An Efficient Gradient-Based Learning Algorithm Applied to Neural Networks with Selective Actuation Neurons

Noel Lopes[1,2] and Bernardete Ribeiro[2]

[1]Institute Polytechnic of Guarda,
Department of Informatics, Guarda, Portugal
[2]CISUC - Centro de Informática e Sistemas,
Department of Informatics Engineering, University of Coimbra, Portugal

ABSTRACT

A new class of Neural Networks (NN), designated the Multiple Feed-Forward (MFF) networks, and a new gradient-based learning algorithm, Multiple Back-Propagation (MBP), are proposed and analyzed. MFF are obtained by integrating two feed-forward networks (a main network and a space network) in a novel manner. A major characteristic is their ability to partition the input space by using selective neurons, whose actuation role is captured through the space localisation of input pattern data. In this sense, only those neurons fired by a particular data point turn out to be relevant, while they retain the capacity to approximate closely to more general, irregular, non-linear features in localized regions. Together, the MFF networks and the MBP algorithm embody a new neural architecture, ensuring, in most cases, a better design choice than the one provided by the Multi-Layer Perceptron (MLP) networks trained with the Back-Propagation (BP) algorithm. The utilization of computable importance factors for the actuation neurons whose relative magnitudes are derived from the space network properties and the training data is the key reason for its ability to decompose the underlying mapping function into simpler sub-functions requiring parsimonious NN. Experimental results on benchmarks confirm improved efficiency of the gradient-based learning algorithm proposed, borne out by better generalization and in most cases by shorter training times for online learning, as compared with the MLP networks trained with the BP algorithm.

Keywords: Back-Propagation Algorithm, Multi-Layer Perceptrons, Hybrid Architectures, Pattern Recognition

1. INTRODUCTION

The BP algorithm has been successfully applied to a wide range of complex problems in a broad range of areas such as function approximation, pattern recognition, data mining and time series prediction, to name just a few (Haykin, 1994). It has emerged as the most widely used and popular algorithm for the design of MLP networks (Werbos, 1995; Frasconi et al., 1993). According to Werbos, in the middle of the 1990's, 70% of the real world artificial neural networks applications were developed using this algorithm (Werbos, 1995). However, slow convergence and long training times, especially when dealing with complex problems, are disadvantages often mentioned in literature (Fahlman, 1988). Several methods for improving the speed convergence of this algorithm and the generalization capabilities of the resulting networks can be found elsewhere (Almeida, 1997; Schiffmann et al., 1994). Extending these improvements, a new algorithm, designated the MBP, has recently been proposed to train a new class of NN, the MFF networks (Lopes and Ribeiro, 2001). MFF networks are obtained by integrating two feed-forward networks in such a way that one of the networks is responsible for determining the relevance of the other network's neurons, as far as each pattern is submitted to the MFF network. By selectively choosing the importance of each neuron, according to the pattern presented, the MFF network is implicitly empowered with the ability to divide the input space, assigning a different network to each sub-space. The two networks encompassing the MFF network work cooperatively, as a unit and therefore must be trained together. This can be accomplished by a gradient-based algorithm, specifically developed to train MFF networks, designated the MBP algorithm. The MFF networks trained with this algorithm offer advantages over the MLP networks trained by the BP algorithm: (*i*) better generalization capabilities, and (*ii*) shorter training times for the online training mode (Lopes and Ribeiro, 2001). Experimental results that support these advantages are presented and compared below.

This paper is organized as follows. Section 2 describes the MFF networks, while Section 3 is devoted to the detailed description of MBP algorithm. Results for benchmark data sets, in both classification and regression tasks, are set forth and discussed in section 4. Finally, section 5 presents the conclusions.

2. MULTIPLE FEED-FORWARD NETWORKS

Multiple Feed-Forward (MFF) networks encompass two feed-forward networks: a main network and a space network. The main network contains selective actuation neurons that have an importance factor, and this is determined by the space network, according to the pattern submitted to the MFF network. This factor specifies the neuron contribution to the network outputs. Figure 1 illustrates the relation between the two networks that constitute an MFF network.

