World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:1, No:10, 2007

Multi-Context Recurrent Neural Network for Time Series Applications

B. Q. Huang, Tarik Rashid and M-T. Kechadi

***Abstract*—**this paper presents a multi-context recurrent network for time series analysis. While simple recurrent network (SRN) are very popular among recurrent neural networks, they still have some shortcomings in terms of learning speed and accuracy that need to be addressed. To solve these problems, we proposed a multi-context recurrent network (MCRN) with three different learning algorithms. The performance of this network is evaluated on some real-world application such as handwriting recognition and energy load forecasting. We study the performance of this network and we compared it to a very well established SRN. The experimental results showed that MCRN is very efficient and very well suited to time series analysis and its applications.

***Keywords*—**Gradient Descent Method, Recurrent Neural Network, Learning Algorithms, Time Series, BP

I. INTRODUCTION

USUALLY a time series analysis considers the observed temporal series of data values and predicts the future data values. The time series analysis is needed in any applications in which the time is very relevant and crucial such as energy load, stocks, indices, weather, speech recognition, handwriting recognition, etc. Artificial neural networks (ANN) are among the most successful approaches for such time series applications [6, 12, 25, 33]. Among ANNs, Time Delay Neural Network (TDNN) and Recurrent Neural Network (RNN) are the most widely used for time series applications. However, both of them have some limitations.

Generally, some conventional neural networks (see Figure 1a) [17] can not handle time series patterns successfully. In order to overcome this problem, an alternative method that uses the context as part of the next inputs can model the temporal series data by considering the context, which stores the previous inputs, as part of the next inputs. In addition, the time series patterns are characterised by their variable length and most conventional networks accept as input only patterns with equal length [7, 34]. For instance, such context can be represented as a time series of buffer-inputs. Each buffer represents a time step. The input information is then shifted from one buffer to another every time step. The first buffer is always fed with a new input. This type of network is called TDNN [7], (see Figure 1b). TDNN has been successfully applied on many practical applications, such as speech recognition [6, 7], handwriting recognition [25] and forecasting [12].

The authors are in School of Computer Science & Informatics of University College Dublin, Belfield, Dublin 4, Ireland. Email: bingquan..huang@ucd.ie

However, TDNN has a difficulty of dealing with sequences of variable length, due to its fixed input window [1].

Recurrent neural networks [35] provide an alternative to TDNN. They are developed to deal with temporal or time-tagged patterns and are able to model systems having complex dynamics and/or the ob-served (collected) data is noisy. Recurrent networks are characterised from the others by their feed back loops, which allows them to handle temporal sequences. Generally, RNNs can be categorised into fully recurrent networks and partially recurrent network. For a fully recurrent network, every two neurons are connected from both directions.

However, they suffer from heavy computing overheads [18]. A partially recurrent network is a multi-layered perceptron network which augmented with one or more additional context layers storing output values of one of the layers delayed by one step and used for activating other layer in the next time step. Unlike fully recurrent networks, a partially recurrent network has much less computing overhead. Examples of these networks can vary in the type of the feed-back used.

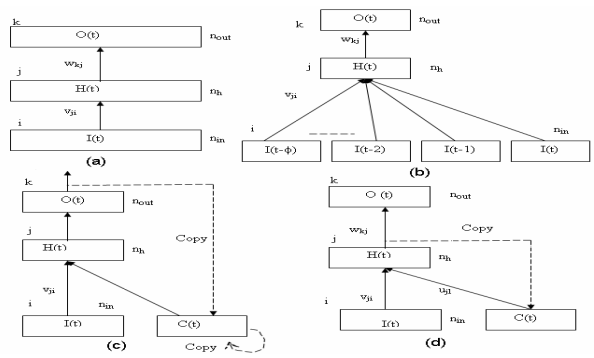

Fig. 1 (a) is the conventional neural network, (b) is the tapped delay time network, (c) is the Jordan network and (d) is the Elman network

The Jordan Network [20] (see Figure 1c) and the Elman network [10] are tow partially recurrent networks that are widely used. For the Jordan Network, the output of the previous step becomes the input to the current step. This network is able to produce a series of outputs given a fixed set of inputs as used for speech recognition [28]. The Jordan Network can only map the output memory but it can not map inputs that are not reflected in the outputs [5, 10, 36]. Therefore, the Elman network, called a simple recurrent

network (SRN) [5, 10], is developed (see Figure 1d) to overcome the limitations of Jordan Network. It uses the activation of context layer from the hidden neurons as the basic recurrent relation. The context layer is completely internal to the SRN because it references only the hidden neurons, and does not associate them with the network outputs.

However, the architecture of the SRN includes small memory, which is only one context layer and also its performance in terms of accuracy possibly depends more information on the hidden layer [8, 38]. Therefore, we propose a new neural network with three learning algorithms to improve the performance of SRN. We will show that this new architecture is well suited to time series applications such as handwriting recognition and energy load forecasting.

This paper is organised as follows: Section II introduces the architecture and principle of MCRN. Section III describes the three learning algorithms for MCRN. The complexity of MCRN and the problems of time serial applications are simply discussed in section IV and V respectively. Section VI provides experimental results. Conclusion and future work are given in section VII.

II. The Multi-context Recurrent Neural Network

The MCRN architecture is shown in Figure 2b and it consists of four layers: input, hidden, output, and multi-context layers. This architecture is similar to Elman tower network in terms of the number of layers. However, MCRN is characterised by the fact that the multi-context layers are directly linked to the output layer. Basically, MCRN combines the features of both Elman and Jordan networks, leading to a more complex connectivity between the layers but reducing the dependency of the network output on the hidden layer and speeding up the learning as we will show in the following sections.

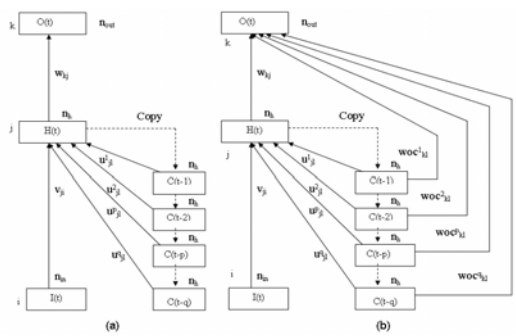

Fig. 2 displays two network architectures, (a) is the Elman Tower Network and (b) is the Multi Context Recurrent Network

In order to analyse this architecture and understand its dynamics and, therefore, validate the learning process on the MCRN network, some definitions and notations are introduced in the following section. Part of the notations are already included in Figure 2.

