, have proposed using Green's function [8].

In [4], the sensitivity set for each neuron is reduced to a subgroup of neurons, thereby decomposing the network into several non-overlapping sub-networks. The key advantage of subgrouping in this manner is the immediate reduction in computations to $O(N^4/g^3)$, where g denotes the number of groups. However, for a small number of subgroups, the advantage becomes negligible. If g is large, there is little crossover of training information from different groups, thereby significantly reducing the network's capabilities. The arbitrary selection of subgroups also appears somewhat weak. To address this concern, recent work has suggested dynamically partitioning the groups in accordance with gradient information being calculated online [5]. Although it constitutes a more intelligent and data-dependent approach, the method is not scalable due to the complex process of dynamically redefining the subgroup boundaries. Moreover, the key problems associated with the number of groups created in [4] remain.

*The authors are with the Electrical and Computer Engineering Department at the University of Tennessee (e-mails: dbudik@utk.edu, itamar@ieee.org). This work has been partially supported by the Department of Energy research contract DE-FG02-04ER25607 and by the Woodrow W. Everett, Jr. SCEEE Development Fund in cooperation with the Southeastern Association of Electrical Engineering Department Heads.

This paper proposes an alternative approach to reducing the resource requirements of online RNN learning. It focuses on reducing the sensitivities of each neuron to weights associated with its incoming and outgoing connections. The approach results in a localized algorithm that lends itself to hardware realization, while retaining the core capabilities associated with RTRL.

Section II provides a brief overview of the RTRL algorithm. In section III, the proposed algorithm is described and analyzed. Section IV presents simulation results, and our conclusions are drawn in section V.

II. Overview of RTRL

In this section we briefly describe the RTRL algorithm. Let us assume that a network consists of a set of N fully connected neurons and a set of M inputs. Let $w_{ij}(t)$ denote the weight (i.e. the synaptic strength) associated with the link originating from neuron j towards neuron i at time t. The net input to neuron k, $s_k(t)$, is defined as the weighted sum of all activations in the network, $z_l(t)$. Based on standard RTRL terminology, we define the activation function of node k at time $t+1$ to be

$$y_k(t+1) = f_k\left(s_k\left(t\right)\right), \tag{1}$$

where

$$s_k(t) = \sum_{l \in N \cup M} w_{kl} z_l(t), \tag{2}$$

$$z_k(t) = \begin{cases} x_k(t) & \text{if } k \in M \\ y_k(t) & \text{if } k \in N \end{cases} \tag{3}$$

and the non-linear activation function, $f(\cdot)$, maps to the range [0,1]. The overall network error at time t is defined by

$$J(t) = -\frac{1}{2} \sum_{k \in \text{outputs}} \left[y_k(t) - d_k(t)\right]^2 \tag{4}$$

$$= -\frac{1}{2} \sum_{k \in \text{outputs}} \left[e_k(t)\right]^2 \tag{5}$$

where $d_k(t)$ denotes the desired target value for output k at time t. Correspondingly, the error is minimized along a positive multiple of the performance measure gradient, such that

$$w_{ij}(t+1) = w_{ij}(t) + \alpha \frac{\partial J(t)}{\partial w_{ij}(t)}. \tag{6}$$

The online calculation of the gradients is achieved by exploiting the following relationship:

$$\frac{\partial J(t)}{\partial w_{ij}(t)} = \sum_{k \in \text{outputs}} e_k(t) \frac{\partial y_k(t)}{\partial w_{ij}(t)}. \tag{7}$$

By identifying the partial derivatives of the activation functions with respect to the weights as sensitivity elements, and denoting the notation by

$$p_{ij}^k(t+1) = \frac{\partial y_k(t+1)}{\partial w_{ij}(t)}, \tag{8}$$

we obtain the following recursive equation:

$$p_{ij}^k(t+1) = f_k'\left(s_k\left(t\right)\right) \left[\sum_{l \in N} w_{kl} p_{ij}^l(t) + \delta_{ik} z_j(t)\right], \tag{9}$$

where $p_{ij}^k(0) = 0$ and δ_{ik} is the Kronecker delta. Equations (9) and (7) allow one to obtain the performance gradient at any given time.