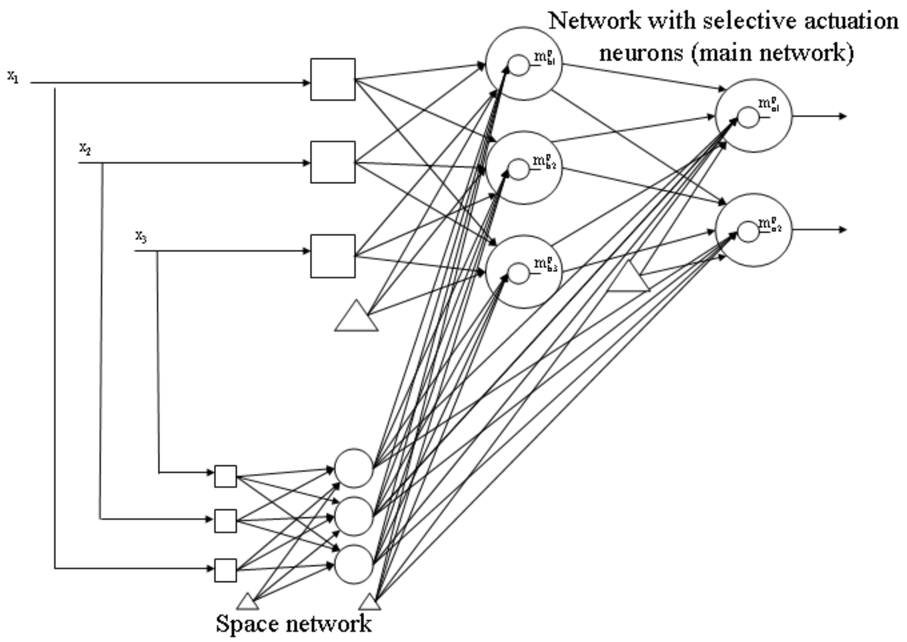


Figure 1. MFF Network: Network with selective actuation neurons (main network) and space network. Output and hidden neurons are represented by circles, input neurons by squares, and bias by triangles.

2.1. Input Space Division

When designing an NN to learn the underlying mapping function f_m of a given problem, a better fit can be obtained by properly dividing the input space into several sub-spaces and by associating an NN to each one, instead of using a single NN to cover all the input space. It is as if f_m is decomposed into several sub-functions. However, for the majority of cases, the precise knowledge necessary to appropriately decompose f_m into simpler sub-functions is not available.

Partition of the input space is not new. In fact, in the Radial Basis Function (RBF) networks it is done implicitly. When a given data pattern is presented, only the neurons from the first hidden layer, whose center is near to that pattern, are activated; this can be regarded as if the other neurons (whose center is far enough away) did not exist. In other words, it is as though each set of patterns in the same space partition has its own associated network.

Biological arguments also favour the implementation of this idea. Looking at the human brain, one can see that different *stimuli* activate different brain areas (set of neurons). This allows each area to be specialized in terms of a given set of *stimuli*. It would therefore be useful if each neuron could be specialized only relative to a given set of *stimuli* (patterns) and react only when confronted with those *stimuli*, ignoring the rest, as opposed to the MLP networks where every neuron reacts to a *stimuli*. The idea is to activate a different set of neurons for each similar set of patterns. This would allow the input space to be divided into several parts, with a different network associated to each one.

2.2. Neurons With Selective Actuation

In order to specify if, and how much, a given neuron k should contribute to the network output, when the pattern p is introduced into the network, it is possible to incorporate a variable, m_k^p, in the neuron output equation that will define its importance for the network output, according to the pattern fed to the network. Thus, assuming the neuron has N input connections, its output, y_k^p, is given by (1):

$$y_k^p = m_k^p \mathcal{F}_k(a_k^p) = m_k^p \mathcal{F}_k(\sum_{j=1}^{N} w_{jk} y_j^p + \theta_k) \ , \qquad (1)$$

where $\mathcal{F}_k$ is the activation function of the neuron k, a_k^p is the activation of the neuron k, w_{jk} represents the weight associated with the connection between neuron j and neuron k, y_j^p is the output of neuron j and θ_k is the bias of neuron k.

The neurons whose output is given by (1) are termed neurons with selective actuation, since their contribution (actuation) to NN outputs is selectively adjusted according to the actual pattern. From (1) we can see that the further m_k^p is from zero, the more important is the k neuron contribution. On the other hand, a value of zero means that the neuron will not contribute to the network output (it is as though it does not exist).

2.3. Space Network

In order to determine the importance of each neuron with selective actuation for each pattern p, i.e., the values of the various m_k^p, it is possible to use a feed-forward network, called a space network. This network will receive the same inputs as the network with selective actuation neurons (main network), and will produce the values of the various m_k^p as outputs (see figure 1). From the standpoint of of a neuron with selective actuation, therefore, some data points can be interpreted as being more important than others. A data point with high importance fires the space network neurons, which adjust the weights that the neurons with selective actuation place on these points.

According to this procedure, by determining the importance of each neuron with selective actuation for each pattern, the space network is implicitly dividing the input space. The main network can only calculate its outputs after the outputs of the space network have been obtained, thus the two networks will work cooperatively, as a unit, and must be trained together. Note that the main network can simultaneously have neurons with and without selective actuation. Figure 1 shows the relation between the two networks that form an MFF network.

Using an NN to divide the input space is similar in effect to the hierarchical mixture of experts (HME) architecture, where gating networks are used to blend the outputs of expert networks (Jordan and Jacobs, 1994).

