# A Linear Learning Method for Multilayer Perceptrons Using Least-Squares

Bertha Guijarro-Berdiñas, Oscar Fontenla-Romero,
Beatriz Pérez-Sánchez, and Paula Fraguela

Department of Computer Science
Facultad de Informática, Universidad de A Coruña,
Campus de Elviña s/n, 15071 A Coruña, Spain

**Abstract.** Training multilayer neural networks is typically carried out using gradient descent techniques. Ever since the brilliant *backpropagation* (BP), the first gradient-based algorithm proposed by Rumelhart et al., novel training algorithms have appeared to become better several facets of the learning process for feed-forward neural networks. *Learning speed* is one of these. In this paper, a learning algorithm that applies linear-least-squares is presented. We offer the theoretical basis for the method and its performance is illustrated by its application to several examples in which it is compared with other learning algorithms and well known data sets. Results show that the new algorithm upgrades the learning speed of several backpropagation algorithms, while preserving good optimization accuracy. Due to its performance and low computational cost it is an interesting alternative, even for second order methods, particularly when dealing large networks and training sets.

## 1 Motivation

Among the many variants of neural network architectures that exist, feed-forward neural networks (and specially, those based on the MultiLayer Perceptron, MLP), are one of the most popular models with successful applications in many fields. The power of these networks comes from having several layers of adaptive weights and nonlinear activation functions (e.g. the sigmoid or hyperbolic tangent). Generally, the sum-of-squares error function is employed for estimating the performance of the network, that compares the desired signal with the network's output. There is not a closed-form solution to find the weight values that minimizes the sum-of-squares error function [9]. Hence the common approach is to use the derivatives of the error function with respect to the weight parameters in gradient-based optimization algorithms for finding the minimum of the error function.

Ever since the first gradient-based algorithm, the brilliant backpropagation (BP) proposed by Rumelhart et al. [1], researchers have focused their efforts on improving the convergence properties of BP, the main concern being the slow convergence speed due to its gradient-descent nature. Some of the newly proposed algorithms that try to improve this aspect are modifications of the

H. Yin et al. (Eds.): IDEAL 2007, LNCS 4881, pp. 365–374, 2007.

original BP such as adding a momentum term [1,2], an adaptive step size [3] or using stochastic learning [4].

Others are second order methods that use the second derivatives of the error function. Some of the most relevant examples of these types of methods are the quasi-Newton approaches including the Levenberg-Marquardt [5] and the conjugate gradient algorithms [6]. Second order methods are among the fastest learning algorithms, however due to their computational cost they are not feasible for large neural networks trained in batch mode.

At last, it is also possible to find methods based on linear least-squares [7,8,9,10,11]. These methods are mostly based on measuring the error of an output neuron *before* the nonlinear transfer function instead of *after* it, as is the usual approach. Usually, they are recommended as training methods for one-layer networks or as initialization methods for multilayer networks.

Specifically, in [10] a method is described to solve a one-layer non-linear neural network using linear least squares. Also using this solution in [12] an algorithm is proposed that linearly calculates the weights of every layer for a multilayer perceptron. However, in this algorithm layers are solved independently and therefore it can only be used as an initialization method. In this paper, we modified this approach by solving each layer in order, from the output layer to the input layer, in such way that the weights in each layer are solved by taking into account the new weights calculated for the succeeding layers. As a consequence, we developed a learning algorithm that improves the learning speed of the basic backpropagation in several orders of magnitude, while maintaining its optimization accuracy.

The organization of this paper is as follows. First, in section 2, we present the method for learning the weights of the network. In Section 3 we investigate the performance of the proposed method on benchmark classification problems and it is compared with several other well-known training methods. In Section 4 these results are discussed. Finally, section 5 bids some suggestions and conclusions.

## 2 The Proposed Algorithm

In this research we will consider, without loss of generality, a two-layer MLP like the one shown in Fig. 1. The variable names are described below.

