Improvement of Practical Recurrent Learning Method and Application to a Pattern Classification Task

Mohamad Faizal bin Samsudin and Katsunari Shibata

Department of Electrical and Electronic Engineering
Oita university, 700 Dannnoharu, Oita 870-1192 Japan
shibata@cc.oita-u.ac.jp

Abstract. Practical Recurrent Learning (PRL) has been proposed as a simple learning algorithm for recurrent neural networks[1][2]. This algorithm enables learning with practical order $O(n^2)$ of memory capacity and computational cost, which cannot be realized by conventional Back Propagation Through Time (BPTT) or Real Time Recurrent Learning (RTRL). It was shown in the previous work[1] that 3-bit parity problem could be learned successfully by PRL, but the learning performance was quite inferior to BPTT. In this paper, a simple calculation is introduced to prevent monotonous oscillations from being biased to the saturation range of the sigmoid function during learning. It is shown that the learning performance of the PRL method can be improved in the 3-bit parity problem. Finally, this improved PRL is applied to a scanned digit pattern classification task for which the results are inferior but comparable to the conventional BPTT.

1 Introduction

The significance of recurrent neural networks (RNNs) is expected to grow more and more hereafter for developing higher functions due to its ability of purposive learning to memorize information or events that have occurred earlier in a sequence. Currently, there are two popular learning algorithms for recurrent neural networks, BPTT[3] and RTRL[4], that have been widely used in many application areas. However, the critical drawback of the conventional algorithms is that they suffer from the necessity of large memory capacity and computational cost.

BPTT requires $O(n^2T)$, order of computational cost and $O(nT)$, order of memory capacity where n is the number of nodes and T is the number of steps for tracing back to the past. That means the past T states of the neural network have to be stored and the learning is done by using them. However, if T is small, it is worried that sufficient learning according to the past state cannot be done. On the other hand, RTRL needs as large as $O(n^3)$ for memory capacity, and $O(n^4)$ for computational cost, in order to modify each connection weight without tracing back to the past. Using RTRL in a large scale network is impractical

M. Köppen et al. (Eds.): ICONIP 2008, Part II, LNCS 5507, pp. 631–638, 2009.

because of the explosion in necessary memory capacity and computational cost. So far, many researches such as [5] have been done based on BPTT and RTRL.

S. Hochreiter and J. Schmidhuber[6] have proposed a special network architecture that has some memory cells that enables constant, non-vanishing error flow within the memory cell. They used a variant RTRL and only O(n2) of computational cost is required. However, special structure is necessary and it cannot be applied to the general recurrent neural networks.

Therefore, clearly, a practical learning algorithm for recurrent neural networks that need $O(n^2)$ or less memory and $O(n^2)$ or less computational cost is required where n^2 is equivalent to the number of synapses. PRL is an algorithm that capable to keep the order of memory size and computational cost as low as $O(n^2)$, by introducing some variable to hold some past states which enables constant memory and local computation to be assigned at each synapse. Therefore, this does not only reduce the memory capacity and computational cost drastically, but also increases the feasibility as a hardware system. In the previous work, it was shown that benchmark problems (sequential EXOR and 3-bit parity problem) could be learned successfully by PRL even though the learning performance was often quite inferior to the conventional BPTT.[1]

This paper presents an extension of the PRL method. The target is to make PRL perform equivalently to or outperform the conventional methods, considering that PRL already excels in computational cost and memory size. The extension is made by adjusting hidden nodes' output to prevent monotonous and biased oscillation during learning as will be described in the next section. Finally, we apply this extended PRL method to a more difficult task and compare it to BPTT.

This paper is organized as follows. In section 2, the extended of PRL method in the discrete time domain is introduced. Section 3 presents the improved result for 3-bit parity and the application to a pattern recognition task. Section 4 presents the conclusion of this paper.

2 Practical Recurrent Learning (PRL)

This section briefly recounts the PRL method in the discrete time domain as proposed in [1] at first. The forward calculation is the same as a regular neural network[3] in which each node computes weighted sum of its inputs and nonlinear transformation by sigmoid function. The basic idea is, some variables that are allocated to each synapse and hold the past information are introduced, considering the relationship between the outputs of post and pre-synaptic nodes. The connection weights between the nodes are modified by using the variables and propagated error signal. In order to keep the memory size and calculation time small, the error is propagated backwards like conventional BP[3] without tracing back to the past. In the past work[2], in the continuous time domain,

[1] Comparison to the RTRL is not shown in this paper, considering that RTRL is less practical than BPTT in terms of memory capacity and computational cost in larger networks.

three kinds of variables to hold the following items of the past information were introduced intuitively based on trials and errors.

