New Second-Order Algorithms for Recurrent Neural Networks Based on Conjugate Gradient

Paolo Campolucci[*], *Michele Simonetti, Aurelio Uncini, Francesco Piazza*[+]
Dipartimento di Elettronica ed Automatica - Università di Ancona, Italy.
[*] E-mail: paolo@eealab.unian.it - Internet: http://nnsp.eealab.unian.it/

Abstract

In this paper we derive two second-order algorithms, based on conjugate gradient, for on-line training of recurrent neural networks. These algorithms use two different techniques to extract second-order information on the Hessian matrix without calculating or storing it and without making numerical approximations. Several simulation results for non-linear system identification tests by locally recurrent neural networks are reported for both the off-line and on-line case, comparing with first-order algorithms and showing faster training.

1. Introduction

Several learning algorithms for neural networks have been proposed in literature and many of them are based on the gradient descend algorithm well known in optimization theory. They usually have poor convergence rate and so are not suited for some DSP problems, especially for on-line computations.

Second-order algorithms have instead better performances because they also use the second-order information stored in the Hessian matrix. There are several examples of this algorithms in literature [3,4], but a sub-class of them, based on the conjugate gradient method, has shown good properties in terms of rate of convergence and computational complexity.

Conjugate directions methods are based on choosing the search direction and the step size of a minimization formula by using second order information. It holds

$$E(\mathbf{w}+\mathbf{y}) \approx E(\mathbf{w}) + \nabla_w E(\mathbf{w})^T \mathbf{y} + \frac{1}{2}\mathbf{y}^T \mathbf{H}(\mathbf{w})\mathbf{y} \qquad (1.1)$$

where $E(\mathbf{w})$ is a generic cost function, of weights $\mathbf{w}$, to be minimized, $\mathbf{H}(\mathbf{w})$ is the Hessian, and $\mathbf{y}$ the weight variation.

[+] This research was supported by the Italian MURST.

The Conjugate Gradient (CG) algorithm is based on the following two iterative formulas respectively for updating weights $\mathbf{w}_k$ and conjugate directions $\mathbf{p}_k$ [9]:

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \frac{\mathbf{p}_k^T \nabla E(\mathbf{w}_k)}{\mathbf{p}_k^T \mathbf{H}(\mathbf{w}_k)\mathbf{p}_k}\,\mathbf{p}_k \qquad (1.2)$$

$$\mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \frac{|\mathbf{r}_{k+1}|^2 - \mathbf{r}_{k+1}^T \mathbf{r}_k}{\mathbf{p}_k^T \mathbf{r}_k}\,\mathbf{p}_k \qquad (1.3)$$

where k is the iteration index and $\mathbf{r}_k = -\nabla E(\mathbf{w}_k)$.

This algorithm has two drawbacks. The first is that for each iteration the Hessian matrix $\mathbf{H}(\mathbf{w}_k)$ has to be calculated and stored; the second one is that this algorithm only works for functions with positive definite Hessian matrices, and the quadratic approximations can be very poor when the current point is far from a local minimum.

Moller has proposed a solution to the second problem based on the Levenberg-Marquardt algorithm combined with the conjugate gradient approach. The problem of the Hessian matrix definition is solved trying to make always positive the quantity in the denominator of (1.2), adding a positive term, which is determined recursively. This algorithm is called Scaled Conjugate Gradient (SCG) [10] and results to be better than CG in terms of convergence properties.

With respect to the first problem some methods exist to extract information on the Hessian matrix without calculating or storing it and without making numerical approximations.

From (1.2) we note that the algorithm needs to calculate the vector $\mathbf{H}(\mathbf{w}_k)\mathbf{p}_k$, and not only the matrix $\mathbf{H}(\mathbf{w}_k)$. Therefore, having an efficient technique to directly obtain the product $\mathbf{H}(\mathbf{w}_k)\mathbf{p}_k$ there is no need for calculating and storing the whole Hessian matrix.

For this reason in this work two methods for calculating such product have been applied to SCG,

obtaining two computationally efficient algorithms, with good speed of convergence but without the complexity and memory usage typical of second order methods already known in literature.

