ESANN'1998 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 22-23-24 April 1998, D-Facto public., ISBN 2-9600049-8-1, pp. 279-284

A multistage on-line learning rule for multilayer neural network

P. Thomas, G. Bloch and C. Humbert

Centre de Recherche en Automatique de Nancy, CNRS UPRES A 7039
ESSTIN, Rue Jean Lamour, 54500 VANDOEUVRE, FRANCE
Tel: +33 (0)3 83 50 33 33; Fax: +33 (0)3 83 54 21 73
E-mail: thomas@cran.esstin.u-nancy.fr

Abstract. One hidden layer feedforward neural networks are considered here. Parameters estimation of such networks requires a supervised learning which can be carried out in batch mode or on-line. From the first batch learning paradigm, i.e. backpropagation or steepest descent method, many improvements have been proposed. Batch learning can nevertheless present some limitations. On-line algorithms, as stochastic processes, can prevent the learning from getting trapped in a local minimum. On the other hand, for applications requiring real-time adaptation, these algorithms are indispensable. In this paper, a new on-line learning algorithm, exploiting the separability of the network into linear and nonlinear modules, is proposed and compared with the standard RPE algorithm, particularly for learning new operating zones.

1. Introduction

The popularity of neural networks in many engineering fields (system identification and control, robotics, signal processing, etc.) is mainly due to their abilities to approximate complex nonlinear mappings directly from data. The neural networks considered here require a supervised learning. This learning can be performed in batch or on-line version. From the first batch learning algorithm presented by Rumelhart and McClelland in 1986 [7], many effective improvements have been proposed [3; 12; 9]. The batch learning can nevertheless present some limitations (slowness for error surface with flat ‘plateaus’ or narrow ‘valleys’, convergence to local minima). On-line algorithms, as stochastic processes, can prevent the learning from getting trapped in a local minimum. On the other hand, for applications requiring real-time adaptation, these algorithms are indispensable, for example when the underlying system to be learned changes during time, or when a new operating point is reached. Basic on-line algorithms suffer also of the slowness of the learning. Other methods exploiting the separability of the networks into linear and nonlinear modules have been thus studied [5; 8; 11].

The one hidden layer perceptron with linear activation function at the output is considered here. The form of this neural network is given, for single output, by:

$$\hat{y} = \sum_{i=1}^{n_1} w_i^2 x_i^1 + b^2 = \sum_{i=1}^{n_1} w_i^2 g(z_i^1) + b^2 = \sum_{i=1}^{n_1} w_i^2 g(\sum_{h=1}^{n_0} w_{ih}^1 x_h^0 + b_i^1) + b^2 \qquad (1)$$

where x_h^0, $h = 1, ..., n_0$, are the inputs of the network, w_{ih}^1 and b_i^1, $i = 1, ..., n_1$, $h = 1, \cdots, n_0$, are the weights and biases of the hidden layer, x_i^1, $i = 1, ..., n_1$ are the outputs of the hidden neurons, w_i^2 and b^2 are the weights and bias of the output neuron. All the weights and biases of the network are grouped in the parameter vector θ. The activation function g used in the hidden layer is the hyperbolic tangent. The proposed learning algorithm is described in the next section and compared in a third part on a simulation example with the RPE algorithm (Recursive Prediction Error) proposed by Billings and Jamaluddin [1].

2. The multistage learning algorithm

The proposed algorithm does not use the backpropagation principle but exploits the structure of the neural network in order to make each neuron work individually. The algorithm is based on the estimation method for the linear systems called ALS (Alternated Least Squares) [4; 6]. ALS method permits the identification of MIMO (Multi-Input, Multi-Output) linear systems which can be divided into MISO (Multi-Input, Single-Output) sub-systems.

In the algorithm presented here, each neuron is considered as a sub-system with inputs, output and parameters. The neural network can be so considered as a MISO system that can be broken up. The outputs of the hidden neurons which are not measured can be estimated by using the inputs of the network as well as by using the output of the network. During the learning, at each time t, the algorithm updates alternatively the hidden weights and biases, then the output weights and biases.

