

# Application of A Fast Real Time Recurrent Learning Algorithm to Text-to-Phoneme Conversion \*

Yee-Ling LU

Man-Wai MAK

Wan-Chi SIU

Department of Electronic Engineering  
The Hong Kong Polytechnic University  
Hong Kong

E-mail: mwmak@encserver.en.polyu.edu.hk

## ABSTRACT

This paper attempts to perform text-to-phoneme conversion by using recurrent neural networks trained with the real time recurrent learning (RTRL) algorithm. As recurrent neural networks deal well with spatial temporal problems, they are proposed to tackle the problem of converting English text streams into their corresponding phonetic transcriptions. We found that, due to the high computational complexity, the original RTRL algorithm takes a long time to finish the learning. We propose a fast RTRL algorithm (FRTRL), with a lower computational complexity, to shorten the time consumed in the learning process.

## 1. Introduction

The real time recurrent learning (RTRL) algorithm is a gradient-following learning algorithm, derived by William and Zipser [1], for fully connected recurrent neural networks. In RTRL networks<sup>1</sup>, the signals emerging from each processing node of the output layer are fed back to every processing node. Thus, the state of the outputs at the current time step is determined by the previous output states and the current inputs. The feedback paths and the states enable the network to encode the temporal relationships of the input sequences. Researches have shown that RTRL networks are good predictors [2-3], and they are suitable for solving spatio-temporal problems [4]. Therefore, this paper proposes an application of RTRL networks to text-to-phoneme conversion.

Text-to-phoneme conversion is a preliminary step in text-to-speech synthesis [5]. It plays an important role in affecting the degree of naturalness and understanding of synthetic speech. In text-to-phoneme conversion, English words are converted into phonetic transcriptions which are exact representations of pronunciation. There are several attempts [6-8] that apply neural networks to convert text into phonetic transcriptions. For instance, in the NETtalk system [6], the assignment of the phonemes partly depends on the relationships between the target letter and its nearby letters. That is, the spatial and temporal information within the in-

coming text sequences are important factors that should be considered. Therefore, this paper proposes to use RTRL networks for text-to-phoneme conversion.

However, the computational complexity of the RTRL algorithm is  $O(n^4)$ , where  $n$  is the number of processing nodes in the output layer of the RTRL network. This feature of the algorithm implies a high computational burden in the training phase of the algorithm, especially when the RTRL network scales up. Several techniques [4, 9, 10] have been published for improving the rate of convergence of the RTRL algorithm. These techniques include dividing the RTRL network into a number of subnetworks [9], changing the error function [4], and modifying the training strategy [10]. However, our proposed technique, FRTRL algorithm, aims at speeding up the training phase as the network scales up while maintaining the feature (fully connected) of RTRL networks. This is implemented by confining the error backpropagation to some randomly selected connections that link between the input and the output layers or within the output layer.

## 2. Theoretical model of the FRTRL Algorithm

The main objective of the RTRL algorithm is to minimize the error function,

$$\begin{aligned} \mathcal{F}(t) &= \frac{1}{2} \sum_{k \in T(t)} [e_k(t)]^2 \\ &= \frac{1}{2} \sum_{k \in T(t)} [d_k(t) - y_k(t)]^2, \end{aligned} \quad (1)$$

\* This work was supported by the Hong Kong Polytechnic University Grant A/C No. 350/256.

<sup>1</sup>The term RTRL network denotes the fully connected recurrent networks trained with the RTRL algorithm.

through updating the weight matrix  $W$ . In Equation 1,  $\mathcal{F}(t)$  denotes the network error at time  $t$ . Moreover,  $e_k(t)$  represents the error between the desired output  $d_k(t)$  and the actual output  $y_k(t)$  at time  $t$ , where  $k$  belongs to the set  $T(t)$  of processing nodes with teaching status, at time  $t$ . The weight matrix  $W$  consists of two sets of weights. Where the weights represent the strength of the connections between the inputs and processing nodes, we denote them as inter-weights (i.e.,  $w_{ij} \forall i \in U$  and  $j \in I$ , where  $U$  and  $I$  are the set of processing and input nodes, respectively). On the other hand, if the weights represent the strength of the recurrent paths, we denote them as intra-weights (i.e.,  $w'_{ij} \forall i, j \in U$ ). For each learning cycle, the small change of each element in the weight matrix is calculated by

$$\Delta w_{ij}(t) = -\alpha \frac{\partial \mathcal{F}(t)}{\partial w_{ij}} = \alpha \sum_{k \in T(t)} e_k(t) \frac{\partial y_k(t)}{\partial w_{ij}} \quad \forall i \in U, j \in I, \quad (2)$$

and

$$\Delta w'_{ij}(t) = -\alpha \frac{\partial \mathcal{F}(t)}{\partial w'_{ij}} = \alpha \sum_{k \in T(t)} e_k(t) \frac{\partial y_k(t)}{\partial w'_{ij}} \quad \forall i, j \in U, \quad (3)$$

where  $\alpha$  is the learning rate. Then, the weight matrix is updated by

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t) \quad \forall i \in U, j \in I, \quad (4)$$

and

$$w'_{ij}(t+1) = w'_{ij}(t) + \Delta w'_{ij}(t) \quad \forall i, j \in U. \quad (5)$$