As can be seen from these equations, each neuron is required to perform $O(N^3)$ multiplications yielding an overall complexity of $O(N^4)$. Moreover, the storage requirements are dominated by the weights $O(N^3)$ and, more importantly, the sensitivity matrices, $p_{ij}^k(t)$, which are $O(N^3)$. Due to the distributed nature of the network, the calculation can be reduced to significantly by having each neuron compute its sensitivities in parallel. If performed in hardware, these computation processes can be accelerated by exploiting pipelining and module replication. However, unlike the computational requirements, the storage requirements cannot be reduced as they constitute a crucial component in the weight update procedure.

Several schemes that have been presented in the literature aim to reduce the storage complexity associated with RTRL. A unifying theme of these methods comprises of subgrouping the neurons into multiple, non-overlapping subnetworks. Although the computational gain is significant, the storage requirements remain high, in particular when a small set of subgroups is employed. We next describe the main contribution of this paper, which focuses on an alternative method for reducing the resource requirements in RTRL.

III. Truncated Real-Time Recurrent Learning (TRTRL)

As stated in earlier, the goal of TRTRL is to obtain a scalable version of the RTRL algorithm while minimizing performance degradation. TRTRL accomplishes this goal by reducing the amount of resources required for each PE. Let us begin with several key definitions that would guide us through the discussion:

Definition 1: Let I_j denote the set of nodes that have a direct link (and, hence, a unique associated weight) to node j. We shall refer to this set as the *ingress* set of node j.

Definition 2: Let E_j denote the set of nodes that node j has a link (and, hence, a unique associated weight) to. We shall refer to this set as the *egress* set of node j.

It should be observed that a node can reside within both ingress and egress sets of another node. Given that TRTRL limits the sensitivities of each neuron to the ingress and egress set, we have the following slightly-revised definition for $z_j(t)$,

$$z_j(t) = \begin{cases} x_j(t) & \text{if } j \in I \\ y_j(t) & \text{if } j \in I_j \end{cases}, \tag{10}$$

where I denotes the set of external inputs. By localizing the information required by each neuron, the calculation of equation 9 is constructed of three main parts. First, for all

nodes that are not in the output set, the egress sensitivity values for node j are calculated such that $j = k$ thereby yielding the following reduced expression:

$$p_{ij}^{j}(t+1) = f_j'(s_j(t)) \left[w_{ji} p_{ij}^{i}(t) + \delta_{ij} y_j(t) \right]. \quad (11)$$

Notice that the summation from equation 9 drops out because $p_{ik}^{l} = 0$ unless $l = i$.

The sensitivities pertaining to the ingress set for node j are calculated where $i = k$ and can be expressed as

$$p_{ij}^{i}(t+1) = f_i'(s_i(t)) \left[w_{ij} p_{ij}^{j}(t) + z_j(t) \right]. \quad (12)$$

From the above two expressions it becomes evident that the aggregate computational load for each neuron is in the order of $2N$. The full calculation of equation 9 is performed for input, bias and output units. Furthermore, the network remains fully recurrent in the sense that all neurons are connected (via unique weights) to all other neurons. The only difference between TRTRL and RTRL, in this context, is that neurons are limited in the sensitivities. To that end, TRTRL is completely local because neurons are no-longer required to fetch information that may be located at a remote part of the network. If the network is implemented in hardware, localizing the memory access is key to guaranteeing high-speed of execution.

In order to complete the description of TRTRL, we refer to the weight update rule given in (6). For the output neurons, a non-zero sensitivity element must exist in order to provide the performance gradient required by the weight update rule. To comply with this requirement, a direct link is added from each output neuron to each of the N neurons in the network. Therefore, each output neuron, o, computes N sensitivity updates (one for each neuron in the network) by performing the following:

$$p_{ij}^{o}(t+1) = f_o'(s_o(t)) \left[w_{oi} p_{ij}^{i}(t) + w_{oj} p_{ij}^{j}(t) + \delta_{io} z_j(t) \right]. \quad (13)$$

The recursive calculation of sensitivity elements for output neurons is reduced to the two left-most terms in the expression above since only two neurons (i and j) are sensitive to w_{ij}. This yields an overall computational complexity of $O(2KN)$, where K denotes the number of output neurons in the network. By reducing the calculations involved as shown above, the overall computational complexity is reduced from $O(N^4)$ to $O(KN^2)$. It should be noted that by restricting the calculation of the gradients in this manner yields a scalable network architecture that can be efficiently implemented in hardware.