The initialization of the learning algorithm is performed by initializing the biases and weights of the network, following [10], and by calculating, for each hidden neuron i, the weighted sum $\vec{z}_i^1(t)$ at time t = 1 (the arrow shows the propagation direction of the information). This weighted sum is calculated by using the initial weights connecting the input layer to the hidden layer by:

$$\vec{z}_i^1(t) = \sum_{h=1}^{n_0} w_{ih}^1 x_h^0(t) + b_i^1 \qquad i = 1, \cdots, n_1 \tag{2}$$

<u>*Step 1*</u>: The weighted sums $\overleftarrow{z}_i^1(t)$ are calculated by using the information given by the desired output of the network y(t), at time t, and by using the weights and bias connecting the hidden layer to the output layer:

$$\overleftarrow{z}_i^1(t) = g^{-1}\Big(y(t) - b^2 - \sum_{\substack{j=1 \\ j \neq i}}^{n_1} w_j^2 . g(\vec{z}_j^1(t))\Big) \quad i = 1, \cdots, n_1 \tag{3}$$

However, when the weighted sum of the hidden neuron i $\overleftarrow{z}_i^1(t)$ is calculated by using the information given by the network output, the weighted sums $\overleftarrow{z}_j^1(t)$ of the other hidden neurons j are not known. If $\hat{\theta}(t)$ is supposed to be close to the real parameters of the system at time t, $\overleftarrow{z}_j^1(t) = \vec{z}_j^1(t)$ can be written. The use of this relation in equation (3) permits the estimation of $\overleftarrow{z}_i^1(t)$. However, this approximation can lead to some problems due to the inversion of the activation function. So the estimation of $\overleftarrow{z}_i^1(t)$, $i = 1, \ldots, n_1$ is given by:

$$\overleftarrow{x}_i^1(t) = y(t) - b^2 - \sum_{\substack{j=1 \\ j \neq i}}^{n_1} w_j^2 . g(\vec{z}_j^1(t)) \qquad i = 1, \ldots, n_1 \tag{4a}$$

and:

$$\overleftarrow{z}_i^1(t) = \begin{cases} \alpha & \text{if } \overleftarrow{x}_i^1(t) > 1 \\ g^{-1}(\overleftarrow{x}_i^1(t)) & \text{if } \left|\overleftarrow{x}_i^1(t)\right| \leq 1 \\ -\alpha & \text{if } \overleftarrow{x}_i^1(t) < -1 \end{cases} \tag{4b}$$

where the positive constant α is taken here equal to 1. Equation (4b) avoids the activation function inversion problems and prevents the saturation of the hidden neurons, particularly if outliers are introduced in the learning. The parameter α can be tuned to constrain the weighted sum $\overleftarrow{z}_i^1(t)$ in a fixed range.

Step 2: For each neuron i of the hidden layer, the weights w_{ih}^1 and bias b_i^1 are estimated by using a recursive algorithm:

$$P(t) = P(t-1) - \frac{P(t-1)\, \psi(t, \hat{\theta}(t-1))\, \psi^T(t, \hat{\theta}(t-1))\, P(t-1)}{1 + \psi^T(t, \hat{\theta}(t-1))\, P(t-1)\, \psi(t, \hat{\theta}(t-1))} \tag{5a}$$

and:

$$\hat{\theta}(t) = \hat{\theta}(t-1) + P(t)\, \psi(t, \hat{\theta}(t-1))\, \varepsilon(t, \hat{\theta}(t-1)) \tag{5b}$$

Here $\hat{\theta}(t)$ comprises the estimated weights and bias of the neuron concerned at time t:

$$\hat{\theta}(t) = [\hat{w}_{i1}^1(t) \cdots \hat{w}_{in_0}^1(t) \;\; \hat{b}_i^1(t)]^T$$

ε is the residual obtained for the output of the neuron:

$$\varepsilon(t, \hat{\theta}(t-1)) = \overleftarrow{z}_i^1(t) - \vec{z}_i^1(t, \hat{\theta}(t-1)) = \overleftarrow{z}_i^1(t) - \sum_{h=1}^{n_0} \hat{w}_{ih}^1(t-1) x_h^0(t) - \hat{b}_i^1(t-1)$$

and Ψ is the gradient given by:

$$\psi(t, \hat{\theta}(t-1)) = \left.\frac{\partial \vec{z}_i^1(t, \theta)}{\partial \theta}\right|_{\theta = \hat{\theta}(t-1)} = [x_1^0(t) \cdots x_{n_0}^0(t) \;\; 1]^T$$

Step 3: The new weighted sums $\bar{z}_i^1(t)$ are calculated by equation (2) by using the new weights and biases obtained at the previous step and the information given by the input of the network.