A. Notation

- **Neurons and layers:** The sets of indices $\{i, i', i''\}$, $\{j, j', j'', j'''\}$, $\{l, l'\}$ and $\{k, k', k'', k'''\}$ represent the input, hidden, context, and output neurons respectively. n_{in}, n_h and n_{out} are the number of neurons in the input, hidden/context, and output layers respectively.
- **Net inputs and outputs**: we denote by t the current time step. The input of neuron i in the input later is denoted by I_i at time t. $\tilde{h}_j$ Is the input of neuron j in the hidden layer, $\tilde{o}_k$ is the input to neuron k in the output layer. H_j is the output of neuron j in the hidden layer, $C_l(t-p)$ is the output of neuron l in the pth context layer O_k is the output of neuron k in the output layer, and d_k is the target of neuron k in the output layer.
- **Connection weights:** the weight connection from the input layer to the hidden layer is denoted by v_{ji} · u_{kl}^{p} is the weight connection from the pth context layer to the hidden layer, woc_{kl}^{p} is the weight connection from the pth context layer to the output layer and w_{kj} is the weight connection from the hidden layer to the output layer.

B. Network Dynamics

Assume Q be the total number of the context layers and q be the number of the active context layers. An "active layer" is a layer that is involved in storing previous time step data. It can be obtained by

$$q = \begin{cases} t & when \quad t < q \\ Q & else \end{cases} \quad \text{..... ....(1)}$$

Every time step the content of a context layer $C(t-i)$ is copied to the next context layer $C(t-i-1)$. At the same time the content is used as input to both hidden and output layers. MCRN is a feedforward network, and the information is propagated from the input to the output layers. Therefore, we first calculate the output of the hidden and context layers and then the output of the network. These are calculated as follows:

1. **Outputs of the hidden layer:** The output $H_j(t)$ of a neuron j at time t in the hidden layer is straightforward and given by the following:

$$\tilde{h}_j(t) = \sum_{i=1}^{n_{in}} I_i(t) v_{ji}(t) + \sum_{p=1}^{q} \sum_{l=1}^{n_h} C_l(t-p) u_{jl}^{p}(t) \text{.....(2)}$$

$$H_j(t) = f\left(\tilde{h}_j(t)\right) \text{.....(3)}$$

The update of the context layer outputs is done by propagating the output of a context layer to the input of the next layer, assuming that the weights for these links are all equal to unity. Equation (4) is used for propagating the information from one context layer to its successor and the equation (5) is used to copy the output of the hidden layer to the first context layer.

$$C_j(t-p) = C_j(t-p+1) \quad (4)$$

$$C_j(t-1) = H_j(t) \quad (5)$$

2. **Network outputs:** The output $O_k(t)$ of the kth neuron in output layer at time t can be obtained by:

$$\tilde{o}_k(t) = \sum_{j=1}^{q} H_j(t) w_{kj}(t) + \sum_{p=1}^{q}\sum_{l=1}^{n_h} C_l(t-p) woc_{kl}^{p}(t) \quad (6)$$

$$O_k(t) = f(\tilde{o}_k(t)) \quad (7)$$

where f in the second equation is the activation function.

III. Learning Algorithms

In this section we examine some learning algorithms and their suitability for the MCRN architecture. In addition, this will allow us to customise the MCRN parameters, such the learning rate, the number of hidden neurons and the number of context layers. The learning algorithms should also be suitable for time series applications, which represent the main reason for designing this network. We propose to explore three algorithms: Back-Propagation (BP) [1, 16, 23, 33, 37], Back-Propagation Through Time (BPTT) [1, 19, 23, 30, 39] and Dynamic Online Learning (DOL) [2, 25, 31]. These are described below.

A. Back Propagation (BP)

BBP is based on gradient descent method and tries to minimise the error of the network, which is the difference between the output and the target. However, usually due to the gradient descent process the BP technique tends to converge into local minima. This network has the ability to provide more information in order to allow the system to converge to better solutions. This information is stored in the context layers and at any given time this information is provided to the output layer to adjust the final solution. This facilitates good mapping and speeds up its convergence. Here we introduce the BP algorithm using an example of supervised learning. Given an input pattern s, the mean square error is expressed by

$$E_s = \frac{1}{2}\sum_{k=1}^{n_{out}} (d_{ks} - O_{ks})^2 = \frac{1}{2}\sum_{k=1}^{n_{out}} (e_{ks})^2 \quad (8)$$

where e_{ks} is the difference between the target and the network output:

$$e_{ks} = d_{ks} - O_{ks}$$

The objective is to minimize the total error E_{total}, which can be obtained by

$$E_{total} = \sum_{s=1}^{S} E(s) \quad (9)$$

The gradient descent method is used to calculate the derivative of the error with respect to the variable weights of the system $\frac{\partial E_{total}}{\partial w_{kj}}$, $\frac{\partial E_{total}}{\partial woc_{kl}^{p}}$ $\frac{\partial E_{total}}{\partial u_{jl}^{p}}$ and $\frac{\partial E_{total}}{\partial v_{ji}}$. According to the chain rule, the defined partial gradient (local gradient) terms in output and hidden layers are used to simplify the formulae. Therefore, $\frac{\partial E_{total}}{\partial w_{kj}}$ can be written as

$$\frac{\partial E_{total}}{\partial w_{kj}} = \sum_{s=1}^{S} \frac{\partial E_s}{\partial e_{ks}} \frac{\partial e_{ks}}{\partial O_{ks}} \frac{\partial O_{ks}}{\partial \tilde{o}_{ks}} \frac{\partial \tilde{o}_{ks}}{\partial w_{kj}} \quad (10)$$

Assume LG_{ks} be the local gradient of the kth neuron in the output layer for input pattern s. It can be expressed as follows:

$$LG_{ks} = \frac{\partial E_s}{\partial e_{ks}} \frac{\partial e_{ks}}{\partial O_{ks}} \frac{\partial O_{ks}}{\partial \tilde{o}_{ks}} = -e_{ks} f'(\tilde{o}_{ks}) \quad (11)$$

The partial gradient LG_{js} of the jth neuron in the hidden layer for an input pattern s, is given by

$$LG_{js} = \frac{E_s}{\partial \tilde{h}_{js}} = \sum_{k=1}^{n_{out}} LG_{ks} w_{kj} f'(\tilde{h}_{js}) \quad (12)$$

From equations (10), (11) and (12), the weight-changes for each category are as follows

$$\frac{\partial E_{total}}{\partial w_{kj}} = \sum_{s=1}^{S} LG_{ks} H_{js} \quad (13)$$

$$\frac{\partial E_{total}}{\partial woc_{kj}^{p}} = \sum_{s=1}^{S} LG_{ks} C_{js}(t-p) \quad (14)$$

$$\frac{\partial E_{total}}{\partial v_{ji}} = \sum_{s=1}^{S} LG_{js} I_{is} \quad (15)$$

$$\frac{\partial E_{total}}{\partial u_{kl}^{p}} = \sum_{s=1}^{S} LG_{js} H_{js}(t-p) \quad (16)$$

The adjustments or updates of the weights are obtained by adding the corresponding weight-changes to the previous values:

$$w'_{ab} = w_{ab} + \frac{\partial E_{total}}{\partial w_{ab}} \quad (17)$$

where w'_{ab} is the new weight between the two lays a and b.