The methods derived have been applied to the difficult problem of training recurrent neural networks both in off-line and on-line mode. Several simulation results for non-linear system identification tests from literature by locally recurrent neural networks are reported for both the off-line and on-line case, comparing with first order algorithms and showing a faster training.

The complexity of the methods proposed is accurately compared with that of the corresponding first order algorithms showing an average increase of about 2-3 times in terms of number of operations per iteration.

2. Product calculation techniques

The first method [11] allows calculating the product **Hp**, where **p** is a generic vector and **H** is the Hessian matrix, by a simple, accurate and numerically robust technique using the following operator

$$R_p\{f(\mathbf{w})\} \equiv \frac{\partial}{\partial r} f(\mathbf{w}+r\mathbf{p})\bigg|_{r=0} \tag{2.1}$$

that gives

$$R_p\{\nabla E(\mathbf{w})\} \equiv \frac{\partial}{\partial r} \nabla E(\mathbf{w}+r\mathbf{p})\bigg|_{r=0} = \mathbf{H}(\mathbf{w})\mathbf{p} \tag{2.2}$$

The previous formula uses the gradient of the cost function E to calculate the vector **H(w)p** therefore the computational complexity is reduced with respect to the direct use of second order derivatives.

Because $R_p\{\}$ is a differential operator, it obeys to the usual rules for differential operators, in particular

$$R_p\{cf(w)\} = cR_p\{f(w)\}$$
$$R_p\{f(w)+g(w)\} = R_p\{f(w)\}+R_p\{g(w)\}$$
$$R_p\{f(w)g(w)\} = R_p\{f(w)\}g(w)+f(w)R_p\{g(w)\}$$

Moreover we can also write [11]

$$R_p\{\mathbf{w}\} = \mathbf{p}. \tag{2.3}$$

These rules are sufficient to derive a new set of equations on a new set of variables called R-variables, from the equations used to compute the gradient. This can be thought of as an adjoint system to the gradient calculation that computes the vector $R_p\{\nabla E(\mathbf{w})\}$, which is the desired vector **H(w)p**.

The second method [5] performs the calculation of **H(w)p,** where **p** is now the conjugate gradient and **H** the Hessian matrix, as a difference between two gradients, computed in two different points.

This technique has four main phases:

1. compute the gradient $\nabla_w E(\mathbf{w})$ of the cost function with respect to the weights vector **w** ;
1. compute $\mathbf{u} = \mathbf{w} - \nabla_w E(\mathbf{w})$;
1. compute the gradient $\nabla_u E(\mathbf{u})$ of the cost function with respect to the vector **u** ;
1. compute the quantity
$$\mathbf{H}(\mathbf{w})\mathbf{p} = \nabla_w E(\mathbf{w}) - \nabla_u E(\mathbf{u}) \tag{2.4}$$
as a difference between the two gradients calculated before.

Must be stressed that both the two methods above must be performed every iteration, i.e. every step of conjugate gradient descend.

But while the first one gives an exact computation of the product **H(w)p**, the second one gives only an approximation of it because $E(\mathbf{w})$ generally isn't a quadratic cost function.

The two previous techniques result to be simple and efficient and can be applied to any neural network (static and dynamic). They allow exploiting the Hessian properties without calculating them explicitly, and using only first order formulas.

The second case, moreover, has also an important advantage when implementing it in software, because it uses directly the same formulas of the gradient calculation.

Must also be stressed that the two previous techniques have never been applied to training of neural networks by the SCG algorithm, particularly the second has been implemented only for adaptive filtering. The SCG with either of the two techniques discussed above gives two new algorithms which are much faster than the first order counterpart, as shown in the section on simulation results.

After using these algorithms with some static problems, with good performances, we have applied them to training locally recurrent neural networks, as explained in the next section.

3. Locally recurrent neural networks

Dynamic recurrent neural networks have been recently attracting great attention from the signal processing community because they are really useful for

DSP application, system identification and spazio-temporal pattern recognition.

Recurrent neural networks can provide better modeling accuracy compared to buffered static Multi Layer Perceptron (MLP) or MLP with Finite Impulse Response (FIR) filter synapses, often known as Time Delay Neural Networks (TDNN) even if these networks are also used for simplicity. In fact for practical modeling problems where the temporal dependence of the outputs with respect to the inputs is not very short, some kind of feedback should be used.

