An Accelerating Learning Algorithm for Block-Diagonal Recurrent Neural Networks

Paris Mastorocostas, Dimitris Varsamis, Constantinos Mastorocostas, Ioannis Rekanos
Dpt. of Informatics, Technological Educational Institute of Serres, 62124, Serres, GREECE
mast@teiser.gr

Abstract

An efficient training method for block-diagonal recurrent neural networks is proposed. The method modifies the RPROP algorithm, originally developed for static models, in order to be applied to dynamic systems. A comparative analysis with a series of algorithms and recurrent models is given, indicating the effectiveness of the proposed learning approach.

1. Introduction

Recurrent neural networks have attracted considerable attention during the last decade. Due to their enhanced temporal capabilities, they have been extensively employed in several real world applications, including system identification, control, and temporal pattern recognition. The Diagonal Recurrent Neural Network (DRNN) [2] is a simplified form of the fully recurrent network (FRNN) [10] with no interlinks among neurons in the hidden layer. As shown in [2], DRNN is capable of controlling dynamic systems in the same way a FRNN does, requiring a considerably reduced parameter set compared to FRNN. A modified DRNN is the Block-Diagonal Recurrent Neural Network (BDRNN) [7], where dynamics is introduced between pairs of neurons in the hidden layer. The BDRNN developed in [7] is proved to be an efficient modelling tool as well.

The Back Propagation Through Time (BPTT) [4], an extension of the popular Back Propagation method to dynamic models, has been considered as an exclusive algorithm for training BDRNNs, due to the fact that it is a well established and easily applicable optimization method. However, the BPTT exhibits certain disadvantages, such as: a) it shows a low speed of convergence, b) most often it becomes trapped to local minima of the error surface.

In this perspective, the present work proposes the **M**odified **R**esilient Back**prop**agation (MRPROP) algorithm for training block-diagonal recurrent neural networks. The RPROP algorithm was originally developed in [5] for static networks and constitutes one of the best performing first order learning methods for neural networks, since it exhibits improved performance characteristics by remedying the drawbacks inherent to gradient descent learning [1]. The rest of this paper is organized as follows: In Section 2 the structure and characteristics of the BDRNN are illustrated. The learning algorithm is developed in Section 3. In the next section a simulation example is presented, in order to highlight the behaviour of MRPROP. The paper concludes with a brief discussion of the proposed method.

2. The block-diagonal recurrent neural network

The block-diagonal recurrent neural network is a two layer network, with the output layer being static and the hidden layer being dynamic. The hidden layer consists of pairs of neurons (blocks); there are feedback connections between the neurons of each pair, introducing dynamics to the network.

The configuration of BDRNN is presented in Fig. 1, where, for the sake of simplicity, a single–input–single–output BDRNN with four blocks of neurons is shown.

The operation of the BDRNN with m inputs, r outputs and N neurons at the hidden layer is described by the following set of state equations:

$$\boldsymbol{x}(k) = f_a\left(W \cdot \boldsymbol{x}(k-1) + B \cdot \boldsymbol{u}(k)\right) \quad (1a)$$

$$\boldsymbol{y}(k) = f_b\left(C \cdot \boldsymbol{x}(k)\right) \quad (1b)$$

where:

- f_a, f_b are the neuron activation functions of the hidden and the output layers, respectively. In the

Proceedings of the 2005 International Conference on Computational Intelligence for Modelling, Control and Automation, and International Confe
Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'05)
0-7695-2504-0/05 $20.00 © 2005 IEEE

following, the activation functions are both chosen to be the sigmoid function $f(z)=\frac{1-e^{-a_n \cdot z}}{1+e^{-a_n \cdot z}}$.

