In The Proceeding of ICONIP'98, Japan, pages 1393 - 1396, 1998.

Training Recurrent Neural Networks by Using Parallel Recursive Prediction Error Algorithm

Daqing Chen and Laiwan Chan
Email:lwchan@cse.cuhk.edu.hk

Dept. of Computer Science and Engineering
The Chinese University of Hong Kong, Shatin, Hong Kong

ABSTRACT

As an alternative local and higher order algorithm, the parallel recursive prediction error algorithm (PRPE) is used to train the recurrent neural networks (RNNs). This algorithm uses a modified form of the well-known recursive prediction error algorithm (RPE) such that the computation can be distributed into each node in the network. Therefore, the algorithm has a better trade-off between computational cost and convergence time. Several examples of training the RNNs to perform time series prediction task are presented to demonstrate the superior convergence performance of the algorithm compared with the real time recurrent learning algorithm (RTRL).

KEYWORDS: recurrent network, learning algorithm, recursive prediction

1. Introduction

Recurrent neural networks (RNNs) have received much research attention because of their powerful capability to preserve information through time. In recent years, a variety of approaches to learning in the RNNs have been proposed, such as the **back propagation through time** (BPTT) [1], **real time recurrent learning** (RTRL) [2], **extended Kalman filter** (EKF) [3] and **recursive least squares** (RLS) [5], etc, all of which are developed as natural extensions of that used for training the static feedforward networks. One common feature of these algorithms is good convergence property, which is gained with cost of expensive computations. The better the property, the more intense are the computations. Thus, locally computational algorithms, such as the **decoupled** EKF algorithm (DEKF) [4] and the **extended recursive least squares** (ERLS) algorithm [5], are of considerable value because they have, to certain extend, a trade-off between computational effort and learning speed.

In this paper, the training of recurrent network is regarded as parameter estimation of dynamic network based on input-output model description of the network, and the **parallel recursive prediction error algorithm** (PRPE) [6] is used as an alternative local and higher order algorithm for the recurrent network learning. This algorithm uses a modified form of the well-known **recursive prediction error** algorithm (RPE) [7] [8] such that the computation can be distributed into each node in the network. Several examples of training the RNNs to perform time series prediction of sunspot activity data are presented to demonstrate the superior convergence performance of the algorithm compared with the RTRL algorithm in terms of training time and prediction capability.

2. Description of the Recurrent Network

Consider an arbitrary recurrent neural network with m-input n-output and one or more recurrent layers. Such a recurrent network is denoted by $m \times N \times n$, where N is the number of recurrent nodes. Without loss of generality, the hidden nodes and output nodes in the network are characterized by continuous nonlinear function and linear function, respectively. The time delay step for the output of each recurrent node is set to one. Let n_w be the number of all the weights and thresholds of the network, and $\mathbf{W} = [w_1 \cdots w_{n_w}]^T$ represents all the weights and thresholds ordered in a chosen way. The over-all input-output relationship of the network is then expressed as

$$\mathbf{Y}(t, \mathbf{W}) = \mathbf{F}(\mathbf{X}(t), \mathbf{W}) \tag{1}$$

where t denotes discrete time, $\mathbf{X}(t) = [x_1(t) \cdots x_m(t)]^T$ and $\mathbf{Y}(t) = [y_1(t) \cdots y_n(t)]^T$ are the network input and output vectors, respectively; $\mathbf{F}(\cdot, \cdot)$ is a nonlinear vector-valued function. It should be noted that because of the existence of internal recurrent layers, eq. (1) defines a dynamic system.

Let $n_w \times n$ matrix $\mathbf{\Psi}(t, \mathbf{W})$ denote the gradient of $\mathbf{Y}(t, \mathbf{W})$ with respect to $\mathbf{W}$, $\mathbf{\Psi}(t, \mathbf{W}) = \left[\frac{d\mathbf{Y}(t,\mathbf{W})}{d\mathbf{W}}\right]^T = \mathbf{g}(\mathbf{X}(t), \mathbf{W})$. For the recurrent network, such a gradient is governed by dynamical equations due to the network dynamics, and therefore is called as **dynamic derivatives**. Usually, the calculation of dynamic derivatives for a recurrent network is more computationally expensive than that for a purely static feedforward network. For example, for the simplest form of the network (1) with only single fully-connected recurrent layer and N hidden nodes, the derivatives can be easily found as