Fully recurrent networks are general but difficult to train. Especially for DSP problems, for which, in the case of stability, a forgetting behavior is usually required, the MLP with Infinite Impulse Response synapses (IIR-MLP) [1] can exhibit better capabilities, due to the prewired forgetting behavior (typical of locally recurrent networks). In fact IIR-MLP can be considered as a non-linear extension of the linear adaptive IIR filter.

Recently Campolucci et al. [8] have proposed a new learning algorithm for locally recurrent neural networks, called Truncated Recursive Back Propagation (TRBP), which is an on-line extension of the Recursive Back Propagation (RBP) algorithm [6,7], and which has good performances. This algorithm however is based on gradient calculation, i.e. on steepest descend, making use of first order information only.

Therefore in this paper the two new algorithms called SCG-R and SCG-u, that apply SCG and respectively the two techniques described above, are presented referring to locally recurrent neural networks, comparing their performances with RBP and TRBP. Although for the sake of brevity we have concentrated our work only for IIR-MLP networks, it can be extended to any locally recurrent networks.

4. SCG-*R* and SCG-*u* algorithms for IIR-MLP neural networks

With the same notation used in [6,7,8], let consider the generic neuron k in the layer l of a IIR-MLP neural network with inputs x, weights w and targets d, trained by an epoch of T learning patterns. The vector of all the weights of a whole IIR-MLP network is given by

$$\tilde{\mathbf{w}} = \begin{bmatrix} \mathbf{w} \\ \mathbf{v} \end{bmatrix} \qquad \text{where}$$

$$\mathbf{w} = \left[\mathbf{w}^{(1)} \dots \mathbf{w}^{(l)} \dots \mathbf{w}^{(M)} \right]^T$$

$$\mathbf{w}^{(l)} = \left[w^{(l)}_{10(0)} \dots w^{(l)}_{kj(p)} \dots w^{(l)}_{N_{l+1}N_l(L^{(l)}_{Nl+1,Nl}-1)} \right]$$

$$\mathbf{v} = \left[\mathbf{v}^{(1)} \dots \mathbf{v}^{(l)} \dots \mathbf{v}^{(M)} \right]^T$$

$$\mathbf{v}^{(l)} = \left[v^{(l)}_{10(0)} \dots v^{(l)}_{kj(p)} \dots v^{(l)}_{N_{l+1}N_l(I^{(l)}_{Nl+1,Nl})} \right].$$

Therefore the generic element of conjugate direction, because of its definition, will depend on index k, j and l, giving the following vector

$$\tilde{\mathbf{p}} = \begin{bmatrix} \mathbf{pw} \\ \mathbf{pv} \end{bmatrix} \qquad \text{where}$$

$$\mathbf{pw} = \left[\mathbf{pw}^{(1)} \dots \mathbf{pw}^{(l)} \dots \mathbf{pw}^{(M)} \right]^T$$

$$\mathbf{pv} = \left[\mathbf{pv}^{(1)} \dots \mathbf{pv}^{(l)} \dots \mathbf{pv}^{(M)} \right]^T$$

$$\mathbf{pw}^{(l)} = \left[pw^{(l)}_{10(0)} \dots pw^{(l)}_{kj(p)} \dots pw^{(l)}_{N_{l+1}N_l(L^{(l)}_{Nl+1,Nl}-1)} \right]$$

$$\mathbf{pv}^{(l)} = \left[pv^{(l)}_{10(0)} \dots pv^{(l)}_{kj(p)} \dots pv^{(l)}_{N_{l+1}N_l(I^{(l)}_{Nl+1,Nl})} \right]$$

As stated above, applying the two previous techniques to calculate the products $\mathbf{H}(\tilde{\mathbf{w}})\tilde{\mathbf{p}}$, we obtain two new algorithms based on the conjugate gradient method. These algorithms contain only first order formulas, i.e. gradient calculation. A critical point is to use an efficient gradient calculation for locally recurrent networks, to this purpose we have chosen the RBP and TRBP algorithms mentioned above.