Constants $I$, $K$, $J$ and $S$ symbolize respectively, the number of inputs, hidden units, outputs and training samples. Each layer of the network consists of a linear matrix $\mathbf{W}^{(n)}$ of weights $w_{ji}^{(n)}$ connecting neuron $j$ in layer $n$ with neuron $i$ in layer $n-1$, thus the superscript $n = 1, \ldots, N$ is used to refer to each layer. These weight matrices are followed by nonlinear mappings $f_j^{(n)}$, regularly selected to be sigmoid-type functions.

For each layer $n$, the input vectors of the MLP are represented as $\mathbf{x}^{(n)}$. The bias of each layer has been included into weight matrix by adding constant inputs $x_{0s}^{(n)} = 1, \forall n$.

In addition, for all $j = 1, ..., J; s = 1, ..., S$, we will denote by $y_{js}$ the real output obtained by the network, $z_{js}$ the inputs to the non-linearities of the output layer and by $d_{js}$ the desired response provided in the training set. Finally,

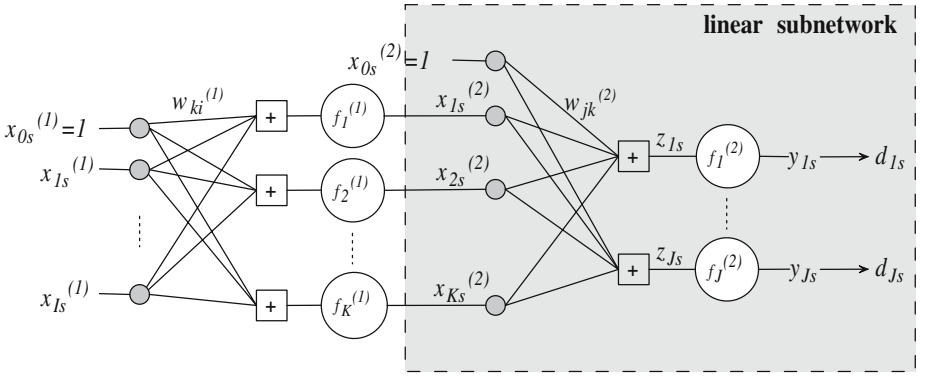


**Fig. 1.** Architecture of a two-Layer MLP

in the following we will consider as the training optimization criterion, the MSE between the real $\mathbf{y}$ and the desired output $\mathbf{d}$.

### 2.1 One-Layer Linear Learning: Determining the Weights

Take, for instance, the one-layer neural network corresponding to the shadowy part of Fig. 1. In [10,11,12], the authors considered the approximate least squares optimization of an one-layer nonlinear network assuming the MSE before the nonlinearity as the criterion to be optimized by means of the following theorem (see the proof in [12]),

**Theorem 1.** *Minimization of the MSE between* $\mathbf{d}$ *and* $\mathbf{y}$ *at the output of the nonlinearity* $f$ *is equivalent (up to first order) to minimizing a MSE between* $\mathbf{z}$ *and* $\bar{\mathbf{d}} = f^{-1}(\mathbf{d})$*, where the inverse function is evaluated at each entry separately. Mathematically, this is given by*

$$\begin{aligned} &\min_{\mathbf{W}} \sum_{s=1}^{S} \sum_{j=1}^{J} \left( f_j \left( \sum_{k=0}^{K} w_{jk} x_{ks} \right) - d_{js} \right)^2 \approx \\ &\min_{\mathbf{W}} \sum_{s=1}^{S} \sum_{j=1}^{J} \left( f_j'(\bar{d}_{js}) \left( \sum_{k=0}^{K} w_{jk} x_{ks} - f_j^{(-1)}(d_{js}) \right) \right)^2 \end{aligned} \tag{1}$$