$$\frac{\partial y_s(k)}{\partial w_{js}^{Out}} = h_j(k) \tag{2}$$

$$\frac{\partial y_s(k)}{\partial w_{ij}^{RS}} = \sum_{l}^{N} w_{ls}^{Out} \frac{\partial h_l(k)}{\partial w_{ij}^{RS}}, l, i, j = 1, \cdots, N \tag{3}$$

$$\frac{\partial y_s(k)}{\partial w_{ij}^{In}} = \sum_{l}^{N} w_{ls}^{Out} \frac{\partial h_l(k)}{\partial w_{ij}^{In}}, j = 1, \cdots, N; i = 1, \cdots, M \quad s = 1, \cdots, n \tag{4}$$

where

$$\frac{\partial h_l(k)}{\partial w_{ij}^{RS}} = h_l^{'}(k)[\delta_{lj} h_i(k-1) + \sum_{t}^{N} w_{tl}^{RS} \frac{\partial h_t(k-1)}{\partial w_{ij}^{RS}}] \tag{5}$$

$$\frac{\partial h_l(k)}{\partial w_{ij}^{In}} = h_l^{'}(k)[\delta_{lj} x_i(k) + \sum_{t}^{N} w_{tl}^{RS} \frac{\partial h_t(k-1)}{\partial w_{ij}^{In}}] \tag{6}$$

with initial conditions $\frac{\partial x_j(0)}{\partial w_{mn}^{RS}} = 0$, $\frac{\partial x_j(0)}{\partial w_{mn}^{In}} = 0$. $h_j(k)$ is the output of the jth hidden node, w_{ij}^{In}, w_{tl}^{RS} and w_{js}^{Out} are the weights linked from the ith input node to the jth hidden node, the tth hidden node to the lth hidden node, and the jth hidden node to the sth output node, respectively; and δ_{lj} is a Kronecker delta equal to 1 when $l = j$ and zero otherwise.

For notation convenience, let the nodes in the network be numbered from 1 to p and the weights and threshold of the ith node be arranged an n_{w_i}-dimensional vector $\mathbf{W}_i$, $i = 1, \cdots, p$. $\mathbf{W}$ and $\mathbf{\Psi}(t, \mathbf{W})$ can then be partitioned and represented as $\mathbf{W} = [\mathbf{W}_1 \cdots \mathbf{W}_p]^T$ and $\mathbf{\Psi}(t, \mathbf{W}) = \left[\mathbf{\Psi}_1(t, \mathbf{W}) \cdots \mathbf{\Psi}_p(t, \mathbf{W})\right]^T$, where $\mathbf{\Psi}_i(t, \mathbf{W})$, an $n_{w_i} \times n$ matrix, is the gradient of $\mathbf{\Psi}(t, \mathbf{W})$ with respect to $\mathbf{W}_i$ and $i = 1, \cdots, p$.

3. Parallel Recursive Prediction Error Algorithm

Based on the input-output model (1), the training of recurrent network can be regarded as parameter estimation of dynamic system. Given a set of sample data $\{\mathbf{X}(t), \mathbf{T}(t)\}_{t=1}^{P}$, where $\{\mathbf{T}(t)\}$ is the n-dimensional desired output vector of the network. Let $\mathbf{e}(t, \mathbf{W}) = \mathbf{T}(t) - \mathbf{Y}(t, \mathbf{W})$ be the discrepancy between $\mathbf{T}(t)$ and $\mathbf{Y}(t, \mathbf{W})$ at time t, and the criterion function for measure of the closeness between $\mathbf{T}(t)$ and $\mathbf{Y}(t, \mathbf{W})$ be the quadratic form of $\mathbf{e}(t, \mathbf{W})$

$$J(\mathbf{W}) = \frac{1}{2P} \sum_{t=1}^{P} \mathbf{e}^T(t, \mathbf{W})\mathbf{e}(t, \mathbf{W}) \tag{7}$$

The optimal $\mathbf{W}$ may then be determined by minimizing (7). Such a method of obtaining a desired $\mathbf{W}$ is well-known as the **prediction error estimation** (PE) method [7]. Minimization of (7) can be achieved iteratively according to

$$\mathbf{W}^{(k)} = \mathbf{W}^{(k-1)} + \alpha \Xi(\mathbf{W}^{(k-1)}) \tag{8}$$

where k denotes the iteration step, $\Xi(\mathbf{W}^{(k-1)})$ is a search direction and α is a positive constant chosen appropriately to guarantee convergence of the iterative procedure.