Referring to the mathematical analysis of the RTRL algorithm [1], the only difference between the RTRL and the FRTRL algorithms is the updating procedure of the learning sensitivity terms:  $P_{ij}^k$  and  $P'_{ij}^k$ . These sensitivity terms describe the influences on the output of the processing nodes through the change of the weight matrix. Thus,  $P_{ij}^k(t)$  ( $= \frac{\partial y_k(t)}{\partial w_{ij}}$ ) and  $P'_{ij}^k(t)$  ( $= \frac{\partial y_k(t)}{\partial w'_{ij}}$ ) represent the learning sensitivities of the processing nodes with respect to the change in inter-weights and intra-weights, respectively, at time  $t$ .

In the original algorithm,  $P_{ij}^k$  and  $P'_{ij}^k$  are updated by

$$P_{ij}^k(t+1) = f'[S_k(t)] \left[ \sum_{l \in U} w'_{kl}(t) P_{ij}^l(t) + \delta_{ik} x_j(t) \right] \quad \forall i \in U, j \in I, \text{ and } k \in U, \quad (6)$$

and

$$P'_{ij}^k(t+1) = f'[S_k(t)] \left[ \sum_{l \in U} w'_{kl}(t) P'_{ij}^l(t) + \delta_{ik} y_j(t) \right] \quad \forall i, j, k \in U, \quad (7)$$

where

$$S_k(t) = \sum_{l \in U} w'_{kl} y_l(t) + \sum_{l \in I} w_{kl} x_l(t) \quad \forall k \in U, \quad (8)$$

$\delta_{ik}$  is the Kronecker delta, and  $f'[\cdot]$  denotes the derivative of the sigmoid function  $f[\cdot]$ . These updating processes dominate the total number of computations in each learning cycle. However, in the FRTRL algorithm, some of the elements in the intra-weight matrix ( $w'_{ij}$  with  $i \neq j$ ) are fixed at an initial value. Moreover, throughout the learning phase, these elements will not be updated. Therefore, we have

$$\Delta w'_{ij}(t) = \alpha \sum_{k \in T(t)} e_k(t) P'_{ij}^k(t) = 0 \quad \forall i, j \in U \text{ and } i \neq j. \quad (9)$$

In order to vanish the specified  $\Delta w'_{ij}$ , we need to set the constraint

$$P'_{ij}^k = 0, \quad \forall i, j, k \in U \text{ and } i \neq j. \quad (10)$$

Under these constraints, the complexity of the RTRL algorithm is reduced from  $O(n^4)$  to  $O(n^3)$ .

We have defined the RTRL network with  $m$  input nodes and  $n$  processing nodes in the input and the output layers, respectively. The sensitivity terms  $P_{ij}^k$  and  $P'_{ij}^k$  are updated as in Equations 6 and 7, respectively. According to Equation 6,  $P_{ij}^k$  is calculated by adding  $n$  products of the terms  $w'_{kl}$  and  $P_{ij}^l$  and then scaling with  $f'[S_k(t)]$ . Therefore, the number of mathematical operations for each  $P_{ij}^k$  is  $n+1$ . There are totally  $n^2 m$  of  $P_{ij}^k$  in the network. Thus, the number of computations in Equation 6 is  $n^2 m [n+1]$  ( $= n^3 m + n^2 m$ ). According to Equation 7, the number of mathematical operations for each  $P'_{ij}^k$  is the same as that of each  $P_{ij}^k$  (i.e.,  $n+1$ ). There are  $n^3$  of  $P'_{ij}^k$  to be updated in each learning cycle of the RTRL algorithm. However, there are only  $n^2$  of  $P'_{ij}^k$  to be updated in each learning cycle of the FRTRL algorithm. Thus, the total numbers of computations of the RTRL and the FRTRL algorithms are  $[n+1][n^3 + n^2 m]$  ( $= n^4 + n^3 m + n^3 + n^2 m$ ) and  $[n+1][n^2 + n^2 m]$  ( $= n^3 m + n^3 + n^2 m + n^2$ ), respectively. In both algorithms, the number of computations in updating the sensitivity terms dominates the total number of computations in each learning cycle. By comparing the terms  $n^4 + n^3 m + n^3 + n^2 m$  and  $n^3 m + n^2 m + n^3 + n^2$ , the complexities of the RTRL and the FRTRL algorithms are  $O(n^4)$  and  $O(n^3)$ , respectively.