- $\boldsymbol{u}(k)=\left[u_i(k)\right]$ is a *m*–element vector, comprising the inputs of the network at time *k*.
- $\boldsymbol{x}(k)=\left[x_i(k)\right]$ is a *N*–element vector, comprising the outputs of the hidden layer. In particular, $x_i(k)$ is the output of the *i*-th hidden neuron at time *k*.
- $\boldsymbol{y}(k)=\left[y_i(k)\right]$ is a *r*–element vector, comprising the outputs of the network at time *k*.
- $B=\left[b_{i,j}\right]$ and $C=\left[c_{l,j}\right]$ are $N\times m$ and $r\times N$ input and output weight matrices, respectively.
- $W=\left[w_{i,j}\right]$ is the $N\times N$ block diagonal feedback matrix. In particular,

$$w_{i,j}=\begin{cases}\neq 0 & \text{if } i=j\\ \neq 0 & \text{if } i\neq j \text{ and } i=j-1 \text{ and } i \text{ is odd}\\ \neq 0 & \text{if } i\neq j \text{ and } i=j+1 \text{ and } i \text{ is even}\\ 0 & \text{otherwise}\end{cases}$$

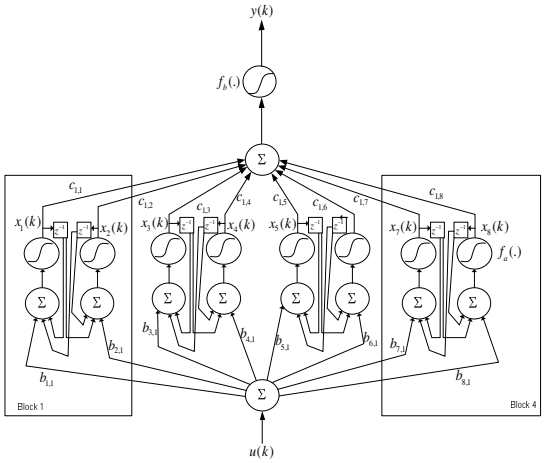

Figure 1.Configuration of BDRNN

The feedback matrix, *W*, is block diagonal: $W=diag\left[W^{(1)},...,W^{(N/2)}\right]$; each diagonal element, corresponding to a block of recurrent neurons, has a block submatrix in the form

$$W^{(i)}=\begin{bmatrix} w_{2i,2i} & w_{2i,2i+1}\\ w_{2i+1,2i} & w_{2i+1,2i+1}\end{bmatrix} \qquad i=1,2,...,\frac{N}{2} \qquad (2)$$

Equation (2) describes the general case of BDRNN, which is called *BDRNN with free-form submatrices*. A special case of BDRNN consists of *scaled orthogonal submatrices* in the form

$$W^{(i)}=\begin{bmatrix} w_{2i,2i} & w_{2i,2i+1}\\ -w_{2i,2i+1} & w_{2i,2i}\end{bmatrix}=\begin{bmatrix} w_i^{(1)} & w_i^{(2)}\\ -w_i^{(2)} & w_i^{(1)}\end{bmatrix} \qquad (3)$$

where $i=1,2,...,\frac{N}{2}$.

From (2) and (3) becomes evident that the Free-Form BDRNN consists of feedback submatrices with four distinct elements and provides a greater degree of freedom compared to the Scaled Orthogonal BDRNN, which has two weights at each feedback submatrix. Nevertheless, as discussed in [6], the latter network exhibits superior modeling capabilities than the Free-Form BDRNN, and the forthcoming learning method will be developed for this network. It should be noted that the method is general, applicable to the Free-Form BDRNN after a small number of modifications take place.

In view of the above, the state equations (1) for the Scaled Orthogonal BDRNN can take the following form:

$$x_{2i-1}(k)=f_a\Big(\sum_{j=1}^{m} b_{2i-1,j}\cdot u_j(k)+w_i^{(1)}\cdot x_{2i-1}(k-1)+$$

$$+w_i^{(2)}\cdot x_{2i}(k-1)\Big) \qquad i=1,...,\frac{N}{2} \qquad (4a)$$

$$x_{2i}(k)=f_a\Big(\sum_{j=1}^{m} b_{2i,j}\cdot u_j(k)-w_i^{(2)}\cdot x_{2i-1}(k-1)+$$

$$+w_i^{(1)}\cdot x_{2i}(k-1)\Big) \qquad i=1,...,\frac{N}{2} \qquad (4b)$$

$$y_l(k)=f_b\Big(\sum_{j=1}^{N} c_{l,j}\cdot x_j(k)\Big) \qquad l=1,...,r \qquad (4c)$$

where $w_i^{(1)}$, $w_i^{(2)}$ are the feedback weights at the hidden layer.