3. MULTIPLE BACK-PROPAGATION

In the proposed MFF networks, there are two contributions to the network output errors: (*i*) the connection weights of the main network; and (*ii*) the importance given to each neuron with selective actuation, which is defined by the space network. Minimizing the error between the desired outputs and the network outputs therefore means adjusting the weights of both networks.

3.1. Updating the Main Network Weights

The main network weights are adjusted using the gradient descent method, as in the BP algorithm, which typically minimizes the quadratic error function[1] (2):

$$E^p = \frac{1}{2}\sum_{o=1}^{N_o}(d_o^p - y_o^p)^2 \ , \tag{2}$$

where N_o is the number of outputs, y_o^p is the output of neuron o for pattern p, and d_o^p is the corresponding desired output. The weights are thus adjusted by (3):

$$\Delta_p w_{jk} = \gamma \delta_k^p y_j^p + \alpha \Delta_q w_{jk} \ , \tag{3}$$

where γ is the learning rate, δ_k^p the local gradient of neuron k, $\Delta_q w_{jk}$ is the weight change w_{jk} for the last pattern q and α is the *momentum* term. The equations of the local gradient for the output and hidden neurons are given respectively by (4) and by (5):

$$\delta_o^p = (d_o^p - y_o^p) m_o^p \mathcal{F}_o^{'}(a_o^p) \ , \tag{4}$$

$$\delta_h^p = m_h^p \mathcal{F}_h^{'}(a_h^p) \sum_{o=1}^{N_o} \delta_o^p w_{ho} \ . \tag{5}$$

Equations (3), (4) and (5) allow recursively to adjust the weights of the main network. Note that if we consider all the m_k^p to be constant and equal to 1, i.e., that all neurons are equally important regardless of the pattern, we can verify that equations (4) and (5) are identical to the corresponding BP equations. Thus the MBP algorithm can be viewed as a generalization of BP algorithm.

[1]Online training mode is considered.

3.2. Updating the Importance of the Neurons with Selective Actuation

In an MFF network, the importance given to each neuron, according to the pattern introduced into the network, contributes to the network output errors. Therefore the space network weights must be adjusted. By doing this, we are changing the way the input space is divided in order to find the proper space division. The variation that the importance of a given neuron k should undergo, when the pattern p is introduced, $\Delta_p m_k^p$, can be obtained by using the gradient descent method, as stated in (6):

$$\Delta_p m_k^p = -\frac{\partial E^p}{\partial m_k^p} \ . \tag{6}$$

In order to calculate (6) it is necessary to separate the computations of the variations in the importance of the output neurons from the computations of the variations in the importance of the hidden neurons. It is then easy to see that equation (6) becomes (7) and (8) respectively for output and hidden neurons:

$$\Delta_p m_o^p = (d_o^p - y_o^p)\mathcal{F}_o(a_o^p) \ , \tag{7}$$

$$\Delta_p m_h^p = \sum_{o=1}^{N_o} \delta_o^p w_{ho} \mathcal{F}_h(a_h^p) \ . \tag{8}$$

Equations (7) and (8) define the errors associated with the output neurons of the space network. They thus allow updating of the space network weights by means of the BP algorithm.

3.3. Putting All Together

After introducing a pattern into the MFF network, the space network determines the importance of each neuron with selective actuation. The main network will then process that input pattern and calculate its outputs. During the training phase, the next step is to calculate the variation of the importance of the selective actuation neurons and to update the main network weights. Only after this step can the space network weights be updated, using the BP algorithm or any other supervised algorithm, leading, in the latter case, to a hybrid architecture.

4. EXPERIMENTAL RESULTS AND DISCUSSION

In order to determine the feasibility of the proposed architecture a program package was developed in C++ (Lopes, 2001) to implement the MBP algorithm for the training of MFF networks. For comparison purposes the program also allows the MLP networks, to be trained with the BP algorithm, so that we can rely on the results obtained.

4.1. Experimental Setup Conditions

Although the program developed admits a wide variety of training configurations, for the experimental setup, it was nevertheless decided to use an adaptive learning step size technique (Almeida, 1997), with a given increment u and a given decrement d. A robust training method, described in algorithm 1, was implemented such that the step sizes were decreased by a reducing factor r each time the RMS error increased more than 0.1%. The step sizes, γ, of the networks were initialized to 0.7 and the *momentum* terms[2], α to 0.7. The latter were updated automatically every 300 epochs, in accordance with algorithm 2.

[2]Note that a MFF network can have two *momentum* terms: α_m for the main network and α_s for the space network.