In the next sections we show the batch and on-line versions of two new second-order algorithms for training this kind of neural networks.

5. SCG-*R* algorithm

Applying the $R_{\tilde{p}}$ operator first to the forward and then to the backward phase of RBP algorithm [6,7] we have a set of formulas that represents the SCG-R algorithm in batch mode.

Because of $\tilde{\mathbf{w}}$ and $\tilde{\mathbf{p}}$ definition, we can then write

$$R_{\tilde{p}}\left\{ \frac{\partial E}{\partial w^{(l)}_{nm(p)}} \right\} = \left[R_{\tilde{p}} \left\{ \nabla E(\tilde{\mathbf{w}}) \right\} \right]_{nm(p)^{(l)}} = \left[\mathbf{H}(\tilde{\mathbf{w}})\tilde{\mathbf{p}} \right]_{nm(p)^{(l)}} \tag{5.1}$$

As stated above this is a batch mode algorithm that can't work on-line [6], but we can obtain an on-line

version of SCG-*R* simply applying the $R_{\tilde{p}}$ operator to the TRBP algorithm.

The forward phase is the same as in the previous case. Applying the $R_{\tilde{p}}$ operator to the backward phase we have the formulas shown in the Appendix.

We note that in practice the computational complexity and the memory usage of SCG-*R* is about 2 times RBP (or TRBP), because we must compute and store the adjoint $R_{\tilde{p}}\{.\}$ terms. The entire algorithm is performed using only first order formulas.

6. SCG-*u* algorithm

In this case we calculate the products $\mathbf{H}(\tilde{\mathbf{w}})\tilde{\mathbf{p}}$ using the following expression

$$\mathbf{H}(\tilde{\mathbf{w}})\tilde{\mathbf{p}} = \nabla_{\tilde{w}} E(\tilde{\mathbf{w}}) - \nabla_{\tilde{u}} E(\tilde{u}) \tag{6.1}$$

where $\tilde{\mathrm{u}} = \tilde{\mathbf{w}} - \nabla_{\tilde{w}} E(\tilde{\mathbf{w}})$.

The crucial point of the $\nabla_{\tilde{u}} E(\tilde{\mathbf{u}})$ computation is given by also applying RBP (or TRBP for on-line mode) to a second network that has the same structure of the first, but with weights vector $\tilde{\mathbf{u}}$ instead of $\tilde{\mathbf{w}}$.

Therefore we have the same formulas of RBP, or TRBP, but we applied them to a set of different variables that we call *u*-variables.

Computational complexity and memory storage of the SCG-*u* are about double with respect to RBP (or TRBP) algorithm, because in practice we performed RBP (or TRBP) two times for each iteration: first to calculate $\nabla_{\tilde{w}} E(\tilde{\mathbf{w}})$ and then to calculate $\nabla_{\tilde{u}} E(\tilde{\mathbf{u}})$.

We have omitted the SCG-*u* formulas because they are the same as for RBP (or TRBP in the on-line case).

7. Simulations results

In this section the results of applying the new SCG-*R* and SCG-*u* algorithms to two problems of identification of non-linear dynamic systems from literature are presented.

The locally recurrent architecture chosen for the simulations is the IIR-MLP with two layers: three hidden neurons with hyperbolic tangent activation function and one linear output neuron.

Both for batch and for on-line mode we compare SCG with first-order algorithms, respectively RBP and TRBP. The results are given in terms of Mean-Square-Error (MSE) expressed in dB, and its variance, computed on the learning set after each epoch (after all the input-output samples were presented) and averaged over 20 runs, each with a different weight initialization.

The first set of experiments consisted in identifying the non-linear system with memory presented in [2]. The network used had both MA and AR order equal to three for each neuron. From Fig. 1-2 is clear that the new SCG algorithms have good performances.

The second set of experiments was carried out on the more realistic problem of identifying a baseband equivalent Pulse Amplitude Modulation (PAM) transmission system in presence of non linearity, see [6,7] for details. The network used in this case had MA and AR order respectly equal to four and two for each neuron. Fig. 3-4 shows that the new SCG algorithms perform very well.