According to this new error criterion the weights can be optimized by solving a system of $J \times S$ linear equations defined by:

$$\begin{aligned} &\frac{\partial MSE}{\partial w_{jp}} = 2 \sum_{s=1}^{S} \left( f_j'(\bar{d}_{js}) \left( \sum_{k=0}^{K} w_{jk} x_{ks} - f_j^{(-1)}(d_{js}) \right) \right) x_{ps} f_j'(\bar{d}_{js}) = 0; \\ &p = 0, 1, \ldots, K; \quad \forall j \ . \end{aligned} \tag{2}$$

The use of this system presents two main advantages: 1) the global optimum of the training set is obtained, and 2) there is a considerable savings in training time with respect to other gradient-based optimization techniques.

### 2.2 One-Layer Linear Learning: Determining the Inputs

In [11,12], the authors considered that the weights $\mathbf{W}$ can be fixed, and therefore the input vector $\mathbf{x}$ becomes the free optimization variable. This is a very useful result if $\mathbf{x}$ corresponds to the output (after non linearity) of a hidden layer (see, for example, $\mathbf{x}^{(2)}$ in Fig. 1). Consequently, this result allows the backpropagation of the desired signal for $\mathbf{z}$ to a desired signal for $\mathbf{x}$ through the linear weight layer. The result is summarized in the following theorem, that is proved in [12].

**Theorem 2.** *Let* $\mathbf{W}$ *be the fixed weight matrix and* $\mathbf{x}$ *the actual input. Then the optimal input* $\mathbf{x}_{opt}$ *that minimizes the MSE between* $\bar{\mathbf{d}} = f^{-1}(\mathbf{d})$ *and* $\mathbf{z}$ *is the input* $\mathbf{x}_{opt}$ *that minimizes a modified MSE criterion between* $\mathbf{x}$ *and the best solution to* $\mathbf{W}\mathbf{x}_{opt} = \bar{\mathbf{d}}$ *in the least squares sense. That is,*

$$\min_{\mathbf{x}} E[(\bar{\mathbf{d}} - \mathbf{z})^T(\bar{\mathbf{d}} - \mathbf{z})] \approx \min_{\mathbf{x}} E[(\mathbf{x}_{opt} - \mathbf{x})^T \mathbf{W}^{\mathbf{T}} \mathbf{W}(\mathbf{x}_{opt} - \mathbf{x})] \tag{3}$$

*In this case, the bias is included in the matrix* $\mathbf{W}$ *and its corresponding input* $(x_0)$ *is fixed to 1. In this way,* $\mathbf{x}_{opt} = \{x_{1s}, x_{2s}, ..., x_{Ks}\}, \forall s.$

### 2.3 Combining Theorem 1 and Theorem 2 for Linear Learning of a MLP

Theorem 1 can be used as a basis to provide the desired signal before the non-linearity for every layer of the network and, therefore, linearly find the optimal weights for each layer. Moreover, and in combination with Theorem 2, it is employed to provide a global linear solution for the networks.

The proposed algorithm for linear learning of a multilayer feedforward neural network is as follows:

**Step 1:** Set the initial weights $\mathbf{W}^{(n)} \forall n$.
**Step 2:** Using the current weights, propagate the signal forward to calculate the outputs of each layer.
**Step 3:** Evaluate the value of the $MSE$ between $\mathbf{y}$ and $\mathbf{d}$ and update $\mathbf{W}^{(2)}$ (i.e., the output layer) using the linear system of equations presented in equation 2.
**Step 4:** Calculate the optimum desired inputs of the output layer (i.e. the desired outputs for the hidden layer) by using the linear system of equations resulting from the right side of equation 3.
**Step 5:** Update the weights of the hidden layer $\mathbf{W}^{(1)}$ according to the optimal desired outputs calculated in Step 4 and using again the linear system of equations in 2.
**Step 6:** Check convergence criteria. If they are not reached, continue from Step 2.