Several running tests were performed, with a view to investigating the capabilities of the new architecture and comparing it with the MLP networks trained by the BP algorithm. The study includes the XOR and the n bit parity problem as well as five other benchmarks (two classification, two regression and one prediction problem). For each benchmark, the networks tested were trained in a Pentium III at 650Mhz, until the Root Mean Square (RMS) error for the training examples was less than a predefined value. The RMS error was defined as follows:

$$RMS = \frac{1}{2}\sqrt{\frac{1}{N_p N_o}\sum_{p=1}^{N_p}\sum_{o=1}^{N_o}(d_o^p - y_o^p)^2}\ , \tag{9}$$

where N_p represents the number of patterns.

The results obtained for each network were accumulated over 10 runs. The generalization performance was then evaluated for the test set using the RMS error and the classification rate.

Except where explicitly stated otherwise, all the networks in the experimental setup had two layers and the activation function chosen for their neurons was the logistic function. In all the MFF networks the space network had just one layer responsible for determining the importance of the selective actuation neurons that were placed in the first, hidden, layer of the main network. For the networks trained in online mode, examples were presented randomly.

Algorithm 1 Robustly trains a neural network during one epoch.

```
Let $net$ represent the neural network being trained
Let $\alpha$ represent the momentum term
Let $r$ be the reducing factor
Let $tolerance$ be the adimissible percentage of growth in the neural network error, for the training examples, without implying a reduction in the step sizes
Let $train(network, momentum)$ be a function that trains a given network during one epoch, using a given momentum, and returns the trained network error
Let $E_{actual}$ be the actual network error, for the training examples
Let $maxTrials$ be the maximum number of trials for the step size reduction

$net_{old} \leftarrow net$
$E_{new} \leftarrow train(net, \alpha)$
$trials \leftarrow 0$
while $trials < maxTrials$ and $E_{new} > (1 + tolerance)E_{actual}$ do
  {Reduce the step sizes of the network by the factor $r$}
  for each step size $\gamma$ in $net_{old}$ do
    $\gamma \leftarrow r\gamma$
  end for

  $net \leftarrow net_{old}$
  $E_{new} \leftarrow train(net, 0.0)$
  $trials \leftarrow trials + 1$
end while
$E_{actual} \leftarrow E_{new}$
```

Algorithm 2 Trains a neural network during one epoch and updates the *momentum* term every *interval*.

```
Let $net$ represent the neural network being trained
Let $t$ represent the current training epoch
Let $\alpha$ represent the momentum term
Let $interval$ be the interval of epochs in which $\alpha$ is updated
Let $\omega$ be the percentage variation that $\alpha$ should undergo, if by varying $\alpha$, a redution in the training error occurs
Let $\phi$ be the variation that $\omega$ should undergo after each update of $\alpha$
Let $rtrain(network, momentum)$ be a function that robustly trains a given network during one epoch, using a given momentum, and returns the trained network error

$net_{old} \leftarrow net$
$E_{actual} \leftarrow rtrain(net, \alpha)$
if $t$ is multiple of $interval$ then
  $net \leftarrow net_{old}$
  $E_{new} \leftarrow rtrain(net, \alpha(1-\omega))$
  if $E_{new} < E_{actual}$ then
    $E_{actual} \leftarrow E_{new}$
    $\alpha \leftarrow \alpha(1-\omega)$
    $\omega \leftarrow \omega + \phi$
  else
    $net \leftarrow net_{old}$
    $E_{new} \leftarrow rtrain(net, \alpha(1+\omega))$
    if $E_{new} < E_{actual}$ then
      $E_{actual} \leftarrow E_{new}$
      $\alpha \leftarrow \alpha(1+\omega)$
      $\omega \leftarrow \omega + \phi$
    else
      $net \leftarrow net_{old}$
      $E_{actual} \leftarrow rtrain(net, \alpha)$
      if $\omega > \phi$ then
        $\omega \leftarrow \omega - \phi$
      end if
    end if
  end if
end if
```

4.2. The Exclusive OR (XOR) and the Parity Problems

A single neuron without selective actuation cannot solve the XOR problem, since it cannot classify input patterns that are not linearly separable (Haykin, 1994). On the contrary, only one selective actuation neuron can solve the XOR problem as shown in figure 2. Table 1

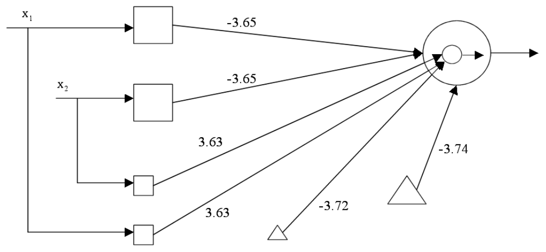


Figure 2. Selective actuation neuron for the XOR problem with the hyperbolic tangent funtion.

shows that the importance of the neuron with selective actuation, shown in the figure 2, is almost 1 when both inputs are set to 1 and almost −1 otherwise. Thus, it is as if the input space is divided into two parts, each one with a different associated network[3], as shown in figure 3.