IV. Simulation Results

We performed a comparison between the RTRL and TRTRL algorithms on several commonly employed testbenches that required the network to learn to configure itself in such a way that temporal dependencies are required, i.e. information that arrives at a given time has strong impact on the value of outputs at later times. In other words, the network is required to learn to represent useful state information internally so as to successfully accomplish these tasks.

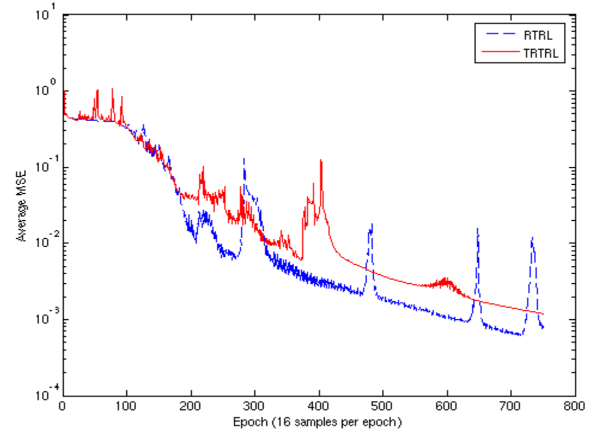

Fig. 1. Average learning curves for the frequency doubler simulation showing RTRL and TRTRL.

A. Frequency Doubler

The first task chosen was the frequency doubler system. For this task, the networks were required to produce a sinusoidal signal that has twice the frequency of the signal applied at its input. The latter is a sinusoid with a 16-sample period while the desired output signal is a sinusoid with an 8-sample period. This is a suitable basic task for the network as the input to output mapping is a nonlinear function.

For both RTRL and TRTRL, the network consisted of a single hidden layer with 15 fully recurrent neurons, one bias input neuron whose (constant) value is 1 and a single linear output neuron. The same set of random initial weights was applied to both networks each time they were trained.

Figure 1 below depicts the average learning curves for both algorithms over 10 runs. During the first 200 epochs, both algorithms train with the same rate but afterwards, the RTRL algorithm improves faster than TRTRL. As the epochs increase, however, the difference in the output accuracies become minor. This basic test case illustrates the inherent sufficiency of the gradient calculations in TRTRL.

B. Chaotic Time Series Prediction

The next task chosen was chaotic time series prediction, whereby the networks are required to predict future values of the Mackey-Glass (MG) [9] [10] chaotic series, which has been extensively used as a benchmark. The MG series is based on the time-delayed differential equations,

$$\frac{\partial x(t)}{\partial t} = -0.1x(t) + \frac{0.2x(t-\tau)}{1 + x^{10}(t-\tau)}. \quad (14)$$

To obtain values at integer time points, the fourth-order Runge-Kutta method was used to find the numerical solution to the above MG equation. Here, we assume that the time step is 1, $x(0) = 0.1$, $t = 17$ and $x(t) = 0$ for $t < 0$.

The task is to predict the value of $x(t+30)$ given the current input and internal state representation. The chaotic time series prediction task was chosen because it is significantly more difficult for the networks to solve than the frequency doubler.

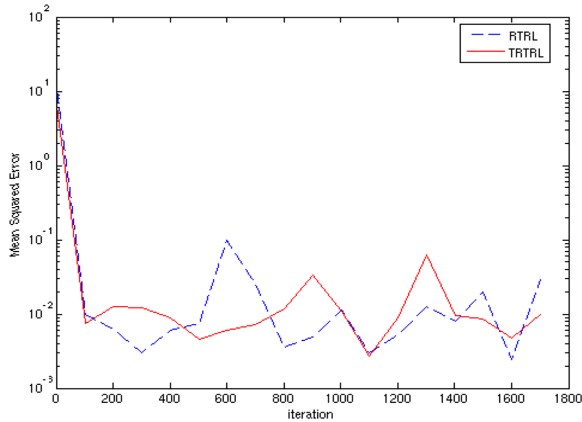

Fig. 2. Learning curves for the chaotic time series prediction task, applied to both RTRL and TRTRL.