The most commonly used search direction is the modified negative gradient direction defined by

$$\Xi(\mathbf{W}) = \mathbf{M}(\mathbf{W})(-\nabla J(\mathbf{W})) \tag{9}$$

where $\mathbf{M}(\mathbf{W})$ is a positive definite $n_w \times n_w$ matrix and

$$\nabla J(\mathbf{W}) = -\frac{1}{P} \sum_{t=1}^{P} \mathbf{\Psi}(t, \mathbf{W})\mathbf{e}(\mathbf{t}, \mathbf{W}) \tag{10}$$

is the gradient of the criterion (7) with respect to $\mathbf{W}$.

Let $\mathbf{M}^{-1}(\mathbf{W})$ be chosen to be the following near-diagonal matrix

$$\mathbf{M}^{-1}(\mathbf{W}) = \frac{1}{P} \sum_{t=1}^{P} \begin{bmatrix} \mathbf{\Psi}_1(t, \mathbf{W})\mathbf{\Psi}_1^T(t, \mathbf{W}) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \mathbf{\Psi}_p(t, \mathbf{W})\mathbf{\Psi}_p^T(t, \mathbf{W}) \end{bmatrix} \tag{11}$$

the search direction $\Xi(\mathbf{W})$ can be decomposed into p smaller vectors

$$\Xi_i(\mathbf{W}) = \left[\frac{1}{P} \sum_{t=1}^{P} \mathbf{\Psi}_i(t, \mathbf{W})\mathbf{\Psi}_i^T(t, \mathbf{W})\right]^{-1} \times \frac{1}{P} \sum_{t=1}^{P} \mathbf{\Psi}(t, \mathbf{W})\mathbf{e}(t, \mathbf{W}), i = 1, \cdots, p \tag{12}$$

and the algorithm (7) can then be decomposed into p sub-algorithms, each corresponding to a node in the network

$$\mathbf{W}_i^{(k)} = \mathbf{W}_i^{(k-1)} + \alpha \Xi_i(\mathbf{W}^{(k-1)}), i = 1, \cdots, p \tag{13}$$

Algorithm (13) is the **parallel prediction error** algorithm (PPE) [6]. By applying recursive approximations, the PPE algorithm can take its recursive form, which leads to the parallel recursive prediction error algorithm as [6]

$$\begin{aligned}
\begin{bmatrix} \mathbf{Y}(t) \\ \mathbf{\Psi}(t) \end{bmatrix} &= \begin{bmatrix} \mathbf{Y}(\mathbf{X}(t), \hat{\mathbf{W}}(t-1)) \\ \mathbf{g}(\mathbf{X}(t), \hat{\mathbf{W}}(t-1)) \end{bmatrix} && (14) \\
\mathbf{\Delta}_i(t) &= \gamma_m \mathbf{\Delta}_i(t-1) + \gamma_g \mathbf{\Psi}_i(t)\mathbf{e}(t) && (15) \\
\mathbf{P}_i(t) &= \frac{1}{\lambda}\{\mathbf{P}_i(t-1) - \mathbf{P}_i(t-1)\mathbf{\Psi}_i(t) \\
&\quad \times \left[\lambda \mathbf{I} + \mathbf{\Psi}_i^T(t)\mathbf{P}_i(t-1)\mathbf{\Psi}_i(t)\right]^{-1} \\
&\quad \times \mathbf{\Psi}_i^T(t)\mathbf{P}_i(t-1)\} && (16) \\
\hat{\mathbf{W}}_i(t) &= \hat{\mathbf{W}}_i(t-1) + \mathbf{P}_i(t)\mathbf{\Delta}_i(t) && (17)
\end{aligned}$$

where $i = 1, \cdots, p$. $\hat{\mathbf{W}}_i(t)$ is the estimate of $\mathbf{W}_i$ at time t, γ_m and γ_g are the gain and momentum parameter, respectively, λ is the usual forgetting factor, $0 < \lambda < 1$, and $\mathbf{I}$ is a $n_{w_i} \times n_{w_i}$ identity matrix. $\mathbf{\Delta}_i(t)$ and $\mathbf{P}_i(t)$ are the n_{w_i}-dimensional smoothed stochastic gradient and