B. Back propagation through time (BPTT)

The BP algorithm, which is a static learning algorithm, can be extended to explicitly consider the time, by using what is called "Unfolding" [1]. The extended algorithm is called

Back-propagation through the time (BPTT). When a very long sequence is used for training with BPTT, the unfolded partial gradients will vanish [23, 30]. BPTT is more preferred for training recurrent networks than BP, because the process of BPTT is a replication of the input and hidden neuron outputs for a number of previous time steps, including computing partial gradients. In this process, $\widetilde{h}_{js}(t-p)$ was used as a supplementary input. Partial gradient $LG_{js}(t)$ at time t is used to modify the weights from the additional input. Incorporating this with equation (12) the unfolding of the partial gradient can be expressed as:

$$LG_{js}t-p) = \sum_{l=1}^{n_h} LG_{ls}(t-p+1)u_{kl}^{p} f'(\widetilde{h}_{js}(t-p), \quad 2 \le p \le q \quad ..(18)$$

The MCRN trained with BPTT may not be more effective than the Elman network or even the Elman tower network. The reason for this is that the MCRN keeps a larger amount of historical information than the Elman network [1].

C. Dynamic Online Learning (DOL)

The DOL is basically a measure of the sensitivity of the value of the output of neuron k *at time* t to a small change in the value *of* w_{ab}, taking into account the effect of such a change in the weight over the entire network trajectory t_0 *to* T. It is also considered to be an on-line learning algorithm. It updates the weights at every time step, and can deal with sequences of arbitrary length and it needs less memory than BPTT.

The cost function is *def*ined by the sum of the mean square errors from output neurons at time t.

$$E(t) = \frac{1}{2}\sum_{k=1}^{n_{out}}\left(d_k(t) - O_k(t)\right)^2 = \frac{1}{2}\sum_{k=1}^{n_{out}}\left(e_k(t)\right)^2 .....(19)$$

where $e_k(t)$ is the difference $d_k(t) - O_k(t)$ between the target and its actual output of the kth output neuron. Given a sequence input data s, the network error for this sequence is

$$E_s = \sum_{t=1}^{T} E(t) .....(20)$$

and the total error for all the past input data can be written as

$$E_{total} = \sum_{s=1}^{S} E_s .....(21)$$

E_{total} is used to evaluate a trained network. BP and BPTT are used to minimise the error E_s as mentioned above. Unlike BP and BPTT, DOL is used to minimise the error $E(t)$ for this recurrent network. According to the chain rule, the estimation of a change weight is

$$\frac{\partial E(t)}{\partial w_{ab}(t)} = \frac{\partial E(t)}{\partial e_k(t)}\frac{\partial e_k(t)}{\partial O_k(t)}\frac{\partial O_k(t)}{\partial \widetilde{o}_k(t)}\frac{\partial \widetilde{o}_k(t)}{\partial w_{ab}(t)} .....(22)$$

where w_{ab} is a valid weight in the network. In order to simplify the expressions, the local gradients (also called partial gradients) are introduced in each layer, except the input layer. At the kth neuron of the output layer, its local gradient is following:

$$LG_k(t) = \frac{\partial E(t)}{\partial e_k(t)}\frac{\partial e_k(t)}{\partial O_k(t)}\frac{\partial O_k(t)}{\partial \widetilde{o}_k(t)}$$
$$= -e_k(t)\left(1 - f\left(\widetilde{o}_k(t)\right)\right) f\left(\widetilde{o}_k(t)\right) .....(23)$$

The local gradient at the jth neuron in the hidden layer is

$$LG_j(t) = \sum_{k=1}^{n_{out}} LG_k(t) w_{kj}(t) .....(24)$$

For a neuron l in the context layer p, the local gradient is calculated *as follows:*

$$LG_l^{\,p}(t) = \sum_{k=1}^{n_{out}} LG_k(t) woc_{kl}^{\,p} .....(25)$$

According to gradient descent method, the changed weights $\frac{\partial E(t)}{\partial w_{kj}(t)}$, $\frac{\partial E(t)}{\partial woc_{kl}^{p}(t)}$ $\frac{\partial E(t)}{\partial u_{jl}^{p}(t)}$ and $\frac{\partial E(t)}{\partial v_{ji}(t)}$ can be estimated. The above gradients will be used to simplify the equations. The detail of the processing

- Estimating of $\frac{\partial E(t)}{\partial w_{kj}(t)}$

According to formulae (22) and (23), $\frac{\partial E(t)}{\partial w_{kj}(t)}$ can be calculated as following:

$$\frac{\partial E(t)}{\partial w_{kj}(t)} = LG_k(t)\frac{\partial \widetilde{o}_k(t)}{\partial w_{kj}(t)} .....(26)$$

where

$$\frac{\partial \widetilde{o}_k(t)}{\partial w_{kj}(t)} = H_j(t) .....(27)$$

- Estimating of $\frac{\partial E(t)}{\partial woc_{kl}^{p}(t)}$

According to formulae (22) and (23), $\frac{\partial E(t)}{\partial woc_{kl}^{p}(t)}$ can be obtained by

$$\frac{\partial E(t)}{\partial woc_{kl}^{p}(t)} = LG_k(t)\frac{\partial \widetilde{o}_k(t)}{\partial woc_{kl}^{p}(t)} .....(28)$$

where

$$\frac{\partial \tilde{o}_k(t)}{\partial woc_{kl}^{p}(t)} = C_l(t-p) \text{.....} \text{.....} \text{.....} (29)$$

➢ Estimating of $\frac{\partial E(t)}{\partial u_{jl}^{p}(t)}$

According to the chain rule and formula (22), $\frac{\partial E(t)}{\partial u_{jl}^{p}(t)}$ can be expressed by

$$\frac{\partial E(t)}{\partial u_{jl}^{p}(t)} = \sum_{k=1}^{n_{out}} e_k(t) \frac{\partial E(t)}{\partial u_{jl}^{p}(t)} \text{.....} (30)$$

$$\frac{\partial E(t)}{\partial u_{jl}^{p}(t)} = \sum_{k=1}^{n_{out}} e_k(t) \frac{\partial e_k(t)}{\partial O_k(t)} \frac{\partial O_k(t)}{\partial \tilde{o}_k(t)} \frac{\partial \tilde{o}_k(t)}{\partial u_{jl}^{p}(t)} \text{.....} (31).$$