3. The MRPROP method

3.1. The algorithm

In gradient-based methods like BPTT the weight changes are proportional to the size of the gradient of an error function *E*:

$$\Delta w_i(t)=-\mu\frac{\partial^+ E(t)}{\partial w_i} \qquad (5)$$

where $\frac{\partial^+ E}{\partial w_i}$ is the *ordered partial derivative* of *E* with respect to a weight w_i, employed in recurrent models ([9]), *t* represents the epoch index and μ is the learning rate, which is kept fixed throughout the

learning process and is common to all weight updates.

Therefore, an appropriate selection of the learning rate is crucial to the evolution of the learning process and constitutes a significant constraint. The MRPROP attempts to alleviate this disadvantage of BPTT by allowing each fitting parameter to have its individual step size, which is adjusted during the learning process based on the sign of the respective partial derivative at the current and the previous epoch. Therefore, the effect of the adaptation process is not blurred by the influence of the size of the parameter gradient but is only dependent on the temporal behavior of the gradient ([5]).

Particularly, let $\frac{\partial^+ E(t)}{\partial w_i}$ and $\frac{\partial^+ E(t-1)}{\partial w_i}$ denote the ordered derivatives of E with respect to w_i at the present and the preceding epochs, respectively. MRPROP is described in pseudo-code as follows:

(a) For all weights w_i initialize the step sizes $\Delta_i^{(1)} = \Delta_0$

Repeat

(b) For all weights w_i compute the error gradient: $\frac{\partial^+ E(t)}{\partial w_i}$

(c) For all weights w_i, update step sizes:

(c.1) If $\frac{\partial^+ E(t)}{\partial w_i} \times \frac{\partial^+ E(t-1)}{\partial w_i} > 0$

then $\Delta_i^{(t)} = \min\left\{ \eta^+ \cdot \Delta_i^{(t-1)}, \Delta_{\max} \right\}$ (6)

(c.2) Else if $\frac{\partial^+ E(t)}{\partial w_i} \times \frac{\partial^+ E(t-1)}{\partial w_i} < 0$

then $\Delta_i^{(t)} = \max\left\{ \eta^- \cdot \Delta_i^{(t-1)}, \Delta_{\min} \right\}$ (7)

(c.3) Else $\Delta_i^{(t)} = \Delta_i^{(t-1)}$ (8)

(d) Update weights w_i: $\Delta w_i(t) = -sign\left(\frac{\partial^+ E(t)}{\partial w_i} \right) \cdot \Delta_i^{(t)}$ (9)

Until convergence

where the step sizes are bounded by $\Delta_{\min}$, $\Delta_{\max}$. The initial values of the step sizes $\Delta_i^{(1)} = \Delta_0$ are chosen rather moderately (e.g. 0.1), since these values directly determine the sizes of the first parameter changes. The increase and attenuation factors are set to $n^+ \in [1.01, 1.3]$ and $n^- \in [0.5, 0.9]$, respectively.

The adaptation mechanism described above has the advantage of correlating the step sizes not to the size of the derivatives but to their signs. Hence, whenever a parameter moves along a direction reducing E (the derivatives at successive epochs have the same sign), its step size is increasing independently of the size of the derivative. In this way, the step sizes can sufficiently increase when needed, even at the final stage of the learning process when the sizes of the derivatives are rather small. Additionally, when changes in the sign of the derivative occur, the step size is diminishing to prevent the error measure from oscillating.

3.2. Extraction of the error gradients

In the sequel, the MRPROP algorithm will be applied to the BDRNN, using as error measure the Mean Squared Error (MSE), defined by

$$E = \frac{1}{k_f} \sum_{k=1}^{k_f} \sum_{l=1}^{r} \left[y_l(k) - \hat{y}_l(k) \right]^2 \qquad (10)$$

where $y_l(k)$ is the l-th model output, $\hat{y}_l(k)$ is the l-th desired (actual) output of the system at time step k.