1. the latest outputs of pre-synaptic nodes,
2. the outputs of pre-synaptic nodes that change recently among all the inputs to the post-synaptic node,
3. the outputs of the pre-synaptic node that caused the changed in the post-synaptic node's output.

These information are held by some variables named p_{ji}, q_{ji} and r_{ji} respectively. However, even a sequential EXOR problem could not be learned.

Then, for easy analysis, the algorithm is converted into the discrete time domain and only variable r_{ji} was used in the previous work[1]. Among the three variables, r_{ji} is particularly important because r_{ji} does not change when the output of post synaptic node does not change and is useful for the problems that need to memorize some past information before a long time lag. r_{ji} in the discrete time domain is updated at each time step as

$$r_{ji,t} = r_{ji,t-1}(1 - |\Delta x_{j,t}|) + f'(S_{j,t})x_{i,t}|\Delta x_{j,t}| \tag{1}$$

where $f'(S_j)$ is the derivate of the sigmoid function of j-th post-synaptic node, x_i, x_j is the output of the pre- and post-synaptic node respectively and $\Delta x_{j,t}= x_{j,t} - x_{j,t-1}$.

The important feature of r_{ji} is that, it holds the information about the output of the pre-synaptic node that caused the change of the pre-synaptic node's output. Each synaptic weight is updated as

$$\triangle\ w_{ji} = \eta\delta_j r_{ji} \tag{2}$$

where η is a learning rate and δ_j is propagated error of the post synaptic nodes.

2.1 Improvement of the PRL Method

Prevention of monotonous and biased oscillation in hidden nodes. By observing and analyzing the result from the previous work[1], it is shown that there is some monotonous oscillation in the change of hidden nodes' output for PRL during the learning phase. Fig.1 shows the change of the hidden node's output during the learning for BPTT and PRL whose connection weight to the output node is the largest. Almost half of the nodes' outputs in the hidden layer for PRL oscillate monotonously in some biased range of value. The net value of this output lies around the saturation range of the sigmoid function. It is well known that when the net value lies around the saturation range, the learning does not progress and it largely affects the learning performance.

Then, in order to accelerate learning, the output of hidden nodes is adjusted by moving the value to the vicinity of 0 when the output of the node oscillates around the saturation range of the sigmoid function. At first, the temporal average of the output is calculated in each epoch by

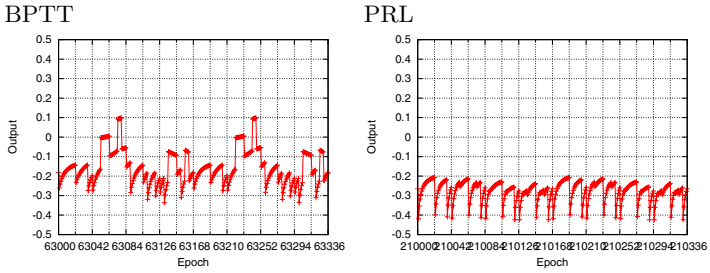

Fig. 1. The change of the output of the hidden node who has the largest connection weight to the output node for both methods. The left side is for BPTT, and the right one is for PRL.

$$\bar{o_j} = \frac{\sum_{\tau=0}^{T} o_{j,\tau}}{T}, \tag{3}$$

where $\overline{o_j}$ is the average of hidden jth output, T indicates the number of time steps for one epoch. Then the value of $\overline{o_{j,n}}$ is compared to average of $\overline{o_{j,n-1}}$ in the previous epoch according to the following equation.

$$\triangle\, o_{j,n} = \overline{o_{j,n}} - \overline{o_{j,n-1}}. \tag{4}$$

Then, if the difference of average value $\mid \triangle\, o_j \mid$ was below 0.1 and the state continued for 8 epochs, the hidden node's output is adjusted to the vicinity of 0 by the following equation before starting the next epoch. Here, we used the sigmoid function whose value range from -0.5 to 0.5.

$$o_{j,t} = o_{j,t} - \overline{o_{j,t-1}}. \tag{5}$$

3 Experimental Results

In this section, two different experiment results are presented to show the performance of the proposed extension PRL. The first experiment is the 3-bit parity problem as a benchmark test to show that modification of oscillated hidden neurons' output could improve the learning performance. The second experiment is a pattern classification task which is used to test whether PRL can perform in a more practical task.