Conceptually, if we fill  $P'_{ij}^k$  into a 3-dimensional matrix, all elements excepting those which fall on the diagonal plane of the matrix are vanished. Therefore, all the computations confine to the matrix elements within the diagonal plane. As the

size of the matrix increases, the ratio between the number of elements within the diagonal plane and the total number of elements in the matrix becomes smaller. Thus, as the network scales up, the number of computations of the FRTRL algorithm is apparently smaller as compared with that of the RTRL algorithm.

However, in some applications, there is a large number of input nodes in the RTRL network. This leads to a large inter-weight matrix and a large number of sensitivity terms  $P_{ij}^k$ . Therefore, in the FRTRL algorithm, a specific number of randomly selected elements in the inter-weight matrix ( $w_{ij}$ ) will be kept constant throughout the learning phase. This can be achieved by setting  $P_{ij}^k = 0$  for some randomly selected  $i$ . Hence, the complexity of the FRTRL algorithm can be further reduced. A number of inter-weights is chosen for backpropagating the error. In order to maintain the generalization capability and the convergency ability of the network, those inter-weights are randomly selected. A specific number  $r$  is assigned to be the number of inter-weights responsible for backpropagating the error to each input node. Therefore, the total number of inter-weights that should be updated in each learning cycle is only  $mr$  ( $0 < r \leq n$ ) in the FRTRL algorithm instead of  $mn$  in the RTRL algorithm. Thus, for those inter-weights  $w_{ij}$  that should be kept constant, we have

$$\Delta w_{ij}(t) = \alpha \sum_{k \in T(t)} e_k(t) P_{ij}^k(t) = 0$$

$$\forall i \in U' \subset U, j \in I, \quad (11)$$

where  $U'$  contains  $n-r$  elements randomly selected from  $U$ . That is, we need to set the constraint

$$P_{ij}^k = 0, \quad \forall i \in U' \subset U, j \in I, k \in U. \quad (12)$$

### 3. Results and Discussions

In the experiments, a subset of the NETtalk[6] database is used as the training set. When one English letter of the incoming text stream is input to the network, the network outputs its corresponding phoneme. The letters and the phonemes were represented in different ways [6]. The letters were represented by 29 dedicated units, one for each letter of the alphabet, plus an additional 3 units to encode punctuations and word boundaries. The whole set of phonemes were classified by 21 articulatory features and 5 additional units encoding stress and syllable boundaries. Thus, the total number of actual outputs of the network is 26.

The network is designed to be fully connected with 30 input nodes (including bias). The learning rate ( $\alpha$ ) was set to 1.05 and the initial values of the fixed intra-weights ( $w_{ij} \quad \forall i \neq j$ ) were set

to 0.05. With the above settings, several networks (with various numbers of processing nodes and various values for the specific number  $r$ ) were trained until 5 training epochs<sup>2</sup> have been reached. The training set consists of 1001 training patterns. The amount of CPU time consumed for different network sizes is illustrated in Figure 1. As shown in Figure 1, the FRTRL algorithm diminishes the computation time as compared with the original algorithm.

We not only investigate the computational complexity of the FRTRL algorithm, but also investigate its generalization capability and convergency ability in the task of text-to-phoneme conversion. In the second experiment, the RTRL network was set with  $\alpha = 1.05$ ,  $w'_{ij} = 0.05 \quad \forall i \neq j$ , and  $r = n = 35$ . The training sequence and the unseen test set consist of 957 and 934 words (8079 and 5961 patterns), respectively. Both were randomly selected from the NETtalk database. Each output vector is represented by a 26-bit codeword. The most significant 21 bits and the remaining 5 bits of the codeword represent the phoneme and its corresponding stress or syllable boundary, respectively. A threshold of 0.5 was set such that when the actual output value is greater than the threshold, the corresponding bit of the codeword was set to '1'; otherwise, it was set to '0'. Therefore, each bit of the codeword is either a '0' or a '1'. If all the 26 bits of one codeword are exactly the same as that of the desired phoneme, this output phoneme is said to be correct. Figure 2 shows the accuracy (percentage of correct) of the output phonemes for various numbers of training epochs. The accuracies of the network obtained by the training and the test sets are very close. This shows that the RTRL network does generalize the relationship between the input text and its corresponding phonetic transcriptions.