Table 1: Sum of squared error of the $13 \times 5 \times 1$ network trained 10 times by the PRPE algorithm

Parameters: $\lambda = 0.99, \gamma_g^{In} = 0.15, \gamma_g^{RS} = 0.001, \gamma_g^{Out} = 0.1, \gamma_m^{In} = \gamma_m^{RS} = \gamma_m^{Out} = 0$

Initial Weights	Stop Learning	Test on Learning Samples	Prediction
$[-0.1, 0.1]$	0.542900	0.538795	1.246756
$[-0.2, 0.2]$	0.536116	0.533376	1.230346
$[-0.3, 0.3]$	0.525223	0.521615	1.225483
$[-0.4, 0.4]$	0.535999	0.528975	1.250875
$[-0.5, 0.5]$	0.547631	0.542872	1.254175

the $n_{w_i} \times n_{w_i}$ covariance matrix for the ith node, respectively.

It is clear that the RTRL algorithm can be obtained from the PRPE algorithm by replacing each $\mathbf{P}_i(t)$ in (16) with the identity matrix.

4. Simulation Experiments

In this section, several simulation experiments are conducted to investigate and demonstrate the efficiency of the PRPE algorithm compared with the RTRL algorithm.

In the following simulations, a recurrent network is trained to perform one-step prediction for time series. The data-set is produced by the yearly averaged sunspot activity recorded from years 1700 to 1979. The network architecture is chosen as $13 \times 5 \times 1$, where the input nodes correspond to 12 time delays of the original series and a constant unit input used for the thresholds of hidden nodes, respectively. The first 100 samples of the sunspot series are used for training the network, and the last 168 samples for testing the prediction capability of the resulting network. Each hidden node is characterized by the sigmoidal function $h(\cdot) = \frac{1-e^{-(\cdot)}}{1+e^{-(\cdot)}}$. The calculation of dynamic derivatives of the network is given as eq. (2) - eq. (6), and the initial state of each hidden node in the network is set to zero. The initial network parameters are set randomly to values in the intervals $[-0.1, 0.1]$, $[-0.2, 0.2]$, $[-0.3, 0.3]$, $[-0.4, 0.4]$, and $[-0.5, 0.5]$, respectively. The learning parameters for the two algorithms are given as

PRPE algorithm (14) - (17):
$\lambda = 0.99$, $\mathbf{P}_i(0) = 100\mathbf{I}_i$, $\forall i$
$\gamma_g^{In} = 0.15$, $\gamma_g^{RS} = 0.001$, $\gamma_g^{Out} = 0.1$

RTRL algorithm:
Set 1: $\gamma_g^{In} = 0.01, \gamma_g^{RS} = 0.001, \gamma_g^{Out} = 0.01$
Set 2: $\gamma_g^{In} = 0.01, \gamma_g^{RS} = 0.01, \gamma_g^{Out} = 0.01$
Set 3: $\gamma_g^{In} = 0.001, \gamma_g^{RS} = 0.001, \gamma_g^{Out} = 0.1$

For both the algorithms $\gamma_m^{In} = \gamma_m^{RS} = \gamma_m^{Out} = 0$.

Under the same initial weights and thresholds, the network is trained 10 times by using the PRPE algorithm and 500 times by the RTRL algorithm, respectively. The results obtained from the network training and prediction with each set of initial network parameters and learning parameters are summarized in Table 1 and 2.

From the simulation experiments above, it is obvious that the number of iterations required by the PRPE algorithm to convergence is much less than what is required by the RTRL algorithm, and meanwhile, the resulting networks still have good or even better prediction capability.

5. Concluding Remarks

Recurrent networks have the potential to solve problems with temporal extent significantly better than the purely static feedforward networks due to their dynamic nature. Unfortunately, the training algorithms for recurrent learning have not been well developed. Because of the calculation of dynamic derivatives, some commonly used algorithms like the BPTT and RTRL, developed as the extension of that used for training static networks, are usually inefficient. Although some higher order algorithms, such as the EKF algorithm, have been presented to improve convergence property, they are computationally intensive, and consequently, the advantage gained in smaller amount of training time may be negated by the higher computational requirements.