Due to the temporal relations existing in a dynamic system, the extraction of the ordered partial derivatives is not straightforward and is accomplished via a set of recursive equations.

In order to determine the error gradients of the dynamic part of BDRNN, let us introduce

a) the *state vector* $st(t)$, defined as:

$$\boldsymbol{st}(k) = \left[x_1(k), ..., x_N(k), y_1(k), ..., y_r(k) \right]^T \qquad (11)$$

comprising the outputs of the hidden and the output layer.

b) the *control vector* $\boldsymbol{\theta}$ comprising the synaptic and feedback weights ($N \times (m+r+1)$ weights)

$$\boldsymbol{\theta} = \left[b_{1,1}, ..., b_{N,m}, w_1^{(1)}, w_{\frac{N}{2}}^{(1)}, w_1^{(2)}, w_{\frac{N}{2}}^{(2)}, c_{1,1}, ..., c_{r,N} \right]^T \qquad (12)$$

For a data set including k_f pairs, the state equations are written

$$\boldsymbol{f}\left(\boldsymbol{st}(k), \boldsymbol{\theta}(k) \right) = \boldsymbol{0}, \quad k = 1, ..., k_f \qquad (13)$$

$f_{2i-1}^{(1)}(k)$ $i = 1, ..., \frac{N}{2}$:

$$f_a\left(\sum_{j=1}^{m} b_{2i-1,j} u_j(k) + w_i^{(1)} x_{2i-1}(k-1) + w_i^{(2)} x_{2i}(k-1) \right) -$$

$$- x_{2i-1}(k) = 0 \qquad (k_f \times \frac{N}{2} \text{ equations}) \qquad (14a)$$

$f_{2i}^{(1)}(k)$ $i = 1, ..., \frac{N}{2}$:

$$f_a\left(\sum_{j=1}^{m} b_{2i,j}u_j(k) - w_i^{(2)}x_{2i-1}(k-1) + w_i^{(1)}x_{2i}(k-1)\right) -$$

$$-x_{2i}(k) = 0 \qquad (k_f \times \frac{N}{2} \text{ equations}) \qquad (14b)$$

$f_l^{(2)}(k)$ $l = 1,...,r$:

$$f_b\left(\sum_{j=1}^{N} c_{l,j}x_j(k)\right) - y_l(k) = 0 \quad (k_f \times r \text{ equations}) \qquad (14c)$$

The error gradients are given by

$$\frac{\partial^+ E}{\partial \boldsymbol{\theta}} = \boldsymbol{\lambda}^T \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{\theta}} \qquad (15)$$

where the extraction of the Lagrange multipliers $\boldsymbol{\lambda}$ ([4]) is based on the formula

$$\frac{\partial E}{\partial \boldsymbol{x}} + \boldsymbol{\lambda}^T \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{\theta}} = \boldsymbol{0} \qquad (16)$$

After calculations are conducted in (16), the multipliers are determined through the following recursive equations:

$$\lambda_l^{(2)}(k) = \frac{1}{k_f} \cdot \left[y_l(k) - \hat{y}_l(k)\right] \qquad (17a)$$

$$\lambda_{2i-1}^{(1)}(k) = \frac{1}{k_f} \cdot \sum_{l=1}^{r}\left\{c_{l,2i-1} \cdot f_b'^{(l)}(k) \cdot \left[y_l(k) - \hat{y}_l(k)\right]\right\} +$$

$$+\sum_{l=1}^{r}\left\{\lambda_{El}^{(2)}(k) \cdot c_{l,2i-1} \cdot f_b'^{(l)}(k)\right\} + \lambda_{2i-1}^{(1)}(k+1) \cdot w_i^{(1)} \cdot f_a'^{(2i-1)}(k+1) -$$

$$-\lambda_{2i}^{(1)}(k+1) \cdot w_i^{(2)} \cdot f_a'^{(2i)}(k+1) \qquad (17b)$$

$$\lambda_{2i}^{(1)}(k) = \frac{1}{k_f} \cdot \sum_{l=1}^{r}\left\{c_{l,2i} \cdot f_b'^{(l)}(k) \cdot \left[y_l(k) - \hat{y}_l(k)\right]\right\} +$$