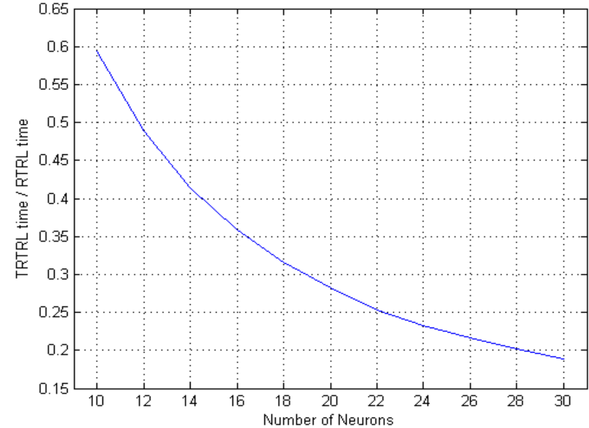

Fig. 3. The ratio between TRTRL and RTRL training times for the Mackey-Glass (MG) chaotic time series prediction task.

The network topology was identical to the one used for the frequency doubler task. Both networks were constructed using 25 fully recurrent neurons, one bias input neuron whose (constant) value is 1 and a single linear output neuron. The same set of random initial weights was used for each network during every training run. An adaptive learning rate algorithm with a momentum term was used to improve the weight update rule of each algorithm [11].

Figure 2 illustrates the learning curves for both algorithms. After completing 1800 time steps of training, both networks learned to predict the output 30 time steps ahead of the input with a similar degree of accuracy. This simulation task proved difficult for both networks as the error did not drop below 10^{-3}. Once again, however, the outcome of this simulation indicates that TRTRL is a valid method for performing temporal processing tasks.

A significant benefit to TRTRL is the improvement in computation time. Because the number of calculations of the sensitivities is reduced, the computational speedup gain is by two orders of magnitude from $O(N^4)$ in RTRL to $O(N^2)$ with TRTRL. This speedup gain is reflected by the processing time ratio shown in figure 3. The time to train the networks sing the chaotic time series task was recorded as the number of neurons was increased. As one would expect, the speedup gain for TRTRL increases for larger networks.

V. Conclusions

In this paper, we presented a novel approach for reducing the resource requirements of RTRL to $O(N^2)$ while retaining high-performance. The method is based on limiting the sensitivities for each neuron to weights associated with either incoming or outgoing links. Based on standard testbench scenarios, it is demonstrated that the performance degradation is kept low while speed and storage requirements are significantly reduced. Moreover, the resulting network architecture lends itself to hardware implementations as it is highly localized.

References

[1] R. J. Williams and J. Peng, "An efficient gradient-based algorithm for on-line training of recurrent network trajectories," *Neural Computation*, vol. 2, pp. 490–501, 1990.

[2] P. Werbos, "Backpropagation through time: what it does and how to do it," in *Special issue on neural networks, Proceedings of IEEE*, vol. 78, pp. 1550–1560, October 1990.

[3] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural Computation*, no. 1, pp. 270–280, 1989.

[4] D. Zipser, "A subgrouping strategy that reduces complexity and speeds up learning in recurrent networks," *Neural Computation*, no. 1, pp. 552–558, 1989.

[5] N. Euliano and J. Principe, "Dynamic subgrouping in RTRL provides a faster o(n2) algorithm," in *Acoustics, Speech, and Signal Processing, 2000. ICASSP '00. Proceedings. 2000 IEEE International Conference on*, vol. 6, (Istanbul), pp. 3418–3421, 2000.

[6] J. L. Elman, "Finding structure in time," *Cognitive Science*, no. 14, pp. 179–211, 1990.

[7] J. Schmidhuber, "A fixed size storage o(n3) time complexity learning algorithm for fully recurrent continually running networks," *Neural Computation*, vol. 4, pp. 243–248, 1992.

[8] G.-Z. Sun, H.-H. Chen, and Y.-C. Lee, "Green's function method for fast on-line learning algorithm of recurrent neural networks.," in *NIPS*, pp. 333–340, 1991.

[9] M. Mackey and L. Glass, "Oscillation and chaos in physiological control systems," *Science*, vol. 197, pp. 287–289, July 1977.

[10] R. C. III, "Predicting the mackey-glass timeseries with cascade-correlation learning," in *D. Touretzky, G. Hinton and T. Sejnowski eds., Connectionist Models Summer School Proceedings*, pp. 117–123, 1990. Carnegie Mellon University.

[11] D. S. V.P. Plagianakos and M. Vrahatis, "An improved backpropagation method with adaptive learning rate," in *2nd International Conference on Circuits, Systems, and Computers*, 1998. Piraeus, Greece.