Table 1. Importance (m) and output (y) of the neuron with selective actuation.

$\mathbf{x_1}$	$\mathbf{x_2}$	**m**	**y**
−1	−1	−1.0000	−0.9984
−1	1	−0.9988	0.9977
1	−1	−0.9988	0.9977
1	1	0.9983	−0.9983

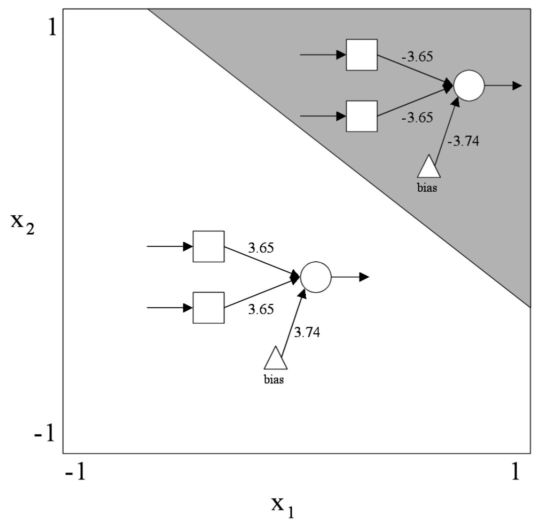


Figure 3. Input space division produced by a neuron with selective actuation.

In MLP networks, non-linearity is obtained exclusively by the use of non-linear activation functions; in MFF networks, however, non-linearity is also obtained through the division of the input space. In fact, a selective actuation neuron can solve the XOR problem or any other bit parity problem even if the activation functions used are linear.

[3]Note that the hyperbolic tangent is asymmetric, i.e. $tanh(-x) = -tanh(x)$.

4.3. The Two-Spirals Benchmark

The two-spirals benchmark was downloaded from the CMU repository of machine learning database (CMU, 2001). The task, which is considered extremely hard for algorithms of the BP family to solve (Fahlman and Lebiere, 1990), consists of learning to discriminate between two sets of training points which lie on two distinct spirals on the x-y plane. The spirals coil three times around the origin and around one another. For building the training set the spirals were generated using a density of 1, resulting in 194 training examples (97 of each class). For the test set, the spirals were generated using a density of 4, providing 770 testing examples (350 of each class). Each example is composed of an x-y coordinate (2 inputs) and one output which specifies the associated class.

Table 2 identifies the neural network topologies and the learning algorithms used in the experimental setup. All the networks listed had 3 layers, with the number of neurons of the second hidden layer being constant and set to 10, for all the networks. In the training setup the values of u, d and r were set respectively to 1.05, 0.95 and 0.75. The networks were then trained until the RMS for the training examples was less than 0.01. The two algorithms (BP and MBP) were compared in terms of convergence speed and interpolation capabilities.

Table 2. Neural networks characterization layout.

Network	First Hidden Layer Neurons	Network identification for Batch Training	Network identification for Online Training
MLP	25	BP$_{b25}$	BP$_{o25}$
MLP	30	BP$_{b30}$	BP$_{o30}$
MLP	35	BP$_{b35}$	BP$_{o35}$
MFF	15	MBP$_{b15}$	MBP$_{o15}$
MFF	20	MBP$_{b20}$	MBP$_{o20}$
MFF	25	MBP$_{b25}$	MBP$_{o25}$

Table 3 shows the mean, the standard deviation (Std.), the minimum (Min.) and maximum (Max.) values with respect to the number of epochs and to the time required to train the networks. Table 3 confirms that the MBP$_{o15}$ and MBP$_{o20}$ networks present the best average, minimum and maximum training times. This shows that the MBP is much faster than the BP for online training, despite not being as fast as the BP for batch training. Unlike the BP, the MBP algorithm is stochastic, even in batch mode, since changes in the space network weights affect the main network by changing the point in the surface error where the gradient is calculated. Similarly, alterations in the main network weights modify the space network error surface. Thus, as the MBP is stochastic by nature, it is usually more pratical to train in online mode, since it converges faster towards a solution.

Table 4 shows the RMS error for the test set and the percentage of examples correctly classified by the trained networks. Since the MBP divides the input space we may expect better interpolation capabilities for the resulting networks. In fact, table 4 shows that the MBP$_{o15}$, MBP$_{o20}$ and MBP$_{o25}$ networks present the best mean and minimum classification rates. Thus the MFF networks trained with the MBP in online mode present much better classification rates than the MLP networks trained with the BP algorithm. However, the same does not seem to apply when the MFF networks are trained in batch mode. This is because the input space is not yet correctly divided, whereas with further training, the results should improve as long as the input space is better divided.

Table 3. Two-spirals problem training data results.