In this paper, the parallel recursive prediction error algorithm is used for recurrent network learning. The PRPE algorithm is a locally computational and higher order algorithm. The main attraction of the PRPE algorithm is its relative simplicity in computation and can be integrated into the parallel structure of the network. Therefore, the algorithm has a better control over the trade-off between computational cost and convergence time. The effectiveness of such an algorithm have been examined by training recurrent network to perform time series prediction task. The results obtained from the simulation experiments have shown that the training time could be reduced considerably by using the PRPE algorithm, and the prediction capability of the resulting network is superior or comparable to that of the network trained by

Table 2: Sum of squared error of the $13 \times 5 \times 1$ network trained 500 times by the RTRL algorithm

Parameters: $\gamma^{In} = 0.01, \gamma^{RS} = 0.001, \gamma^{Out} = 0.01, \gamma_m^{In} = \gamma_m^{RS} = \gamma_m^{Out} = 0$

Initial Weights	Stop Learning	Test on Learning Samples	Prediction
$[-0.1, 0.1]$	0.607001	0.599954	1.334655
$[-0.2, 0.2]$	0.612499	0.605069	1.341206
$[-0.3, 0.3]$	0.619302	0.611302	1.347286
$[-0.4, 0.4]$	0.624365	0.615990	1.351898
$[-0.5, 0.5]$	0.626777	0.618024	1.355297

Parameters: $\gamma^{In} = 0.01, \gamma^{RS} = 0.01, \gamma^{Out} = 0.01 \gamma_m^{In} = \gamma_m^{RS} = \gamma_m^{Out} = 0$

Initial Weights	Stop Learning	Test on Learning Samples	Prediction
$[-0.1, 0.1]$	0.527440	0.523087	1.367918
$[-0.2, 0.2]$	0.528698	0.525148	1.382673
$[-0.3, 0.3]$	0.528356	0.524244	1.406690
$[-0.4, 0.4]$	0.525332	0.519747	1.443918
$[-0.5, 0.5]$	0.523338	0.516985	1.511928

parameters: $\gamma^{In} = 0.001, \gamma^{RS} = 0.001, \gamma^{Out} = 0.1 \gamma_m^{In} = \gamma_m^{RS} = \gamma_m^{Out} = 0$

Initial Weights	Stop Learning	Test on Learning Samples	Prediction
$[-0.1, 0.1]$	0.620160	0.621735	1.356867
$[-0.2, 0.2]$	0.624745	0.617296	1.367910
$[-0.3, 0.3]$	0.623457	0.615972	1.407448
$[-0.4, 0.4]$	0.614547	0.606536	1.464070
$[-0.5, 0.5]$	0.609241	0.600572	1.549402

the RTRL algorithm.

References

[1] D. E. Rumelhart, G. E. Hinton, and R. L. Williams, Learning internal representation by error propagation. in Parallel Distributed Processing: Exploration in the Microstructure of Cognition, Vol. 1, Cambridge, MA: Boadford Books, 1986.

[2] R. J. Williams and D. Zipser, A Learning Algorithm for Continually Running Full Recurrent Neural Networks. Neural Computation, Vol. 1, pp. 270-280, 1989.

[3] R. J. Williams, Training Recurrent Networks Using the Extended Kalman Filter. International Joint Conference on Neural Networks, Vol. IV, pp. 241-246, Baltimore 1992.

[4] G. V. Puskorius and L. A. Feldkamp, Neurocontrol of Nonlinear Dynamical Systems with Kalman Filter Trained Recurrent Networks. IEEE Trans. Neural Networks, Vol. 5, No. 2, pp. 279-297, 1994.

[5] Q. Xu, K. Krishnamurthy, *et al.* A Recursive Least Squares Training Algorithm for Multilayer Recurrent Neural Networks. Proc. 1994 American Control Conference. Vol. 2, pp. 1712-1716, 1994.

[6] S. Chen, C. F. N. Cowan, *et al.* Parallel recursive prediction error algorithm for training layered neural networks. International Journal of Control, Vol. 51, No. 6, pp. 1215-1228, 1990.

[7] Lennart Ljung and Torsten Söderström, Theory and Practice of Recursive Identification. The MIT Press, 1983.

[8] S. Chen and S. A. Billings, Recursive prediction error estimator for non-linear models. International Journal of Control, Vol. 49, No. 2, pp. 569-594, 1989.

Acknowledgement The authors would like to thank The Research Grants Council, HK for support.