Step 4: The weights w_i^2 and the bias b^2 of the output neuron are estimated by using equations (5) with here $\hat{\theta}(t)$ comprising the estimated weights and bias of the neuron concerned at time t:

$$\hat{\theta}(t) = [\hat{w}_1^2(t) \cdots \hat{w}_{n_1}^2(t)\ \ \hat{b}^2(t)]^T$$

ε is the residual obtained for the output of the neuron:

$$\varepsilon(t,\hat{\theta}(t-1)) = y(t) - \hat{y}(t,\hat{\theta}(t-1)) = y(t) - \sum_{i=1}^{n_1} \hat{w}_i^2(t-1) g(\bar{z}_i^1(t)) - \hat{b}^2(t-1)$$

and Ψ is the gradient given by:

$$\psi(t,\hat{\theta}(t-1)) = \left.\frac{\partial \hat{y}(t,\theta)}{\partial \theta}\right|_{\theta=\hat{\theta}(t-1)} = [g(\bar{z}_1^1(t)) \cdots g(\bar{z}_{n_1}^1(t))\ \ 1]^T$$

Step 5: If terminated criterion is not satisfied, t = t + 1 and go to step 1.
This algorithm is called JDA (Job Distribution Algorithm) in the next section.

3. Simulation results

The two recursive algorithms JDA and RPE [1] are compared. The simulation system is a discrete time NARX system proposed by Chen *et al.* [2]:

$$\begin{aligned} y(t) = {} & (0{,}8 - 0{,}5\, e^{-y^2(t-1)})\, y(t-1) - (0{,}3 + 0{,}9\, e^{-y^2(t-1)})\, y(t-2) \\ & + u(t-1) + 0{,}2\, u(t-2) + 0{,}1\, u(t-1)\, u(t-2) + e(t) \end{aligned} \tag{6}$$

where the input of the system at time t, u(t), is a sequence of steps of random duration with magnitude randomly chosen between -1 and 2, y(t) is the output of the system, e(t) is a Gaussian white noise of mean 0 and variance 0.16. The input vector of the network comprises the two delayed inputs u(t-1) and u(t-2) and the two delayed outputs y(t-1) and y(t-2). In the simulations, the network used has 5 hidden neurons and the two algorithms use the same initial parameters. The test criterion is the MSE (Mean Square Error):

$$MSE = \frac{1}{n} \sum_{t=1}^{n} (y(t) - \hat{y}(t))^2 \tag{7}$$

In a first time, the performances of the two algorithms to learn on-line the parameters of the network are compared. Figure 1 shows the evolution of the MSE criterion, for one example, during learning for the two algorithms. Other simulations give similar results. This figure shows that the two algorithms have similar performances at the beginning of the learning, but that the JDA algorithm is faster in the sequel.

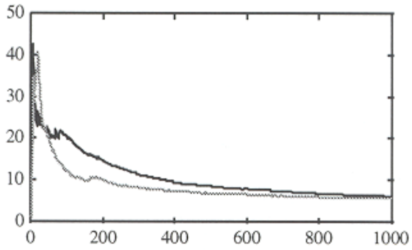

Fig. 1. Evolution of the MSE criterion (JDA : gray - RPE : black).

These two algorithms are recursive. They should thus adapt to an evolution of the system or to a change of operating zone. The first case has been studied in [11]. In this case, the JDA algorithm presents better results, i.e. faster adaptation, than the RPE algorithm. An example of change of operating zone is presented here. In this case, the initialization of the network is performed by a batch learning with inputs comprised between -1 and 2. For the on-line learning, the input is a sequence of steps of random duration with magnitude randomly chosen between -1 and 2 for $t < 300$, and between 3 and 6 for t > 300. Figure 2 presents the input of the system for the on-line learning.

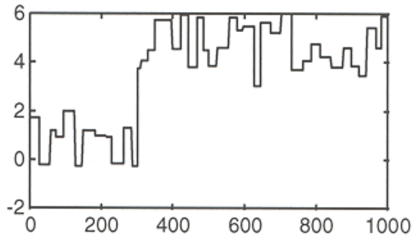

Fig. 2. Input of the system for the on-line learning.