	Epochs				Training Time			
NNet	Mean	Std.	Min.	Max.	Mean	Std.	Min.	Max.
BP_{b25}	22311	18083	8556	64078	0:23:36	0:19:09	0:09:03	1:07:53
BP_{b30}	10845	1558	8579	13408	0:13:26	0:01:56	0:10:37	0:16:38
BP_{b35}	14923	15328	8380	58331	0:21:11	0:21:47	0:11:54	1:22:52
MBP_{b15}	34176	28954	12089	98045	0:34:40	0:29:23	0:12:14	1:39:29
MBP_{b20}	19982	7202	13495	38013	0:25:55	0:09:22	0:17:26	0:49:20
MBP_{b25}	18880	8954	13058	40278	0:29:46	0:14:07	0:20:35	1:03:29
BP_{o25}	4449	3083	1848	12300	0:05:20	0:03:33	0:02:15	0:14:13
BP_{o30}	7316	4508	3166	16805	0:10:25	0:06:22	0:04:32	0:23:55
BP_{o35}	6061	5240	1548	19245	0:09:46	0:08:03	0:02:33	0:29:35
MBP_{o15}	2395	754	1264	3710	0:02:47	0:00:53	0:01:28	0:04:20
MBP_{o20}	2030	727	1364	3978	0:03:00	0:01:02	0:02:01	0:05:45
MBP_{o25}	3074	3514	1695	13043	0:05:27	0:06:01	0:03:03	0:22:31

Table 4. Performance criteria (RMS and classification rate) on the two-spirals testing data.

	RMS				Classification Rate (%)			
NNet	Mean	Std.	Min.	Max.	Mean	Std.	Min.	Max.
BP_{b25}	0.0474	0.0148	0.0122	0.0617	98.91	0.60	98.16	100.00
BP_{b30}	0.0453	0.0213	0.0104	0.0814	98.71	1.06	96.75	100.00
BP_{b35}	0.0381	0.0220	0.0151	0.0849	99.11	1.06	96.75	100.00
MBP_{b15}	0.0514	0.0236	0.0115	0.0901	98.49	1.15	96.32	100.00
MBP_{b20}	0.0535	0.0198	0.0174	0.0913	98.44	1.12	95.76	99.86
MBP_{b25}	0.0486	0.0260	0.0171	0.0965	98.61	1.37	95.62	100.00
BP_{o25}	0.0364	0.0165	0.0110	0.0641	99.38	0.55	98.16	100.00
BP_{o30}	0.0408	0.0180	0.0131	0.0610	99.05	0.81	97.88	100.00
BP_{o35}	0.0319	0.0165	0.0122	0.0637	99.42	0.61	98.16	100.00
MBP_{o15}	0.0292	0.0109	0.0130	0.0462	99.58	0.31	99.01	100.00
MBP_{o20}	0.0273	0.0112	0.0108	0.0437	99.60	0.33	99.01	100.00
MBP_{o25}	0.0338	0.0163	0.0106	0.0700	99.43	0.35	98.87	100.00

Figures 4 and 5 illustrate, respectively, the network output and the decision boundaries of the MBP$_{o20}$ for the "two-spirals" classification problem.

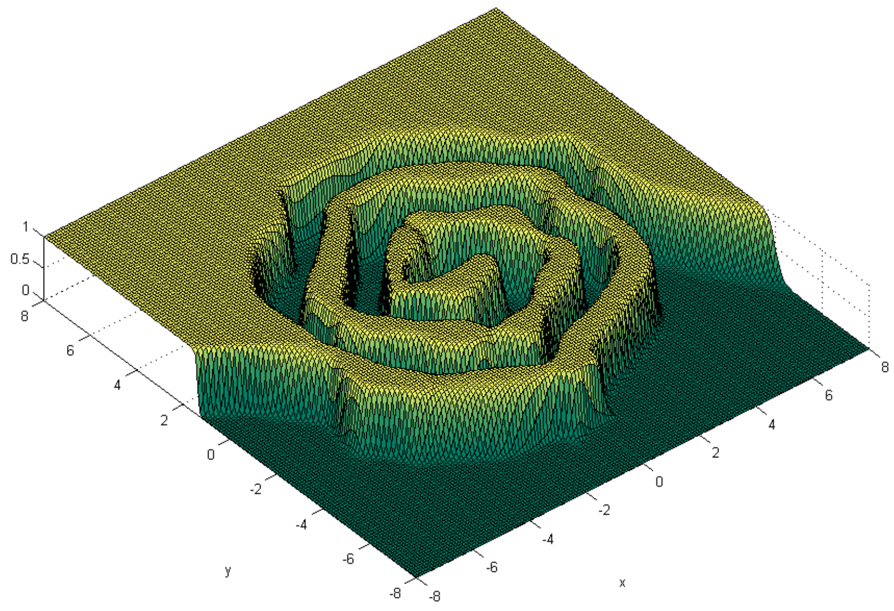


Figure 4. Network output (MBP$_{o20}$) for the spiral classification problem.