From equation (6), we can write

$$\frac{\partial \tilde{o}_k(t)}{\partial u_{jl}^{p}(t)} = \sum_{j'=1}^{n_h} \frac{\partial H_{j'}(t)}{\partial u_{jl}^{p}(t)} w_{kj'}(t) + \sum_{p'=1}^{q} \sum_{r=1}^{n_h} \frac{\partial H_r(t-p')}{\partial u_{jl}^{p}(t)} woc_{kr'}^{p'}(t) \text{.....} (32)$$

where $C_r(t-p')$ can be converted to $H_r(t-p')$ by time steps. According to equations (23), (24), (25), (28), (29), (28), (32) and (30), equation (32) can be written as

$$\frac{\partial E(t)}{\partial u_{jl}^{p}(t)} = -\sum_{k=1}^{n_{out}} e_k(t)\left(1 - f\left(\tilde{o}_k(t)\right)\right) f\left(\tilde{o}_k(t)\right) \times \left(\sum_{j'=1}^{n_h} \frac{\partial H_{j'}(t)}{\partial u_{jl}^{p}(t)} w_{kj'}(t) + \sum_{p'=1}^{q} \sum_{r=1}^{n_h} \frac{\partial H_r(t-p')}{\partial u_{jl}^{p}(t)} woc_{kr'}^{p}(t) \right)$$

$$= \sum_{j'=1}^{n_h} \left[LG_{j'}(t) \frac{\partial H_{j'}(t)}{\partial u_{jl}^{p}(t)} + \sum_{p'=1}^{q} \sum_{r=1}^{n_h} LG_r^{p'}(t) \delta_{rj'}(t) \frac{\partial H_{j'}(t-p')}{\partial u_{jl}^{p}(t)} \right] \text{....} (33)$$

where δ_{ab} is Kronecker delta. $\frac{\partial H_{j'}(t)}{\partial u_{jl}^{p}(t)}$ is clarified as

$$\frac{\partial H_{j'}(t)}{\partial u_{jl}^{p}(t)} = \frac{\partial H_{j'}(t)}{\partial \tilde{h}_{j'}(t)} \frac{\partial \tilde{h}_{j'}(t)}{\partial u_{jl}^{p}(t)}$$

$$= f'\left(\tilde{h}_{j'}(t)\right) \times \frac{\partial \left(\sum_{i=1}^{n_{in}} I_i(t) v_{j'i}(t) + \sum_{p'=1}^{q} \sum_{l'=1}^{n_h} C_{j'}(t-p') u_{j'l'}^{p'}(t) \right)}{\partial u_{jl}^{p}(t)}$$

$$= f'\left(\tilde{h}_{j'}(t)\right) \times \left(\delta_{j'j} \sum_{p''=1}^{q} \delta_{pp''} H_l(t-p'') + \sum_{p'=1}^{q} \sum_{j''=1}^{n_h} \sum_{l'=1}^{n_h} \frac{\partial H''(t-p'')}{\partial u_{jl}^{p}(t)} u_{j'l'}^{p'}(t) \right) \text{..} (34)$$

For the initial condition (time t=0), $\frac{\partial H_{j'}(0)}{\partial u_{jl}^{p}(0)}$ =0. By having these initial values, we can obtain all the values for $t \neq 0$ recursively.

➢ Estimating of $\frac{\partial E(t)}{\partial v_{ji}(t)}$

Using the same procedure, $\frac{\partial E(t)}{\partial v_{ji}(t)}$ can be obtained as follows:

$$\frac{\partial E(t)}{\partial v_{ji}(t)} = -\sum_{j'=1}^{m} \left[LG_{j'}(t) \frac{\partial H_{j'}(t)}{\partial v_{ji}(t)} + \sum_{p'=1}^{q} \sum_{r=1}^{n_h} LG_r^{p'} \delta_{lj'}(t) \frac{\partial H_{j'}(t-p')}{\partial u_{ji}^{p}(t)} \right] (35)$$

where

$$\frac{\partial H_{j'}(t)}{\partial v_{ji}(t)} = f'\left(\tilde{h}_{j'}(t)\right) \left(\delta_{j'j} I_i(t) + \sum_{p'=1}^{q} \sum_{j''=1}^{n_h} \sum_{l'=1}^{n_h} \delta_{j'j} \partial u_{j'l}^{p'}(t) \frac{\partial H''(t-p'')}{\partial v_{ji}(t)} \right) \text{...} (36)$$

For the initial condition (time t = 0), $\frac{\partial H_{j'}(0)}{\partial v_{ji}(0)}$ =0. Again, recursively, we can obtain all the other values.

Once the weight-changes are estimated, all the weights can be updated according to following equations:

$$w_{kj}(t+1) = \frac{\partial E(t)}{\partial w_{kj}(t)} + w_{kj}(t) \text{.....} (37)$$

$$woc_{kl}^{p}(t+1) = \frac{\partial E(t)}{\partial woc_{kl}^{p}(t)} + woc_{kl}^{p}(t) \text{.....} (38)$$

$$u_{jl}^{p}(t+1) = \frac{\partial E(t)}{\partial u_{jl}^{p}(t)} + u_{jl}^{p}(t) \text{.....} (39)$$

$$v_{ji}(t+1) = \frac{\partial E(t)}{\partial v_{ji}(t)} + v_{ji}(t) \quad (40)$$

The summary of the DOL algorithm is outlined as follows:

1) Starting from a time step t = 0, calculate the outputs of neurons for the hidden and output layers via equations (3) and (7) respectively; and compute the instantaneous error given by the equation (19).
2) Calculate the local gradients (partial gradients) according to the equations (26), (28), (30), (35), 30) and (35).
3) Update the weights using equations (37), (38), (39) and (40). Copy and update the activations of the context layers by using equations (4) and (5).
4) If t = T, (the maximum number of time steps) evaluate the network learning performance through equation (21). If the total error is less than or equal to the error threshold, then stop training, otherwise (continue to train the network) go to step1.