$$+\sum_{l=1}^{r}\left\{\lambda_l^{(2)}(k) \cdot c_{l,2i} \cdot f_b'^{(l)}(k)\right\} + \lambda_{2i-1}^{(1)}(k+1) \cdot w_i^{(2)} \cdot f_a'^{(2i-1)}(k+1) +$$

$$+\lambda_{2i}^{(1)}(k+1) \cdot w_i^{(1)} \cdot f_a'^{(2i)}(k+1)$$

(17c)

where $i = 1,...,\frac{N}{2}$, $l = 1,...,r$ and $f_b'^{(l)}(k)$, $f_1'^{(2i-1)}(k+j)$, $f_1'^{(2i)}(k+j)$ are the derivatives of $y_j(k+l)$ and $x_{2i-1}(k+j)$, $x_{2i}(k+j)$, respectively, with respect to their arguments. Equations (17) are backward difference equations that can be solved for $k = k_f, k_f - 1,...,1$ using the following boundary conditions:

$$\lambda_l^{(2)}(k_f) = \frac{1}{k_f} \cdot \left[y_l(k_f) - \hat{y}_l(k_f)\right] \qquad (18a)$$

$$\lambda_{2i-1}^{(1)}(k_f) = \frac{1}{k_f} \cdot \sum_{l=1}^{r}\left\{c_{l,2i-1} \cdot f_b'^{(l)}(k_f) \cdot \left[y_l(k_f) - \hat{y}_l(k_f)\right]\right\} +$$

$$+\sum_{l=1}^{r}\left\{\lambda_l^{(2)}(k_f) \cdot c_{l,2i-1} \cdot f_b'^{(l)}(k_f)\right\} \qquad (18b)$$

$$\lambda_{2i}^{(1)}(k_f) = \frac{1}{k_f} \cdot \sum_{l=1}^{r}\left\{c_{l,2i} \cdot f_b'^{(l)}(k_f) \cdot \left[y_l(k_f) - \hat{y}_l(k_f)\right]\right\} +$$

$$+\sum_{l=1}^{r}\left\{\lambda_l^{(2)}(k_f) \cdot c_{l,2i} \cdot f_b'^{(l)}(k_f)\right\} \qquad (18c)$$

Substituting (14) and (17) to (15), and taking into consideration (12), the error gradients are given by

$$\frac{\partial^+ E}{\partial c_{li}} = \sum_{k=1}^{k_f}\left\{\lambda_l^{(2)}(k) \cdot x_i(k) \cdot f_b'^{(l)}(k)\right\} \qquad (19a)$$

$$\frac{\partial^+ E}{\partial b_{ij}} = \sum_{k=1}^{k_f}\left\{\lambda_i^{(1)}(k) \cdot u_j(k) \cdot f_a'^{(i)}(k)\right\} \qquad (19b)$$

where $i = 1,...,N$, $j = 1,...,m$, $l = 1,...,r$.

$$\frac{\partial^+ E}{\partial w_i^{(1)}} = \sum_{k=1}^{k_f}\{\lambda_{2i-1}^{(1)}(k) \cdot x_{2i-1}(k-1) \cdot f_a'^{(2i-1)}(k) +$$

$$+\lambda_{2i}^{(1)}(k) \cdot x_{2i}(k-1) \cdot f_a'^{(2i)}(k)\} \qquad i = 1,...,\frac{N}{2} \qquad (19c)$$

$$\frac{\partial^+ E}{\partial w_i^{(2)}} = \sum_{k=1}^{k_f}\{\lambda_{2i-1}^{(1)}(k) \cdot x_{2i}(k-1) \cdot f_a'^{(2i-1)}(k) +$$

$$-\lambda_{2i}^{(1)}(k) \cdot x_{2i-1}(k-1) \cdot f_a'^{(2i)}(k)\} \qquad i = 1,...,\frac{N}{2} \qquad (19d)$$