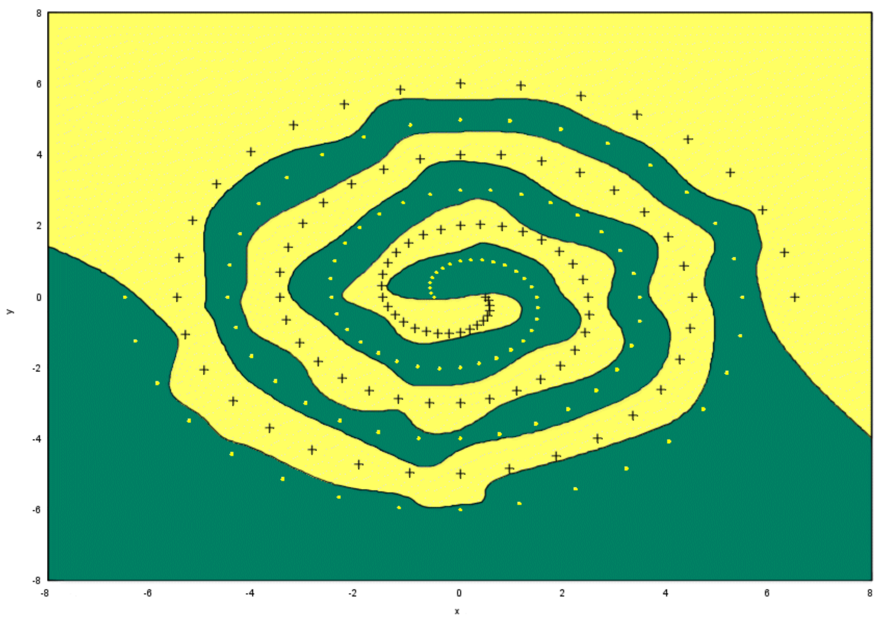


Figure 5. Decision boundaries (MBP$_{o20}$ network) for the spiral classification problem.

4.4. The Sonar Benchmark

The sonar benchmark was also acquired from the CMU repository of machine learning database (CMU, 2001). It is a classification problem where the task consists of training a network in order to discriminate between sonar signals bounced off a metal cylinder and those bounced off a roughly cylindrical rock. There are 104 training examples and 104 test examples. Each exemplar is composed of 60 continuous inputs and 2 outputs.

Several neural network topologies and learning algorithms were used, although the number of layers remained at two for all the networks tested. The activation functions used were chosen as (*i*) the hyperbolic tangent, for the hidden neurons, and (*ii*) the logistic function for the output neurons, including those in the space network. In the training setup, the values of u and d were set respectively to 1.01 and 0.98; the value of r was 0.98 for the networks trained in batch mode and 0.75 for the networks trained in online mode. Again, the networks were trained until the RMS for the training examples was less than 0.01. The two algorithms (BP and MBP) were then compared in terms of speed of convergence and generalization capabilities. Table 5 shows the results in terms of the number of epochs and of the time required to train the networks. Note that the neural networks characterization layout used is similar to the one presented in Table 2, and it is not here presented for simplification reasons. Thus, for example, BP_{b10} represents an MLP network with 10 hidden neurons, trained with the BP algorithm in batch mode. Networks MBP_{o3}, MBP_{o5} and

Table 5. Results on the sonar training data.

	Epochs				Training Time			
NNet	Mean	Std.	Min.	Max.	Mean	Std.	Min.	Max.
BP_{b10}	1053	266	777	1736	0:00:45	0:00:12	0:00:33	0:01:15
BP_{b12}	826	110	676	1065	0:00:42	0:00:06	0:00:34	0:00:54
BP_{b14}	809	63	713	893	0:00:48	0:00:04	0:00:42	0:00:52
MBP_{b3}	2090	834	1138	3985	0:01:04	0:00:23	0:00:38	0:01:59
MBP_{b5}	1615	1139	1104	4839	0:01:14	0:00:53	0:00:51	0:03:43
MBP_{b7}	1033	173	787	1323	0:01:05	0:00:12	0:00:48	0:01:24
BP_{o10}	148	67	64	268	0:00:07	0:00:03	0:00:03	0:00:13
BP_{o12}	137	59	81	262	0:00:08	0:00:03	0:00:05	0:00:15
BP_{o14}	173	122	66	470	0:00:12	0:00:08	0:00:05	0:00:31
MBP_{o3}	194	32	133	240	0:00:06	0:00:01	0:00:04	0:00:08
MBP_{o5}	126	36	76	178	0:00:06	0:00:02	0:00:04	0:00:09
MBP_{o7}	98	24	70	134	0:00:07	0:00:02	0:00:05	0:00:10

MBP_{o7} present the best mean and maximum training times, confirming the results obtained for the previously studied two-spirals benchmark, indicating once more that although the MBP is not as fast as the BP for batch training, it is faster than the BP for online training.