D. Activation Functions

Neural networks support a wide range of activation functions depending on the goal to be achieved. For instance the cost and error functions are directly dependent on the activation function and the network and learning algorithm parameter optimisation depends heavily on these functions. Furthermore, the selection of the activation function and cost functions depend on the application and also on the way the network inputs and outputs are coded. So far, only few function types are used by default as shown below, the others are available for customization [9]. The common assumptions of cost and activation functions for the BP and BPTT algorithms were described in [1]. The selection of activation and cost functions for MCRN, depends on the type of the problem to be solved. For example if the task output such as classification follows a probability distribution, then the logistic sigmoid function is more suitable and in accordance with equation (7).

$$O_k(t) = f(\tilde{o}_k(t)) = \frac{1}{1+e^{-\tilde{o}_k(t)}} \quad (41)$$

The cost function can be written as

$$E(t) = \frac{1}{2}\sum_{k=1}^{n_{out}} \left(d_k(t)\ln O_k(t) + (1-d_k(t))\ln(1-O_k(t))\right) \quad (42)$$

Therefore the local gradient can be written as equation (23).

If the problem is that of "1-of-n" classification, a multinomial distribution is appropriate. Therefore, the Softmax function is selected

$$f(\tilde{o}_k(t)) = \frac{e^{\tilde{o}_k(t)}}{\sum_l e^{\tilde{o}_l(t)}} \quad (43)$$

and cost function can be written as

$$E(t) = -\sum_k d_k(t)\ln O_k(t) \quad (44)$$

Therefore the local gradient in equation (23) has to be replaced by

$$LG_k(t) = d_k(t) - O_k(t) \quad (45)$$

E. Momentum technique

The gradient descent approach used to update the weight does not always converge to reasonably good solutions as it gets trapped into local minima [13]. To solve this problem a momentum factor was introduced. This will allow the network to ignore small features in the error surface of the cost function, and therefore, will improve the performance of the network. In addition, the momentum technique works well with the adaptive learning rate and speeds up the training phase. The generic equations for adding the momentum is

$$\Delta w' = \alpha \frac{\partial E}{\partial w} + \beta \Delta w \quad (47)$$

and the weights can be updated as

$$w'_{ab} = w_{ab} + \Delta w'_{ab} \quad (48)$$

α and β are the learning rate and momentum respectively.

IV. Complexity Structure of The MCRN

It is clear that a context layer in a recurrent network provides the potential for the network to store information about previous inputs. Usually, having more than one context layer improves the network accuracy, mainly on sequential tasks, due to more accurate information about the previous inputs. However, this will increase the network complexity in terms of the number connection-weights that should be calculated and updated per epoch. The goal of this section is, then to calculate the MCRN complexity and compare it to similar networks.

Let n be the number of input neurons, h the number of hidden neurons, and o the number of output neurons. Let c be the number of context layers. The number of weights and biases for the Elman tower network is given below. Note that the biases are used in the hidden and input layers only.

$$w_{ET}(o,h,n,c) = ch^2 + h(n+o) + h + n \quad (49)$$

Similarly, the number of weights and biases for MCRN is

$$w_{MCRNN}(o,h,n,c) = ch^2 + (c+1)ho + hn + h + n \quad (50)$$

The relationship between h and c for Elman network is given by the equations:

$$c = \frac{w_{ET} - n - h(o+n+1)}{h^2} \quad (51)$$

The relationship between h and c for MCRN is:

$$c = \frac{w_{MCRNN} - n - h(o+n+1)}{h^2 + ho} \quad (52)$$

For the same number of neurons in the input, hidden, and output layers and the same number of context layers, MCRN is more complex than the Elman network. However, for the same data set (application) MCRN can get better results with the same complexity as Elman Tower network. Therefore, to evaluate these networks, different numbers of context layers and hidden neurons were tested with the same or approximate number of connection-weights for both networks. This will allow us to optimise the MCRN network architecture and compare it to Elman network. In the following we will introduce the two applications used for testing and comparing these two networks.

V. Time Series Applications

A time series [23] is considered to be a progression of sequences of records, and each sequence of records represents a value of a particular feature. These features are observed at different times. Examples include stock market prices, currency exchange rates, the volume of product sales, biomedical measurements, weather data, etc. The time series problems can behave in four different ways; linearity, stationary, periodically, or randomly. A time series application can be represented as a linear problem when the future observation can be a linear function of the past observations, else it is represented as a nonlinear problem. A time series can be stationary when it has a constant mean and variance, else it is considered to be non-stationary, which means that the future observations cannot be foreseen and this is a very difficult problem and is hard to model. Time series can be periodic, which means that a time series with dominant periodic components will display regular periodic variations. Finally, time series can be a random noise problem, which means that there can be a random noise included in some parts or the entire frequency spectrum of the time series. There are several techniques implemented to handle time series applications. For instance, the simple moving average and exponential moving average techniques are used to deal with linear and stationary time series problems, the simple regression methods, auto regressive and auto regressive average deal with non-stationary time series problems, and decomposition methods deal with seasonal time series problems.

A difficulty with regression techniques is that the correlation between the component variables which affects the observation demand is not stationary but depends on spatial-temporal components. However, the regression techniques are not capable of tackling this chronological disparity. The time series techniques are a type of regression and therefore they have the same problem. The future observations are only a function of previous observations. This means that the absence of variables that are affecting future observations will result in the predictions being inaccurate and unstable [14, 26, 27, 31]. In this paper the MCRN is used to handle time series applications by employing non-linear modeling and adaptation. Suitable learning algorithms are chosen depending on the application. DOL is very powerful and uses less parameters and still can successfully carry out the task. However, BP is the fastest among the three learning algorithms that have been studied, but the results are not always as accurate as the others.

VI. Experimental Results

Three sets of experiments were carried out for the three time serial applications, which are shaping digit 8, electrical energy load forecasting and online handwriting symbol recognition. The MCRN and the Elman networks are both applied on these applications. Results of these two networks are given for each set of experiments.

A. *The shaping of the digit 8*

The shaping of the digit 8 is one of simplest time series application. However, it is very useful and common for testing a network's performance. 16 points in Cartesian coordinates (x, y) in the range [0 : 1] are selected to represent the shape of digit 8. The patterns are designed as follows: the target of an input pattern is the next input pattern, and the target of the last input pattern is the first input pattern. The central crossing point (0:5, 0:5) appears twice, depending on which direction the stroke takes. This crossing point creates an extra difficulty for the network as it has two different successors depending on the previous input pattern. Various network structures with different parameters are tested according to the computational complexity involved. After performing training sessions for a network, an arbitrary point in the data set including the crossing center point is selected and fed to the network. The output produced by the network is fed back again to the network. This process continues until the network draws the shape of the digit 8. Table 1 shows the three selected network's structures (Elman, Elman tower, and the MCRN networks). These structures are selected taking into account the complexity of computation. Note that all the networks are trained with DOL. We ran the three networks simultaneously on the same datasets. From Figure 3(a), we can notice that the MCRN outperformed the other two networks in terms of accuracy and learning time. It was also observed that the MCRN network training is stable (the error is decreasing steadily and smoothly). The Elman tower version was faster than the simple Elman network. This is due to enormous internal computations on a large number of hidden neurons.