Here, the network architecture used in this paper is an Elman-type RNN. Conventional BP method is used for the learning between hidden and output layers, and PRL is used for the learning between input and hidden layers.

3.1 3-Bit Parity Problem

In the preceding work[1], it was shown that 3-bit parity problem could be learned by PRL, but the learning performance is quite inferior to the BPTT method. RNN with 1 output, 20 hidden units and 4 input signals is used here. 3 of the

inputs are the input signals to calculate the parity, and is given at t=5, 10, 15 sequentially. The other one is given to distinguish the starting time of one epoch and it is always 1 at t=0.

Table.1 summarizes some improving results of the PRL method. Here, successful learning is clarified when a squared error of less than 10^{-3} is continues for 100 patterns. In terms of success ratio, it is shown that PRL can perform better than before and similar to BPTT even though no trace back is done in this method. Although conventional BPTT performs better, in PRL, the average success iteration can be improved more than 50% compared to before.

In addition, Fig.2 shows the change of hidden node's output that have been adjusted to the vicinity of 0, resulting in faster learning than before.

Table 1. Comparison results of learning success and average success iterations

Learning rate of variable r	Learning success (/100times)	Average success iteration
Before modification		
1	99	33,107
2	85	29,737
4	42	33,666
10	7	17,331
After modification		
1	100	24,703
2	98	19,576
4	100	15,199
10	95	11,628
BPTT	100	6,297

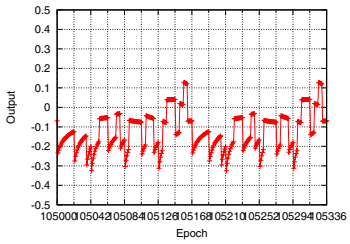

Fig. 2. The change of hidden node's output after introducing the method to move the value to the vicinity of 0

3.2 Pattern Classification Task

A critical test of the presented algorithm is to directly deal with high-dimensional, multimedia data, such as images or speech. Here, we carry out a handwritten digit recognition to investigate the performance of the proposed PRL.

The experiment was conducted on a digit database whose samples were collected by using a pen tablet as shown in Fig.3. Each of 10 different people wrote

each of 10 numbers from 0 to 9 twice. Thus, 200 images were collected for training in total. In addition, we added 3 sample sets from 3 other people for test data as shown in Fig.3 to observe the generalization ability of both method. Each image has 100 rows and 100 columns, and each of 10,000 pixels has binary value. Considering the introduction of continuous-value inputs and effective generalization, the size of the image was reduced to 20 × 20 pixels by calculating the average of every 5 × 5 pixels. Then, in order to enter this digit image signals into a recurrent neural network, it is scanned column by column, resulting in 20 inputs per each step as shown in Fig.4. In addition, the number of steps is set to 20 in one iteration to represent 20 rows, and the training signal is provided only at t=20.

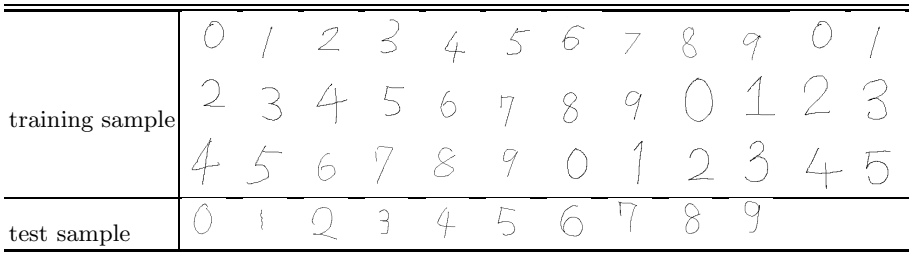

Fig. 3. Examples of handwritten 09 digit numbers

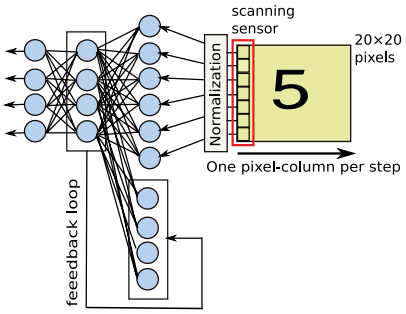

Fig. 4. A recurrent neural network with a handwritten digit image input by scanning column by column in each step

The task here is to classify the digit images into 10 classes. For instance, if an images of the number '1' is set as training data, the training signal for the 1st node in the output layer is 0.4, while the others will be -0.4. Furthermore, Table 2 shows the other parameter setup of the task.