Figures 3a and b shown the residuals obtained during time for the JDA and RPE algorithm respectively.

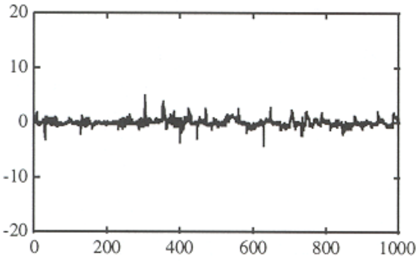

Fig. 3a. Residuals obtained with JDA algorithm.

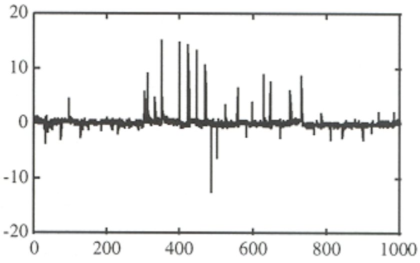

Fig. 3b. Residuals obtained with RPE algorithm.

The comparison of these two residuals shown that the JDA algorithm learns more quickly the new operating zone than the RPE algorithm. The residuals obtained with RPE algorithm are larger than those obtained with JDA algorithm until t = 700, particularly for each change of operating point.

4. Conclusion

In this paper, a new on-line learning algorithm (JDA) has been proposed and compared with the RPE algorithm in a simulation study. The first simulation example shows that the new algorithm learns more quickly the parameters of the network than the RPE algorithm. In a second time, the generalization of the network to a new operating zone is studied and the JDA algorithm gives also better results. This new algorithm seems interesting to use, particularly, for example in adaptive control with neural networks.

5. References

1. BILLINGS S.A., H.B. JAMALUDDIN (1991) ‘A comparison of the backpropagation and recursive prediction error algorithms for training neural networks’, *Mechanical Systems and Signal Processing*, Vol.5, 3, 233-255.
2. CHEN S., S.A. BILLINGS, P.M. GRANT (1990) ‘Non-linear system identification using neural networks’, *Int. J. Control* , Vol.51, 6, 1191-1214.
3. CICHOCKI A., R. UNBEHAUEN (1993) *Neural networks for optimization and signal processing,* John Wiley & Sons, New-York, USA.
4. MIELCAREK D., C. LOFFLER, J. RAGOT, G. BLOCH (1991) ‘A comparative study of some recursive parameters estimation algorithms for multivariable systems’, *IMACS Annals on Computing and applied mathematics*, Vol. Mathematical and Intelligent Models in System Simulation, 59-64.
5. PARISI R., E. DICLAUDIO, G. ORLANDI, B.D. RAO (1996) ‘A generalized learning paradigm exploiting the structure of feedforward neural networks’, *IEEE Trans. on Neural Networks*, Vol.7, 6, 1450-1460.
6. RAGOT J., D. MIELCAREK D. (1992) ‘Recursive identification of multivariable interconnected systems’, *Int. J. Syst. Sci.*, Vol.23, 6, 987-1000.
7. RUMELHART D.E., J.L. MCCLELLAND (1986) *Parallel distributed processing,* The MIT Press, Cambridge, England.
8. SCALERO R.S., N. TEPEDELENLIOGLU (1992) ‘A fast algorithm for training feedforward neural networks’, *IEEE Trans. on Signal Processing*, Vol.40, 1, 202-210.
9. SHEPERD A.J. (1997) *Second-order methods for neural networks,* Springer, London, England.
10. THOMAS P., G. BLOCH (1997) ‘Initialization of one hidden layer feedforward neural networks for non-linear system identification’, *Proceedings of the 15th IMACS World Congress WC'97*, Berlin, Germany, 25-29 August, Vol.4, 295-300.
11. THOMAS P. (1997) *Contribution à l’identification de systèmes non linéaires par réseaux de neurones*, Thèse de doctorat de l’université Nancy I, CRAN, spécialité automatique.
12. VAN DER SMAGT P.P. (1994) ‘Minimisation methods for training feedforward neural networks’, *Neural Networks*, Vol.7, 1, 1-11.