Throughout the experiment, the RTRL and the FRTRL algorithms were operated in turn. When the mean-square-error of the target output nodes starts to increase, all the inter- and intra-weights were updated using the RTRL algorithm. Once the mean-square-error stops increasing, those specified weights were frozen again. After 72 training epochs, the accuracy of the output phonemes for the test set is 55.63%, and the accuracy increases as the number of epochs increases. The same training set, test set, and parameter settings were used to train a Backpropagation (BP) network<sup>3</sup> with 14 hidden nodes. In this case, after 72 training epochs, the accuracy of the output phonemes for the test set is 36.29%, which is significantly lower than that of the FRTRL network.

<sup>2</sup>One training epoch is defined as the presentation of the whole training set once.

<sup>3</sup>The term BP network denotes the feedforward networks trained with the Back-propagation algorithm.

To improve the conversion accuracy, an output post-processing technique is proposed. Instead of using a threshold to make the network outputs to be either '0' or '1', the actual values of the outputs were compared with two codebooks, A and B. Codebook A, with 54 entries, is formed by the 21-bit codewords, and codebook B, with 6 entries, is formed by the 5-bit codewords. Therefore, each entry of codebook A represents a phoneme, while that of codebook B represents a type of stresses or a syllable boundary. Each of the output vectors consists of 26 numbers ranging from 0 to 1, and each vector was divided into two parts - the first 21 numbers and the remaining five numbers. For each output vector, entries of codebooks A and B with the smallest Euclidean distance to the two parts of the output vector were found. If the corresponding phoneme and stress are identical to the desired one, the output vector is said to be correct. With this post-processing step, there is improvement in the accuracies of the FRTRL network and the BP network. The accuracies are 62.83% for the FRTRL network and 54.99% for the BP network. However, they are not comparable with the average accuracy (77%) of the NETtalk system stated in [6]. This is because only one-twentieth of the NETtalk database had been used as the training set in our experiments and the test set used in [6] is the same as ours.

In NETtalk, the temporal information of the input text streams is encoded by converting temporal features into spatial features. This is implemented by using a window with seven consecutive letters, and the window shifts for one letter over the text stream for each learning cycle. However, in our proposed network architecture, only one letter is presented to the network in each learning cycle. The temporal information of the input text streams is encoded by the weighted recurrent paths. Thus, in our proposed method, the weight matrix is apparently smaller because the number of input nodes is reduced by one seventh. Moreover, the window size does not need to be defined in advance. Even though the smallest unit (one letter) is presented to the network, the temporal information can still be learnt.

#### 4. Conclusion

Recurrent neural networks with the fast real time recurrent learning (FRTRL) algorithm is proposed for encoding the spatio-temporal information in text-to-phoneme conversion. The proposed FRTRL algorithm aims at reducing the computational complexity of the RTRL algorithm in order to make the training time being feasible. With the evidence of the experimental results, the ratio of the computational time of the FRTRL and the RTRL algorithms

becomes smaller as the RTRL network scales up. Moreover, in the task of text-to-phoneme conversion, the recurrent neural networks is significantly smaller in size and achieve better performance as compare to that of the feedforward networks.

#### References

- [1] R. J. Williams, and D. Zipser, "Experimental Analysis of the Real Time Recurrent Learning Algorithm," *Connection Science*, vol. 1, pp. 87-111, 1989.
- [2] N. H. Wulff, and J. A. Hertz, "Prediction with Recurrent Networks," *Neural Networks for Signal Processing II, Proceedings of the IEEE-SP Workshop*, pp. 464-473, 1992.
- [3] R. L. Lindsey, D. W. Ruck, S. K. Rogers, and M. Kabrisky, "Function Prediction Using Recurrent Neural Networks," *SPIE, vol. 1710, Science of Artificial Neural Networks*, vol. 2, pp. 438-448, 1992.
- [4] R. Kamimura, "Application of Temporal Supervised Learning Algorithm to Generation of Natural Language," *International Joint Conference on Neural Networks*, vol. 1, pp. 201-207, 1990.
- [5] D. H. Klatt, "Review of Text-to-speech Conversion for English," *Journal of Acoustical Society of America*, vol. 82, no. 3, pp. 737-793, 1987.
- [6] T. J. Sejnowski, and C. R. Rosenberg, "Parallel Networks that Learn to Pronounce English Text," *Complex Systems*, vol. 1, pp. 145-168, 1987.
- [7] S. M. Lucas, and R. I. Damper, "A Connectionist approach to Text-Phonemics Translation using Syntactic Neural Networks," *RNNS/IEEE Symposium on Neuroinformatics and Neurocomputers*, vol. 1, no. 1, pp. 25-36, 1992.
- [8] Y. Yamaguchi, and T. Matsumoto, "Syntactic Analysis and Letter-to-Phoneme Conversion Using Neural Networks - an Application of Neural Networks to an English Text-to-Speech System," *Systems and Computers in Japan*, vol. 24, no. 8, pp. 71-81, 1993.
- [9] D. Zipser, "A Subgrouping Strategy that Reduce Complexity and Speeds Up Learning in Recurrent Networks," *Neural Computation*, vol. 1, pp. 552-558, 1989.
- [10] T. Catfolis, "A Method for Improving the Real Time Recurrent Learning Algorithm," *Neural Networks*, vol. 6, pp. 807-821, 1993.