The main difference between this algorithm and the proposed one in [12] is the order as the weights of each layer are optimized. In the previous approach first the desired target of all layers is estimated and second the weights are calculated. However, in this case the estimation of the desired output and the calculation of the weights are simultaneously obtained, layer by layer, in a backward way.

## 3 Experimental Results

In this section the proposed method (linear algorithm) is illustrated by its application to three classification problems of different size, and its performance is compared with four popular learning methods. Two of these methods are of complexity $O(n)$: the gradient descent (GD) and the gradient descent with adaptive momentum and step sizes (GDX), in which the proposed method is based. The other two methods are the scaled conjugated gradient (SCG) (complexity of $O(n^2)$), and the Levenberg-Marquardt (LM) (complexity of $O(n^3)$).

For each experiment all the learning methods shared the following conditions:

- They were carried out in MATLAB® on a 3.20 GHz Pentium 4 processor with 2.5 GB of RAM memory.
- The logistic function was used as the nonlinear functions for neurons.
- The input data set was normalized in the interval [0,1].
- The training process was run for a maximum of 200 epochs.
- For the GDX and the proposed algorithms initial step size was set to 0.005. Moreover the factor used to decrease/increase the learning rate was fixed to 0.01. These values were tuned in order to obtain good results.
- Regarding the topology, a cross-validation method was used to obtain the optimal number of hidden neurons. In this paper only results of the best topology are shown. Specifically for the Breast Cancer and the Wine datasets, 9 and 15 hidden neurons were employed, respectively.
- Each experiment was repeated five times, using a different set of initial weights for each one. This initial set was the same for all the algorithms, and was obtained by the Nguyen-Widrow [13] initialization method.

### 3.1 Breast Cancer Wisconsin

This two-class problem determines if a patient suffers breast cancer based on several characteristics of the nuclei's cells. The database contains a sample set of 699 instances. The network topology has 9 hidden neurons, 9 inputs and 2 outputs.

Fig. 2 shows the mean MSE training error curves, obtained by each of the tested methods, for the 5 simulations. As can be observed, already in the fourth epoch the proposed method obtains its minimum.

Also, in table 1 some performance measures are shown that allow for the comparison of the algorithms. The first column ($M1$) measures corresponds to the minimum $MSE$ obtained by each method in the training process. The second column ($M2$) is the obtained $MSE$ by each method in the epoch in which the

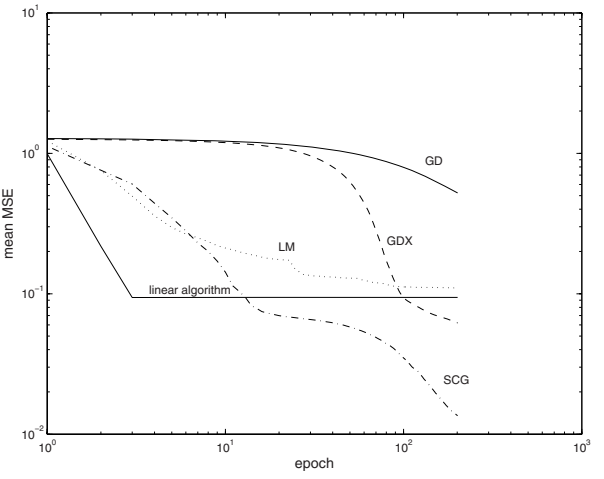


**Fig. 2.** Mean MSE training error curves for the Breast Cancer Wisconsin dataset

**Table 1.** Performance measures for the Breast Cancer Wisconsin dataset

| Algorithm | M1 | M2 |
|---|---|---|
| GD | $5.22 \times 10^{-1} \pm 2.89 \times 10^{-1}$ | $1.26 \pm 3.63 \times 10^{-1}$ |
| GDX | $6.17 \times 10^{-2} \pm 4.76 \times 10^{-3}$ | $1.24 \pm 3.98 \times 10^{-1}$ |
| SCG | $1.35 \times 10^{-2} \pm 4.54 \times 10^{-3}$ | $4.42 \times 10^{-1} \pm 3.26 \times 10^{-1}$ |
| LM | $1.10 \times 10^{-1} \pm 2.88 \times 10^{-1}$ | $3.59 \times 10^{-1} \pm 1.3.99 \times 10^{-1}$ |
| Linear | $9.40 \times 10^{-2} \pm 5.57 \times 10^{-3}$ | $9.41 \times 10^{-2} \pm 5.49 \times 10^{-3}$ |