4. Simulation results

In this section, extensive experimentation is carried out in an attempt to demonstrate the stabilizing properties of the MRPROP approach and compare its performance to other learning schemes. A benchmark problem is employed, the identification of a dynamical system. The example is taken from [3], where the plant to be identified is governed by the following difference equation

$$y_p(k+1) = \frac{y_p(k) \cdot y_p(k-1) \cdot y_p(k-2) \cdot u(k-1) \cdot \left[y_p(k-2) - 1\right] + u(k)}{1 + y_p^2(k-1) + y_p^2(k-2)} \qquad (20)$$

As it can be seen, the current output of the plant depends on three previous outputs and two previous inputs. A parallel identification system is considered, with the input $u(k)$ being the sole input to the network. The BDRNN used comprises four blocks of neurons in the hidden layer, resulting to a model with a total number of 32 weights.

In [7], the BDRNN is trained by use of BPTT. Our intention here is to compare the performance of the MRPROP method to the one reported in [7], in training of the BDRNN. A second objective of the experimentation is to test the level of accuracy and the generalization capabilities attained by the BDRNN and MRPROP in comparison to other recurrent models. In order to comply with previous results reported in the literature, a new training data containing ten batches of 900 patterns is generated. For each data batch, the input $u(k)$ is an independent and identically distributed uniform sequence for the first half of the 900 time steps and a sinusoid given by $1.05\sin(\pi k/45)$ for the rest of the time instants. The checking data set is composed of 1000 samples with a signal described by

$$u(k)=\begin{cases}\sin(\pi k/25) & k<250\\ 1 & 250\le k<500\\ -1 & 500\le k<750\\ 0.3\sin(\pi k/25)+0.1\sin(\pi k/32)+ & \\ +0.6\sin(\pi k/10) & 750\le k<1000\end{cases} \quad (21)$$

The experimental setup includes the MRPROP and the BPTT algorithms, both used to train a BDRNN with four weight blocks. The training parameters of MRPROP and BPTT are chosen as described in the sequel. For each method, several runs are performed with different parameter combinations. In those runs the network starts from the same initial weights. Then, the parameter combination is selected exhibiting the fastest convergence and low values of the error function. Following this approach, we are led to $\Delta_0 = 0.01$, $\eta^+ = 1.05$ and $\eta^- = 0.85$ for the MRPROP, and a learning rate of $\mu_1 = 0.032$ for BPTT, respectively. Next, a series of 100 independent trials are attempted, each with different weight initializations. Particularly, the feedback weights W and the weight matrices B and C are randomly selected within the range $[-0.5, 0.5]$. The slope pertaining to the activation functions of the network neurons is set to 2. For each particular trial and for fair comparison, the same weight initializations are used for MRPROP and BPTT.

Three different types of recurrent neural networks are also considered, namely, the IIR-MLP recurrent network [8], the diagonal recurrent neural network (DRNN) [2], and the memory neural network (MNN) [6]. The IIR-MLP is a multilayered network where the synaptic connections are implemented through IIR filters, including a moving average (MA) and an auto-regressive (AR) part. We selected a 1x8x1 IIR-MLP model with unit delays in the MA and the AR parts, both for the input-to-hidden and the hidden-to-output synaptic filters, respectively. The DRNN is a modified form of the fully recurrent neural network with one hidden layer containing self-recurrent neurons. In our simulations a 1x8x1 DRNN model is selected. The weights of the IIR-MLP and DRNN are randomly initialized in that range [-0.5,0.5]. Following a similar approach as above, the learning rate is set to 0.01, an optimal value decided after an initial experimentation for several training runs. Training of the IIR-MLP and DRNN models is accomplished by means of the BPTT algorithm while MNN is trained using the real time recurrent learning (RTRL) method [4].