Table 3 summarizes the comparison results for both methods. Here, the condition of successful learning is that the corresponding output to the presented image is the maximum among all the output nodes for every presented image. From the results, it is shown that the improved PRL could work even

Table 2. Parameter setups

Nodes in input layer	20 + hidden's layer nodes
Nodes in output layer	10
Nodes in hidden layer	40
Range of sigmoid function	-0.5∼0.5
Initial weight of (self-feedback)	4.0
Initial weight (non-self feed-back)	0.0
Initial weight (input to hidden layer)	random number of -1.0∼1.0
Initial weight (hidden to output layer)	0.0

Table 3. Comparison results of learning success ratio and average success iterations in the handwritten digit classification problem

Method	Learning success (/10 times)	Average success iteration
PRL	10	25,973
BPTT	10	14,666

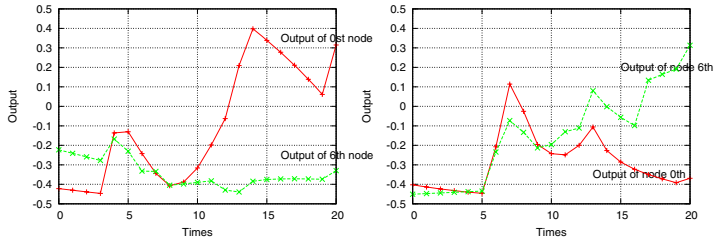

Fig. 5. The right side is the change in the outputs of $0th$ node and $6th$ node for an image of pattern '0' and the left side is the change for the image of pattern '6'

in the hand-writing recognition task with almost continuous-value inputs that is more difficult compared to the parity problem. It is also shown here that in terms of success ratio, PRL can perform almost as good as BPTT, but conventional BPTT still performs better in the number of average success iteration.

In order to show how the RNN classifies the images, two samples output changes in one epoch are shown in Fig.5 for presence of '0' and '6'. The left half of '0' and '6' is similar to each other. It can be seen that the change of output in $0th$ and $6th$ node is similar at the early times, but the corresponding output is increases after the latter half of the epoch.

In order to observe the generalization ability of both method, the performance for the test data is examined. It is shown that all of the test data could be

classified by the PRL method while 15 from 30 images in BPTT. PRL could recognize all the images in test samples, though further experiments are required to validate the results of generalization ability between both methods.

4 Conclusion

One extension are introduced here in order to improve performance of the PRL method that capable to keep the order of memory size and computational cost as low as $O(n^2)$, which are not realized by BPTT and RTRL. By preventing the oscillated hidden node's output from being biased to the saturation of the sigmoid function, it was shown that the learning performance of PRL in 3-bit parity problem could be improved. Furthermore, it was shown that PRL works in scanned digit pattern classification task that is more practical than the parity problem. However, since PRL is still inferior to BPTT in the average number of iteration for learning, future investigations for further improvement and application to more practical tasks are required.

Acknowledgement. A part of this research was supported by JSPS Grant in-Aid for Scientific Research and ♯19300070

References

1. Samsudin, M.F., Hirose, T., Shibata, K.: Practical Recurrent Learning (PRL) in the Discrete Time Domain. In: Ishikawa, M., Doya, K., Miyamoto, H., Yamakawa, T. (eds.) ICONIP 2007, Part I. LNCS, vol. 4984, pp. 228–237. Springer, Heidelberg (2008)
2. Shibata, K., Okabe, Y., Ito, K.: Simple Learning Algorithm for Recurrent Networks to Realize Short-Term Memories. In: Proc. of IJCNN Intl. Joint Conf. on Neural Network, vol. 98, pp. 2367–2372 (1998)
3. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning internal representations by errorpropagating. In: Parallel Distributed Processing, vol. 1, pp. 318–362. MIT Press, Cambridge (1986)
4. Williams, R.J., Zipser, D.: A learning algorithm for continually running fully recurrent neural network. Neural Computation 1, 270–280 (1989)
5. Song, Q., Wu, Y., Soh, Y.C.: Robust Adaptive Gradient-Descent Training Algorithm for Recurrent Neural Networks in Discrete Time Domain. IEEE Transactions on Neural Networks 19(11), 1841–1853 (2008)
6. Hochreiter, S., Schimidhuber, J.: Long Short Term Memory. Neural Computation 9, 1735–1780 (1997)