In these simulation the second-order algorithms seem to perform better than first-order ones. Moreover the complexity of SCG algorithms, in terms of number of floating points operations (flop) per iteration, is only about 2 times higher with respect to RBP and TRBP, as shown in Tab.1-2. This is an important result because the other second-order algorithms already presented in literature have in general much higher complexity.

References

[1] Tsoi A.C. and Back A.D., "*Locally recurrent globally feedforward networks: a critical review of architectures*", IEEE Transactions on Neural Networks, vol. 5, no. 2, 229-239, March 1994.

[2] Back A.D., Tsoi A.C., "A simplified gradient algorithm for IIR-MLP synapse Multilayer Perceptrons", Neural Computation 5, pp.456-462, 1993.

[3] Battiti R., "*First- and Second-Order Methods for Learning : Between Steepest Descend and Newton's Method*", Neural Computation, 4, 1992.

[4] Bishop C., "*Exact calculation of the Hessian matrix for the multilayer perceptron*", Neural Computation 4(4), 1992.

[5] Boray G.K., Srinath M.D., "*Conjugate Gradient Tecniques for Adaptive Filtering*", IEEE Transaction on Circuits and Systems-I :Fundamental Theory and applications, Vol. 39, No. 1, Jan. 1992.

[6] Campolucci P., Piazza F., Uncini A., "*On-line learning algorithms for neural networks with IIR synapses*", Proc. of the IEEE International Conference of Neural Networks, Perth, Nov. 1995.

[7] Campolucci P., Uncini A., Piazza F., "A Unifying View of Gradient Calculations and Learning for Locally Recurrent Neural Networks" Proc. of WIRN97, Italian Workshop on Neural Networks, Vietri sul Mare (Italy), Spriger-Verlag ed., May 1997.

[8] Campolucci P., Uncini A., Piazza F., *"A new IIR-MLP learning algorithm for on-line signal processing"*, International Conference of Acoustic Speech and Signal Processing (ICASSP97), Munich, April 1997.

[9] Fletcher R. and Reeves C.M., *"Function minimization by conjugate gradients"*, The computer Journal, 1964.

[10] Möller M., *"A scaled conjugate gradient algorithm for fast supervised learning"*, Neural Networks 6(4) , 1993.

[11] Pearlmutter B.A., *"Fast exact multiplication by the Hessian"*, Neural Computation 6:147-160, 1994.

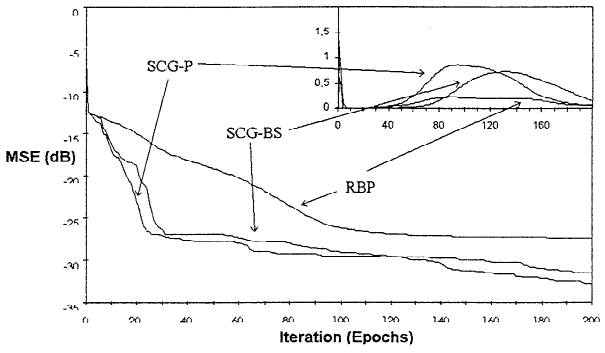

Fig. 1. Back-Tsoi test results in batch mode: MSE and its variance.

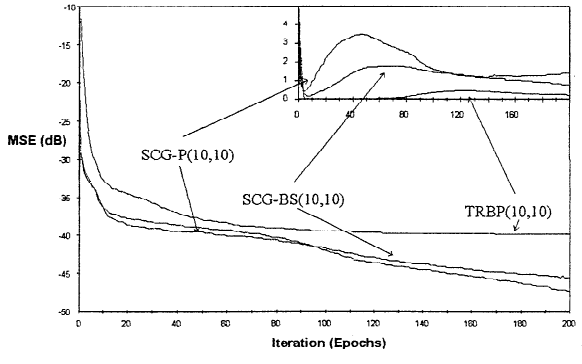

Fig. 2. Back-Tsoi test results in on-line mode, using SCG-{*R,u*} (*h,h'*) or TRBP(*h,h'*). *h* is the past history length considered and *h'* is every how many samples parameters are adapted.