TABLE I
SHOWS THE DIFFERENT NETWORK STRUCTURES IN ACCORDANCE WITH COMPLEXITY COMPUTATIONS

Elman Net	Elman Tower Net	MCRN
1cont. layer	6 cont. layers	6 cont. layers
20 hidden. units	9 hidden. Units	8 hidden. units
2 input units	2 input units	2 input units
2 output units	2 output units	2 output units
456 weights	533 weights	522 weights

Figure 4 (c), displays the shaping of digit 8 shaped by the MCRN. The MCRN shaping is more accurate than in (a) or (b), which are obtained shaped by the Elman tower and Elman Networks respectively.

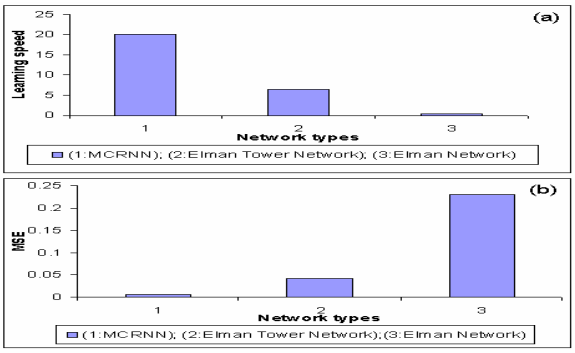

Fig. 3 (a) displays the learning speeds for the three networks, (b) displays the error performances of the three networks

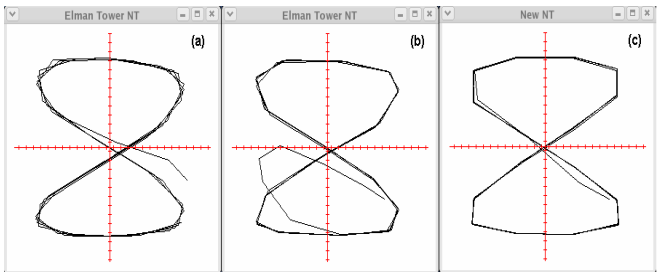

Fig. 4 (a) was shaped by the Elman tower with only one context layer (Elman network), (b) was shaped by the Elman tower network. (c) was shaped by the MCRN

B. *Electrical Energy load Forecasting*

The MCRN is used to handle the daily peak load forecasting for two different electrical power plants. Two different data sets (A and B) [4, 11, 21, 22, 31] were collected from two different countries. The inputs to the network are exogenous variables such as the previous and current change in the weather components, the previous and current status of the day and endogenous variables such as the past change in the loads. For data set (A) the future load is a function of the calendar, the status of the day (social events), the past and current change in the temperature T, and past change in the load L. The future load in the data set (B) is a function of the calendar, the status of the day, the past and current change in the weather components (such as temperate T, cloud rate C, wind speed W and humidity H), and past change in load L.

While the dynamic online learning is the most accurate, it is very time consuming due to the complexity of the computations. Nevertheless in this application and as long as the data sequence length is specified; the modified back propagation is driven to include the recurrent memory [27]. In this case the modified back propagation is efficient in terms of both response time and accuracy. Thus, by using the modified back propagation we are able to improve the performance of the dynamic learning process.

Both the MCRN and the Elman tower networks were trained with BP on data sets (A) and (B), both networks had an almost identical weight matrices. For data set (A) the structure of the Elman tower network was 12-5-3*5-1 (12 input neurons, 5 hidden neurons, 3 context layers each of which has 5 neurons, and 1 neuron output), whereas the structure for the MCRN was 12-4-3*4-1. For data set (B) the structure of the Elman tower network and MCRN were 18-8-3*8-1 and 18-7-3*7-1 respectively. We ran both networks on each data set simultaneously, and found that on each data set, the MCRN is faster. It performed more training cycles and had a smaller mean square error, as can be seen in Figure 5(a) and 5(b), for data set (A) and (B) respectively. The learning speed is defined as the number of cycles per minute that each network can complete. Experiments show that using both endogenous and exogenous variables as inputs to the MCRN rather than only using exogenous or endogenous variables as inputs to the network produces better results. Experiments also show that using the change in variables such as weather components and the change in the past load as inputs to the MCRN rather than the absolute values for the weather components and past load as inputs to the network has a dramatic impact on the prediction process and produces better accuracy [3, 31].

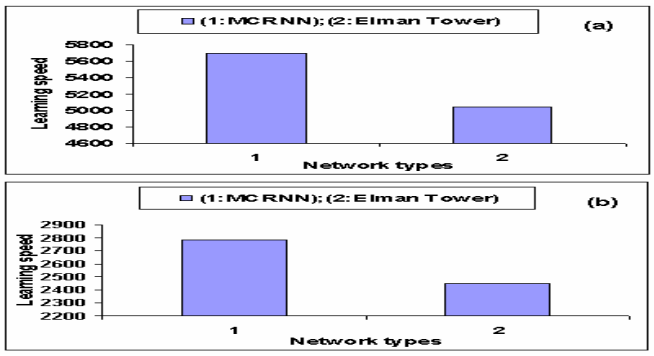

Fig. 5 (a) represents the learning speed when both networks run simultaneously on data set (A). (b) represents the learning speed when both networks run simultaneously on data set (A)

Table 2 represents the performance in terms of the maximum difference between the actual daily peak load and the forecasted daily peak load (MAP and MAX) and the training and testing mean square errors for both networks on each data set. Figures 6(a) and 6(b) represent the forecasting values predicted by both networks on data sets (A) and (B) respectively. One can notice that both networks performed reasonably well. Moreover, MCRN is 19:5% more efficient (in terms of prediction accuracy) than Elman tower network.