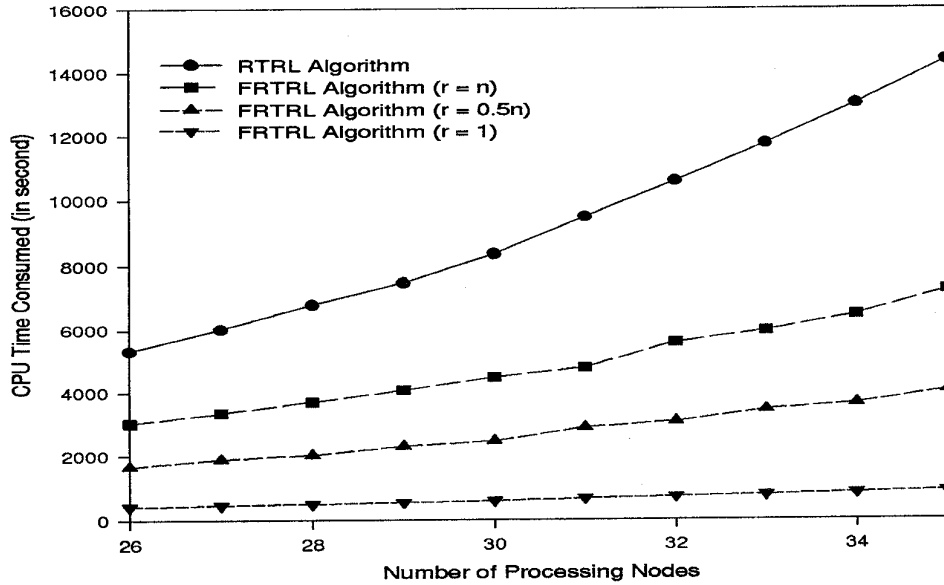


Fig. 1: The consumed CPU time (for 5 training epochs) of the two learning algorithms with different  $n$  and  $r$ .

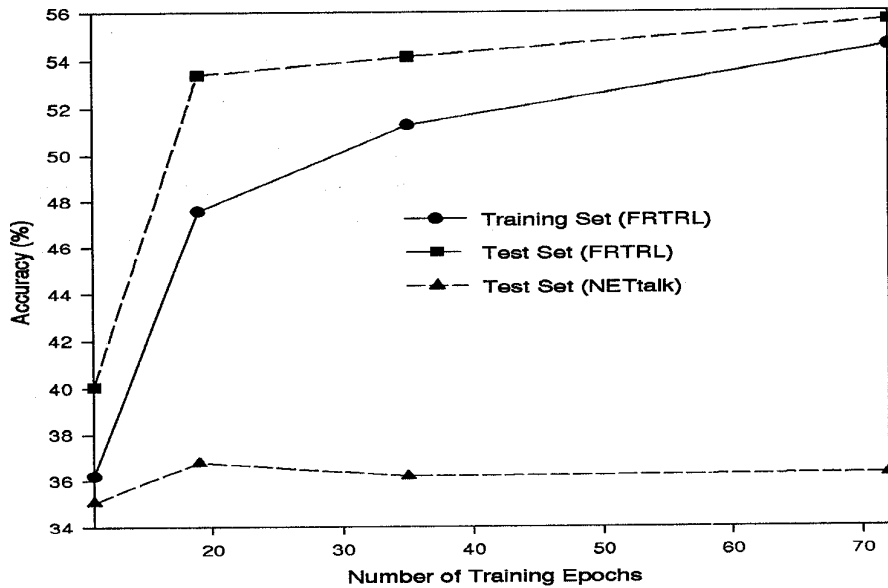


Fig. 2: The accuracy of the output phonemes with different number of training epochs.