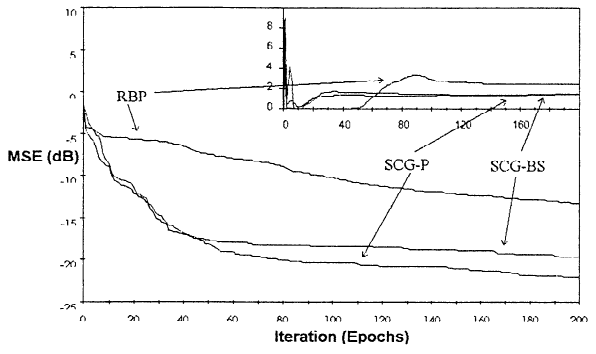

Fig. 3. PAM test results in batch mode.

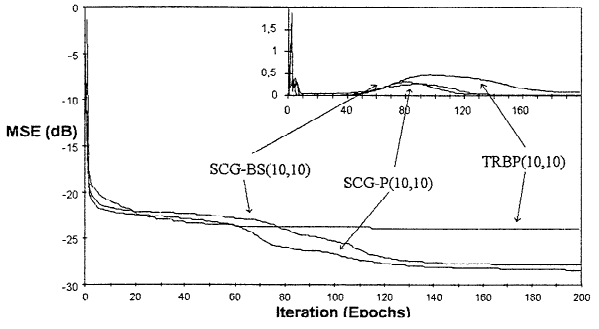

Fig. 4. PAM test results in on-line mode.

	Back-Tsoi (1000 samples)	P.A.M. (2048 samples)
RBP	1.336e+005	1.422e+005
SCG-*R*	3.438e+005	3.531e+005
SCG-*u*	2.655e+005	2.701e+005

Tab. 1. Computational complexity on the Back-Tsoi and P.A.M. (batch mode) tests - number of flop per iteration.

	Back-Tsoi (1000 samples)	P.A.M. (2048 samples)
TRBP(10,10)	1.852e+006	2.692e+006
SCG- *R* (10,10)	3.726e+006	5.361e+006
SCG- *u* (10,10)	3.534e+006	5.065e+006

Tab. 2. Computational complexity on the Back-Tsoi and P.A.M. (on-line mode) tests - number of flop per iteration.

Appendix: the SCG-*R* algorithm (on-line mode)

The following formulas are obtained applying the operator $R\{.\}$ to the formulas of forward and backward phases of the TRBP algorithm for an IIR-MLP neural network [8].

Forward phase:

$$R\left\{y_{nm}^{(l)}[t]\right\}=\sum_{p=0}^{L_{nm}^{(l)}-1}\left(w_{nm(p)}^{(l)}R\left\{x_m^{(l-1)}[t-p]\right\}+pw_{nm(p)}^{(l)}x_m^{(l-1)}[t-p]\right)+\sum_{p=1}^{I_{nm}^{(l)}}\left(v_{nm(p)}^{(l)}R\left\{y_{nm}^{(l)}[t-p]\right\}+pv_{nm(p)}^{(l)}y_{nm}^{(l)}[t-p]\right)$$

$$R\left\{s_n^{(l)}[t]\right\}=\sum_{m=0}^{N_{l-1}}R\left\{y_{nm}^{(l)}[t]\right\}\qquad R\left\{x_n^{(l)}[t]\right\}=R\left\{s_n^{(l)}[t]\right\}sgm'(s_n^{(l)}[t])$$

Backward phase:

$$R\left\{\Delta\tilde{w}_{nm(p)}^{(l)}\right\}=-\frac{1}{2}\sum_{t=\tau-h+1}^{\tau}R\left\{\Delta\tilde{w}_{nm(p)}^{(l)}[t+1]\right\}$$

where, separately for the MA and AR parts of each sinapsys,

$$R\left\{\Delta w_{nm(p)}^{(l)}[t+1]\right\}=R\left\{\delta_n^{(l)}[t]\right\}\frac{\partial s_n^{(l)}[t]}{\partial w_{nm(p)}^{(l)}}+\delta_n^{(l)}[t]R\left\{\frac{\partial s_n^{(l)}[t]}{\partial w_{nm(p)}^{(l)}}\right\}$$