**Table 2.** Train and Test Accuracy for the Breast Cancer Wisconsin dataset

| Algorithm | Acc. Train (%) | Acc. Test (%) |
|---|---|---|
| GD | $81.6 \pm 2.79$ | $80.3 \pm 2.08$ |
| GDX | $97.7 \pm 1.09$ | $96.7 \pm 2.36 \times 10^{-1}$ |
| SCG | $99.8 \pm 3.99 \times 10^{-2}$ | $94.6 \pm 6.90 \times 10^{-1}$ |
| LM | $94.5 \pm 6.46$ | $87.8 \pm 5.21$ |
| Linear | $96.0 \pm 3.06 \times 10^{-2}$ | $96.0 \pm 2.59 \times 10^{-1}$ |

proposed method obtained its minimum $MSE$. As these measures are calculated over the 5 simulations they all are provided in terms of mean and corresponding standard deviation. Moreover, in table 2 train and test classification accuracy, measured at the end of the simulation, are presented.

Finally, table 3 shows both the mean time (in seconds) per epoch $Tepoch_{mean}$ and the mean time of the whole training process $Ttotal_{mean}$ of every algorithm.

**Table 3.** CPU time comparison (in seconds) for the Breast Cancer Wisconsin dataset

| Algorithm | $Tepoch_{mean}$ | $Tepoch_{std}$ | $Ttotal_{mean}$ | $Ttotal_{std}$ |
|---|---|---|---|---|
| GD | $1.49 \times 10^{-2}$ | $2.96 \times 10^{-2}$ | 3.081 | 0.627 |
| GDX | $1.40 \times 10^{-2}$ | $2.85 \times 10^{-2}$ | 2.867 | 0.168 |
| SCG | $2.63 \times 10^{-2}$ | $2.99 \times 10^{-2}$ | 5.249 | 0.628 |
| LM | $8.05 \times 10^{-2}$ | $4.52 \times 10^{-2}$ | 15.312 | 4.988 |
| Linear | $4.04 \times 10^{-2}$ | $2.78 \times 10^{-2}$ | 0.349 | 0.202 |

### 3.2 Wine

These data are the results of a chemical analysis of wines growing in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines. The database includes 178 instances. The network topology has 15 hidden neurons, 13 inputs and 3 outputs.

Fig. 3 shows the mean MSE training error curves obtained by each of the tested methods for the 5 simulations. As can be observed, already in the third epoch the proposed method obtains an error very close to its minimum.

Also, in table 4 again the performance measures $M1$ and $M2$ are shown that allows for the comparison of the algorithms. In this case, 200 iterations are not enough for the other methods to obtain the $MSE$ that the linear algorithm exhibits at the 3rd epoch. Moreover, in table 5 train and test accuracy, measured at the end of the simulation, are also presented.

Finally, table 6 shows the mean time per epoch $Tepoch_{mean}$ and the mean time of the whole training process $Ttotal_{mean}$ of every algorithm.