TABLE II
DISPLAYS THE ERROR PERFORMANCE OF MAP, MAX, MSE ON TRAINING SETS AND MSE AND TESTING SETS FOR BOTH NETWORKS RUN SIMULTANEOUSLY FOR EACH DATA SET

		Data Set A		
Network	MAP	MAX	Trainig MSE Eorror	Testing MSE Eorror
Elman Tower	1.76%	47.5	1.40E-04	0.098637476
MCRNN	1.62	31.11	1.34E-04	0.002109994
		Data Set B		
Network	MAP	MAX	Trainig MSE Eorror	Testing MSE Eorror
Elman Tower	2.26	233.04	8.74E-05	4.25E-05
MCRNN	2.16	196.9	7.98E-05	3.87E-05

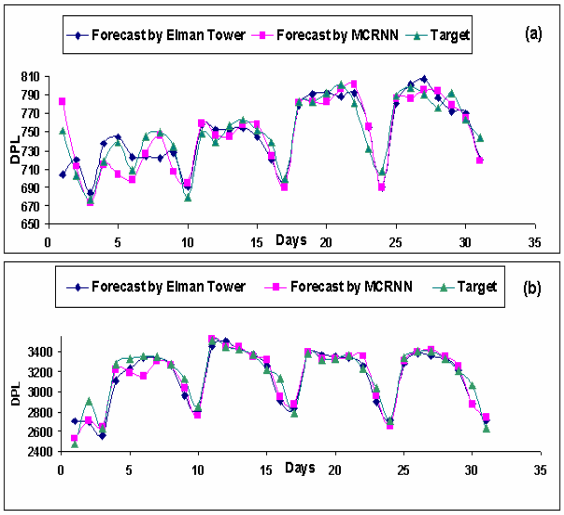

Fig. 6 (a) displays the forecasted results for both networks on the data set (A) (b) Displays the forecasted results for both networks on data set (B)

C. Recognition of online handwritten symbols

The MCRN is also evaluated using another real-world application in the area of handwriting recognition. Its performance is compared to the Elman tower network in recognizing digits from 0 to 9. The symbols are represented as sets of fuzzy feature shapes (c-shape, o-shape and line), orientation, length, x-Centre and y-Centre. These features were fed into the networks [18]. The same training and testing samples from section 1a in Unipen Train-R01/V07 [15] were used in both networks. 500 isolated digits from the Unipen dataset were used as the training data, and 1000 isolated digits were used for the testing. Approximately 750 weights were processed by each network, according to the theoretical model given in section 4. Therefore, the MCRN consists of 10 output neurons, 5 input neurons and 3 context layers, 10 hidden neurons. The Elman tower network consists of 10 output neurons, 5 input neurons, 3 context layers, and 13 hidden neurons. The learning rates and momentum for both networks are LR = 0.1 and MR = 0.02 respectively. Figure 7 shows that the error generated by the MCRN network decreases faster than the error generated by the Elman network. This means that MCRN is training faster. The recognition rates produced on the same dataset is of 79:4% for the Elman network and 89:5% for the MCRN network for the same amount of CPU time, which is about 30mn on a 2Ghz Pentium IV processor. This represented an efficiency improvement of 12.7%. As it was shown in [18], when MCRN is trained with larger datasets for longer periods the recognition rate will increase and the efficiency improvement can be around 20% compared to SRN tower networks. More details about using MCRN for handwriting recognition can be found in [18].

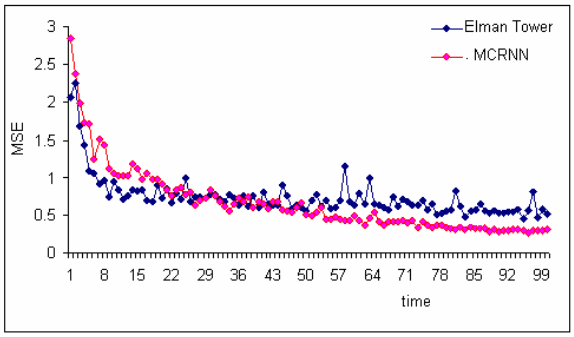

Fig. 7 displays the error performance of both networks: The Elman tower network and the multi context recurrent network

VII. Conclusion

SRN has been successfully used for time series applications analysis. They use a very simple architecture. However, it is strongly dependent on the hidden layer. Mainly, this affects these networks behavior in falling into local minima and their learning speed is very slow. Therefore, we proposed a new multi-context recurrent network (MCRN) to overcome these limitations. MCRN is constructed in such away as to inherit the advantages of SRN while overcoming their limitations. We have shown that this network trains much quicker. For the same network complexity, MCRN is about 20% more efficient than the Elman tower network. We have shown also that the MCRN network can be used with different learning algorithms depending on the given problem. We evaluated its performance on three completely different time series applications. Experimental results showed that MCRN performed better in every situation compared to other SRNs.

In this study, we showed also that some learning algorithms perform better than some others for some time series applications. In fact, the BPTT algorithm, which is theoretically more suitable for time series applications, is not delivering the expected results while used to train MCRN. This is due to the fact that the BPTT requires a very large memory that the context layers cannot satisfy. In the near future we will look at this problem in more details. We believe that the combination of the BPTT algorithm with Estimation-Maximization (EM) algorithm will solve this problem and speeds up the convergence of the MCRN network during the training phase.

References

[1] M. Boden, `A guide to recurrent neural networks and backpropagation',The DALLAS project. *Report from the NUTEK-supported project AIS-8, SICS.Holst: Application of Data Analysis with Learning Systems, (2001).*

[2] M.A. Castano, F. Casacuberta, and A. Bonet, Training Simple recurrent Networks Through Gradient Descent Algorithm, *volume 1240 of ISBN 3-540-63047-3, chapter Biological and Arti7 cial Computation: From Neuroscience to Technology, pp. 493--500, Eds. J. Mira and R. Moreno-Diaz and J. Cabestany, Springer Verlag, 1997.*

[3] W. Charytoniuk and M-S. Chen, `Very short-term load forecasting using neural networks', *IEEE Tran. On Power Systems, 15(1), 1558---1572, (2000).*

[4] B.J. Chen, M.W. Change, and C.J. Lin, `Eunite network competition: Electricity load forecasting', Technical report, In EUNITE 2001 symposium, a forecasting competition, (2001).

[5] Y. Cheng, T.W. Karjala, and D.M. Himmelblau, `Closed loop nonliner process identi┐ cation using internal recurrent nets', *In Neural Networks, 10(3), pp. 573--586, (1997*).

[6] A. Corradini and P. Cohen, `Multimodal speech-gesture interface for hands-free painting on virtual paper using partial recurrent neural networks for gesture recognition', *in Proc. of the Int'l Joint Conf. on Neural Networks (IJCNN'02), volume 3, pp. 2293--2298, (2002).*

[7] B. de Vries and J.C. Principe, `A theory for neural networks with time delays', in NIPS-3: *Proceedings of the 1990 conference on Advances in neural information processing systems 3, pp. 162--168, San Francisco, CA, USA, (1990). Morgan Kaufmann Publishers Inc.*

[8] Georg Dorffner, `Neural networks for time series processing', *Neural Network World, 6(4), pp. 447--468, (1996*).