$$R\left\{\Delta v_{nm(p)}^{(l)}[t+1]\right\}=R\left\{\delta_n^{(l)}[t]\right\}\frac{\partial s_n^{(l)}[t]}{\partial v_{nm(p)}^{(l)}}+\delta_n^{(l)}[t]R\left\{\frac{\partial s_n^{(l)}[t]}{\partial v_{nm(p)}^{(l)}}\right\}$$

where

$$R\left\{\frac{\partial s_n^{(l)}[t]}{\partial w_{nm(p)}^{(l)}}\right\}=R\left\{x_n^{(l-1)}[t-p]\right\}+\sum_{r=1}^{I_{nm}^{(l)}}\left(R\left\{v_{nm(p)}^{(l)}\right\}\frac{\partial s_n^{(l)}[t-r]}{\partial w_{nm(p)}^{(l)}}+v_{nm(p)}^{(l)}R\left\{\frac{\partial s_n^{(l)}[t-r]}{\partial w_{nm(p)}^{(l)}}\right\}\right)$$

$$R\left\{\frac{\partial s_n^{(l)}[t]}{\partial v_{nm(p)}^{(l)}}\right\}=R\left\{y_n^{(l-1)}[t-p]\right\}+\sum_{r=1}^{I_{nm}^{(l)}}\left(R\left\{v_{nm(p)}^{(l)}\right\}\frac{\partial s_n^{(l)}[t-r]}{\partial v_{nm(p)}^{(l)}}+v_{nm(p)}^{(l)}R\left\{\frac{\partial s_n^{(l)}[t-r]}{\partial v_{nm(p)}^{(l)}}\right\}\right)$$

$$R\left\{\delta_n^{(l)}[t]\right\}=R\left\{e_n^{(l)}[t]\right\}sgm'(s_n^{(l)}[t])+e_n^{(l)}[t]sgm''(s_n^{(l)}[t])R\left\{s_n^{(l)}[t]\right\}$$

and where the variables $\frac{\partial s_n^{(l)}[t]}{\partial w_{nm(p)}^{(l)}},\frac{\partial s_n^{(l)}[t]}{\partial v_{nm(p)}^{(l)}},\delta_n^{(l)}[t]$, are defined as in the TRBP algorithm. Moreover it holds

$$R\left\{e_n^{(l)}[t]\right\}=\begin{cases}\begin{cases}-R\left\{x_n^{(M)}[t]\right\} & t\in[\tau-N_c+1,\tau]\\ 0 & t\in[\tau-h+1,\tau-N_c]\end{cases} & l=M\\ \sum_{q=1}^{N_{l+1}}\sum_{k-N_c+1}^{\tau}\left(R\left\{\delta_q^{(l+1)}[k]\right\}\frac{\partial y_{qn}^{(l+1)}[k]}{\partial x_n^{(l)}[t]}+\delta_q^{(l+1)}[k]R\left\{\frac{\partial y_{qn}^{(l+1)}[k]}{\partial x_n^{(l)}[t]}\right\}\right) & l=M-1\\ \sum_{q=1}^{N_{l+1}}\sum_{k=t}^{\tau}\left(R\left\{\delta_q^{(l+1)}[k]\right\}\frac{\partial y_{qn}^{(l+1)}[k]}{\partial x_n^{(l)}[t]}+\delta_q^{(l+1)}[k]R\left\{\frac{\partial y_{qn}^{(l+1)}[k]}{\partial x_n^{(l)}[t]}\right\}\right) & l<M-1\end{cases}$$

$$R\left\{\frac{\partial y_{qn}^{(l+1)}[t+p]}{\partial x_n^{(l)}[t]}\right\}=\begin{cases}pw_{qn(p)}^{(l+1)} & 0\le p\le L_{qn}^{(l+1)}-1\\ 0 & otherwise\end{cases}+\sum_{r=1}^{\min(I_{qn}^{(l+1)},p)}\left(pv_{qn(r)}^{(l+1)}\frac{\partial y_{qn}^{(l+1)}[t+p-r]}{\partial x_n^{(l)}[t]}+v_{qn(r)}^{(l+1)}R\left\{\frac{\partial y_{qn}^{(l+1)}[t+p-r]}{\partial x_n^{(l)}[t]}\right\}\right)$$