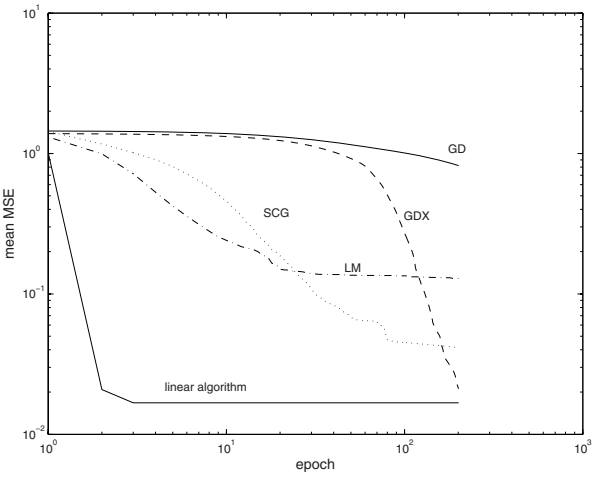


**Fig. 3.** Mean MSE training error curves for the Wine dataset

**Table 4.** Performance measures for the Wine dataset

| Algorithm | M1 | M2 |
|---|---|---|
| GD | $8.20 \times 10^{-1} \pm 2.76 \times 10^{-1}$ | $1.43 \pm 3.33 \times 10^{-1}$ |
| GDX | $2.10 \times 10^{-2} \pm 3.77 \times 10^{-2}$ | $1.37 \pm 3.36 \times 10^{-1}$ |
| SCG | $4.16 \times 10^{-2} \pm 1.53 \times 10^{-1}$ | $1.01 \pm 2.82 \times 10^{-1}$ |
| LM | $1.29 \times 10^{-1} \pm 2.58 \times 10^{-1}$ | $7.18 \times 10^{-1} \pm 4.07 \times 10^{-1}$ |
| Linear | $1.66 \times 10^{-2} \pm 2.18 \times 10^{-3}$ | $1.68 \times 10^{-2} \pm 2.23 \times 10^{-3}$ |

**Table 5.** Train and Test Accuracy for the Wine dataset

| Algorithm | Acc. Train (%) | Acc. Test (%) |
|---|---|---|
| GD | $49.4 \pm 5.32$ | $48.7 \pm 9.37$ |
| GDX | $99.8 \pm 1.78 \times 10^{-1}$ | $97.5 \pm 6.46 \times 10^{-1}$ |
| SCG | $96.7 \pm 4.71$ | $94.0 \pm 3.60$ |
| LM | $89.4 \pm 8.10$ | $86.5 \pm 8.63$ |
| Linear | $99.9 \pm 3.41 \times 10^{-2}$ | $99.0 \pm 2.48 \times 10^{-1}$ |

**Table 6.** CPU time comparison (in seconds) for the Wine dataset

| Algorithm | $Tepoch_{mean}$ | $Tepoch_{std}$ | $Ttotal_{mean}$ | $Ttotal_{std}$ |
|---|---|---|---|---|
| GD | $0.59 \times 10^{-2}$ | $0.77 \times 10^{-2}$ | 1.246 | 0.145 |
| GDX | $0.58 \times 10^{-2}$ | $0.76 \times 10^{-2}$ | 1.182 | 0.055 |
| SCG | $1.25 \times 10^{-2}$ | $0.73 \times 10^{-2}$ | 2.041 | 0.972 |
| LM | $1.48 \times 10^{-1}$ | $5.18 \times 10^{-2}$ | 23.309 | 12.444 |
| Linear | $2.1 \times 10^{-2}$ | $0.82 \times 10^{-2}$ | 0.139 | 0.143 |

## 4 Discussion

From Figs. 2 and 3 and tables 1 and 4 we can see that the proposed method obtains its minimum error at a very early epoch. Regarding its accuracy, it improves the one exhibited by the classical backpropagation and also the Levenberg Marquardt algorithm. Moreover for wine dataset 200 iterations are not enough for the other methods to obtain the minimum $MSE$ that the linear algorithm exhibits.

Also from measure $M2$ in tables 1 and 4 it can be concluded that when our method reaches an error value near its minimum the other algorithms are in a minimum several orders of magnitude higher.