[9] W. Duch and N.Jankowski, `Transfer functions: Hidden possibilities for better neural networks', *in ESANN'2001 proceedings European Symposium on Arti┐ cial Neural Networks, ISBN 2-930307 01-3, pp. 25-27, Belgium, (2001). D-Facto public.*

[10] J.L. Elman, `Finding structure in time', *Cognitive Science, 14(2), pp.179--211, (1999).*

[11] D. Esp, `Adaptive logic networks for east slovakian electrical load forecasting', *Technical report, In EUNITE 2001 symposium, a forecasting competition, (2001).*

[12] D.V. Prokhorov E.W. Saad and D.C. Wunsch, `Comparative study of stock trend prediction using time delay, recurrent and probabilistic neural networks', *IEEE Transactions on Neural Networks, 6(9), PP.1456--1470, (1998).*

[13] L. Fausett, Backpropagation Through Time and Derivative Adaptive Critics: A Common Framework for Comparison, *chapter Englewood Cliffs, NJ: Prentice Hall, 1994.*

[14] G. Gross and F. D. Galianan, `Short-term load forecasting', *In Proceedings of the IEEE., 75(12), 1558--1572, (1987).*

[15] I. Guyon, L. Schomaker, R. Plamondon, M. Liberman, and S. janet, `Unipen project of on-line data exchange and recognizer benchmarks', *in Proceedings of the 12th International Conference on Pattern Recognition, ICPR'94, pp. 29--33, Jerusalem, Israel, (October 1994).*

[16] S. Haykin, Neural Networks, A Comprehensive Foundation, *MacMillan Publishing Company, New York, 1994.*

[17] A. Herve and E. Betty, `Neural networks, quantitative applications', *in In the Social Sciences, volume 124, London: Sage Publications, (1999).*

[18] B. Q. Huang and M-T. Kechadi, `A recurrent neural network recogniser for online recognition of handwritten symbols.', *in ICEIS (2), pp. 27--34, (2005*).

[19] B. Q. Huang, T. Rashid, and T. Kechadi, `A new modi┐ ed network based on the elman network', *in Proceedings of IASTED International Conference on Arti┐ cial Intelligence and Application, ed., M. H. Hamza, volume 1 of ISBN: 088986-404-7, pp. 379--384, Innsbruck, Austria, (2004). ACTA Press.*

[20] M.I. Jordan, `Attractor dynamics and parallelism in a connectionist sequential machine.', *in Proceedings of the 8th Annual Conference of the Cognitive Science Society, Englewood Cliffs, NJ: Erlbaum, pp. 531--546. Reprinted in IEEE Tutorials Series, New York: IEEE Publishing Services, 1990, (1986).*

[21] I. King and J. Tindle, `Storage of half hourly electric metering data and forecasting with arti┐ cial neural network technology', *Technical report, In EUNITE 2001 symposium, a forecasting competition, (2001).*

[22] W. Kowalczyk, `Averaging and data enrichment: Two approaches to electricity load forecasting', *Technical report, In EUNITE 2001 symposium, a forecasting competition, (2001).*

[23] S. Lawrence, C.L. Giles, and S. Fong, `Natural language grammatical inference with recurrent neural networks', *IEEE Trans. on Knowledge and Data Engineering, 12(1), pp. 126--140, (2000).*

[24] A. Lo, H. Mamaysky, and J. Wang, `Foundations of technical analysis: Computational algorithms, statistical inference, and empirical implementation', *Journal of Finance 55, pp. 1705--1765, (2000*).

[25] Yee-Ling LU, Man-Wai MAK, and Wan-Chi SIU, `Application of a fast real time recurrent learning algorithm to text-to-phone conversion', *in Proceedings of the International Conference of Neural Networks, volume 5, pp. 2853--2857, (1995).*

[26] Simone Marinai, Marco Gori, and Giovanni Soda, `Arti┐ cial neural networks for document analysis and recognition.', *IEEE Trans. Pattern Anal. Mach. Intell., 27(1), pp. 23--35, (2005).*

[27] A. D. Papalxopoulos and T. C. Hiterbeg, `A regression-based approach to short-term load forecasting', *In IEEE Tran. On Power Systems, 4(1), pp. 1535--1547, (1990).*

[28] D. Park, M. El-Sharkawia, R. Marks, A. Atlas, and M. Damborg, `Electic load forecasting using arti┐ cial neural networks', *IEEE Trans. on Power Systems, 6(2), 442--449, (1991).*

[29] D. C. Plaut, `Semantic and associative priming in a distributed attractor network', *in Proceedings of the 17th Annual Conference of the Cognitive Science Society, pp. 37--42, Hillsdale, (1995). NJ:Erlbaum.*

[30] D. Prokhorov, Backpropagation Through Time and Derivative Adaptive Critics: A Common Framework for Comparison, *chapter Learning and Approximate Dynamic Programming, Wiley, 2004.*

[31] T. Rashid, B. Q. Huang, and T. Kechadi, `A new simple recurrent network with real-time recurrent learning process', *in The 14th Irish Artifcial Intelligence and Cognitive Science (AICS'03), ed., Padraig Cunningham, volume 1, pp. 169--174, Dublin, Ireland, (2003).*

[32] T. Rashid and T. Kechadi, `A practical approach for electricity load forecasting', *in The proceeding WEC'05, The ThirdWorld Enformatika, ed., C. Ardal, volume 5 of ISBN 975-98458-4-9, pp. 201--205, Isanbul, Turky, (2005). ACTA Press.*

[33] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, Learning Internal Representations by Error Propagation In D. E. Rumelhart, et al. (Eds.), *Parallel Distributed Processing: Explorations in the Microstructures of Cognition, 1: Foundations, MA: MIT Press, Cambridge, 1962.*

[34] M. Schnekel, I. Guyon, and D. Henderson, `On-line cursive script recognition using time delay networks and hidden markove models', *in Proc. ICASSP'94, volume 2, pp. 637--640, Adelaide, Australia, (April 1994).*

[35] P. Stagge and B. Sendho, `Organization of past states in recurrent neural networks: Implicit embedding', *in Proc. The Internation conference Computational Intelligence for Modelling, Control & Automation, pp. 21--27, Amsterdam, (1999). IOS Press.*

[36] J.C. Tomasz and M.Z. Jacek, `Neural network tools for stellar light prediction', *in Proc. of the IEEE Aerospace Conference, volume 3, pp. 415--422, Snowmass, Colorado, USA, (February 1997).*

[37] P. J. Werbos, `Backpropagation through time: What it does and how to do it', *in Proceedings of the IEEE, volume 78, pp. 1550--1560, (1990).*

[38] William H. Wilson, `Learning performance of networks like elman's simple recurrent netwroks but having multiple state vectors', *Workshop of the 7th Australian Conference on Neural Networks, Australian National University Canberra, (1996).*

[39] Shi XH, YC. Liang, and X. Xu, `An improved elman model and recurrent bck-propagation control neural networks', *Journal of Software, 6(14), 1110--1119, (2003).*