All network models and learning schemes are trained following a parallel mode approach, with the exception of the MNN where the series-parallel configuration is adopted, as reported in [6]. Table I hosts the comparative results attained after five training epochs of the entire data set, with the weight updates taking place at the end of each one of the ten batches. The results are averaged over 100 independent runs with different weight initializations. The results for the MNN are taken from [6]. The effectiveness of the training process is measured in terms of the average and the standard deviation of the MSE over the checking data set. From the results in Table I it can be seen that the BDRNN trained by the MRPROP method exhibits the best performance among the competing schemes, with regard to both the average and, especially, the standard deviation of the checking error. The former criterion indicates the accuracy of the MRPROP algorithm while the later one shows its robustness to weight initializations. The BPTT scheme for the BDRNN is considerably inferior to the MRPROP method regarding the accuracy and the generalization property. Furthermore it has an error standard deviation almost twice as large compared to the one attained by MRPROP, leading to the conclusion that MRPROP accelerates the learning process for the BDRNN significantly, while exhibiting insensitivity to initial weight settings.

It should be mentioned that, even though the MRPROP algorithm has been developed for Scaled Orthogonal BDRNN, it can be easily modified in order to be applicable to Free-Form BDRNN. The mere difference in the latter case is the existence of four tunable feedback weights at each block of neurons of the hidden layer.

5. Conclusions

A new learning algorithm for training recurrent systems has been proposed, entitled Modified Resilient Backpropagation. The method has modified the standard RPROP algorithm such that it can be applied to dynamic models and particularly to Block-Diagonal Recurrent Neural Networks, by taking into account the temporal relations existing in a dynamic system. The proposed learning scheme has been applied to an identification problem and a comparative analysis with a series of algorithms and recurrent networks has been conducted, underlining the effectiveness of MRPROP.

Table 1. Comparative results between the suggested MRPROP algorithm, the BPTT and other recurrent network types for the identification problem. Results are derived from 100 independent trials.

Network type	Training method	Checking MSE Avg	Checking MSE StD	No. of weights
BDRNN	MRPROP	0.0305	0.0037	32
BDRNN	BPTT	0.0368	0.0060	32
IIR-MLP	BPTT	0.0303	0.0409	32
DRNN	BPTT	0.0287	0.0118	33
MNN	RTRL	0.0752	-	81

Acknowledgement

This work was supported in part by the European research project Archimedes II.

References

[1] C. Igel and H. Husken, "Empirical Evaluation of the Improved RPROP Learning Algorithms," *Neurocomputing*, Elsevier, 2003, pp. 105-123.

[2] C.-C. Ku and K.Y. Lee, "Diagonal Recurrent Neural Networks for Dynamic Systems Control," *IEEE Trans. Neural Networks*, IEEE, January 1995, pp. 144-156.

[3] K.S. Narendra and K. Parthasarathy, "Identification and Control of Dynamical Systems using Neural Networks," *IEEE Trans. Neural Networks*, IEEE, March 1990, pp. 4-27 1990.

[4] S. Piche, "Steepest Descent Algorithms for Neural Network Controllers and Filters," *IEEE Trans. Neural Networks*, IEEE, March 1994, pp. 198-221.

[5] M. Riedmiller and H. Braun, "A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm," *Proc. IEEE Int. Joint Conf. on Neural Networks*, IEEE, 1993, pp. 586-591.

[6] P.S. Sastry, G. Santharam, and K.P. Unnikrishnan, "Memory Neuron Networks for Identification and Control of Dynamical Systems," *IEEE Trans. Neural Networks*, IEEE, March 1994, pp. 306-319.

[7] S.C. Sivakumar, W. Robertson, and W.J. Phillips, "On-Line Stabilization of Block-Diagonal Recurrent Neural Networks," *IEEE Trans. Neural Networks*, IEEE, January 1999, pp. 167-175.

[8] A.C. Tsoi and A.D. Back, "Locally Recurrent Globally Feedforward Networks: A Critical Review of Architectures," *IEEE Trans. Neural Networks*, IEEE, March 1994, pp. 229-239.

[9] P. Werbos, *Beyond Regression: new tools for prediction and analysis in the behavioral sciences*, Ph.D. thesis, Harvard Univ., Cambridge, MA, 1974.

[10] R.J. Williams and D. Zisper, "A Learning Algorithm for Continually Running Fully Recurrent Neural Networks," *Neural Computation*, MIT, 1989, pp. 270-280.