Regarding the accuracy measures shown in tables 2 and 5 it can be deduced that the proposed algorithm again improves the gradient descent and the Levenberg Marquardt algorithms while maintaining an accuracy similar to the GDX and SCG algorithms. It is important to notice that from the tested algorithms it

is the one that shows the more similar accuracy when the results over the train and test sets are compared, thus confirming its generalization ability. Finally, as it can be observed, our method avoids the overfitting since it maintains the same behavior for the training and test sets.

This conclusion, together with the short time needed for training by the proposed method, as shown in tables 3 and 6, definitely makes it a fast and suitable learning algorithm.

## 5 Conclusions and Future Work

The analyzed results allow us to confirm that the proposed method offers an interesting combination of speed, reliability and simplicity. The method obtains good approximations that even overcome those provided by classic or second order algorithms. These features makes the proposed algorithm suitable for those situations when the speed of the method is important in reaching a good solution, although this could not be always the best one.

## Acknowledgements

We would like to acknowledge support for this project from the Xunta de Galicia (project PGIDT05TIC10502PR) and the Ministerio de Educación y Ciencia, Spain (project TIN2006-02402), partially supported by the European Union ERDF.

## References

1. Rumelhart, D.E., Hinton, G.E., William, R.J.: Learning representations of back-propagation errors. Nature 323, 533–536 (1986)
2. Vogl, T.P., Mangis, J.K., Rigler, A.K., Zink, W.T., Alkon, D.L.: Accelerating the convergence of back-propagation method. Biological Cybernetics 59, 257–263 (1988)
3. Jacobs, R.A.: Increased rates of convergence through learning rate adaptation. Neural Networks 1(4), 295–308 (1988)
4. LeCun, Y., Bottou, L., Orr, G.B., Müller, K.-R.: Efficient BackProp. In: Orr, G.B., Müller, K.-R. (eds.) Neural Networks: Tricks of the Trade. LNCS, vol. 1524, Springer, Heidelberg (1998)
5. Hagan, M.T., Menhaj, M.: Training feedforward networks with the Marquardt algorithm. IEEE Transactions on Neural Networks 5(6), 989–993 (1994)
6. Beale, E.M.L.: A derivation of conjugate gradients. In: Lootsma, F.A. (ed.) Numerical methods for nonlinear optimization, pp. 39–43. Academic Press, London (1972)
7. Biegler-König, F., Bärmann, F.: A Learning Algorithm for Multilayered Neural Networks Based on Linear Least-Squares Problems. Neural Networks 6, 127–131 (1993)
8. Yam, J.Y.F., Chow, T.W.S, Leung, C.T.: A New method in determining the initial weights of feedforward neural networks. Neurocomputing 16(1), 23–32 (1997)

9. Bishop, C.M.: Neural Networks for Pattern Recognition. Oxford University Press, New York (1995)
10. Castillo, E., Fontenla-Romero, O., Alonso Betanzos, A., Guijarro-Berdiñas, B.: A global optimum approach for one-layer neural networks. Neural Computation 14(6), 1429–1449 (2002)
11. Fontenla-Romero, O., Erdogmus, D., Principe, J.C., Alonso-Betanzos, A., Castillo, E.: Linear least-squares based methods for neural networks learning. In: Kaynak, O., Alpaydın, E., Oja, E., Xu, L. (eds.) ICANN 2003 and ICONIP 2003. LNCS, vol. 2714, pp. 84–91. Springer, Heidelberg (2003)
12. Erdogmus, D., Fontenla-Romero, O., Principe, J.C., Alonso-Betanzos, A., Castillo, E.: Linear-Least-Squares Initialization of Multilayer Perceptrons Through Backpropagation of the Desired Response. IEEE Transactions on Neural Networks 16(2), 325–337 (2005)
13. Nguyen, D., Widrow, B.: Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights. In: Proceedings of the International Joint Conference on Neural Networks, vol. 3, pp. 21–26 (1990)