The results in table 6 show that, in general, the MFF networks trained with the MBP algorithm have better generalization capabilities than the MLPs trained with the BP algorithm. This is valid for both online and batch training modes. In fact, the best classification result (92.31%) was obtained for an MFF network trained in batch mode, namely, MBP_{b5}. The minimum RMS error (0.1305) is found in the corresponding table entry.

Table 6. Results (RMS error and classification rates) on the sonar testing data.

	RMS				Classification Rate (%)			
NNet	Mean	Std.	Min.	Max.	Mean	Std.	Min.	Max.
BP_{b10}	0.1788	0.0223	0.1550	0.2001	84.13	2.23	80.77	87.50
BP_{b12}	0.1683	0.0297	0.1512	0.1929	85.38	2.97	80.77	89.42
BP_{b14}	0.1731	0.0281	0.1515	0.1943	84.71	2.81	80.77	88.46
MBP_{b3}	0.1867	0.0396	0.1558	0.2165	83.08	3.96	76.92	88.46
MBP_{b5}	0.1694	0.0355	0.1305	0.1994	85.96	3.55	81.73	92.31
MBP_{b7}	0.1672	0.0333	0.1388	0.1848	85.87	3.33	81.73	91.35
BP_{o10}	0.1755	0.0166	0.1425	0.1971	85.48	3.02	81.73	91.35
BP_{o12}	0.1759	0.0107	0.1610	0.1916	85.19	1.93	82.69	88.46
BP_{o14}	0.1792	0.0110	0.1653	0.2034	84.23	2.23	78.85	86.54
MBP_{o3}	0.1672	0.0121	0.1510	0.1869	86.92	2.36	82.69	90.38
MBP_{o5}	0.1702	0.0179	0.1430	0.1974	85.87	2.72	82.69	90.38
MBP_{o7}	0.1720	0.0113	0.1582	0.1942	85.19	2.28	80.77	88.46

4.5. The function $f(x) = sin(x)/x$

In this regression problem the goal is to design a network capable of approximating the function $f(x) = sin(x)/x$. In order to construct the training set, 100 values, uniformly distributed in the interval $[-10, 10]$, were computed. Several neural network topologies and learning algorithms were used in the experimental setup. In the training setup the values of u, d and r were set respectively to 1.1, 0.9 and 0.75. The networks were then trained until the RMS for the training examples was less than 0.01 and the two algorithms (BP and MBP) were compared in terms of convergence speed.

Table 7 shows that the MBP_{o7} and MBP_{o9} networks present the best mean and maximum training times (the MBP_{o9} network also exhibits the smallest minimum training time), confirming once again that the MBP is faster than the BP for the online training mode, despite being slower for the batch training mode.

Table 7. Results on the approximation of the function $f(x) = sin(x)/x$.

	Epochs				Training Time			
NNet	Mean	Std.	Min.	Max.	Mean	Std.	Min.	Max.
BP_{b7}	7296	2317	5397	12291	0:00:25	0:00:08	0:00:18	0:00:42
BP_{b9}	6050	745	5298	7434	0:00:25	0:00:03	0:00:22	0:00:31
BP_{b11}	5650	290	5188	6198	0:00:28	0:00:01	0:00:26	0:00:31
MBP_{b5}	9507	2811	6704	15201	0:00:48	0:00:14	0:00:33	0:01:16
MBP_{b7}	8259	1193	6658	9915	0:00:54	0:00:08	0:00:44	0:01:06
MBP_{b9}	6966	1296	5103	9224	0:00:57	0:00:11	0:00:41	0:01:15
BP_{o7}	7587	6731	496	19680	0:00:31	0:00:27	0:00:02	0:01:18
BP_{o9}	3654	4518	499	12763	0:00:18	0:00:23	0:00:04	0:01:06
BP_{o11}	5612	3558	1372	14464	0:00:35	0:00:22	0:00:08	0:01:29
MBP_{o5}	3927	3105	1073	10490	0:00:24	0:00:18	0:00:06	0:01:03
MBP_{o7}	1597	588	743	2349	0:00:13	0:00:05	0:00:06	0:00:19
MBP_{o9}	1281	624	483	2385	0:00:13	0:00:06	0:00:02	0:00:24

4.6. The function $f(x,y) = 1 - \frac{y^2}{100} sin(\frac{x}{3})$

In this regression problem the goal is to create of a network capable of approximating the function $f(x,y) = 1 - \frac{y^2}{100} sin(\frac{x}{3})$. In order to create the training set, x and y were varied between -10 and 10 with an increment of 0.5 A total of 1681 training examples was obtained. Figure 6 shows the graphic of the function in the refered interval.

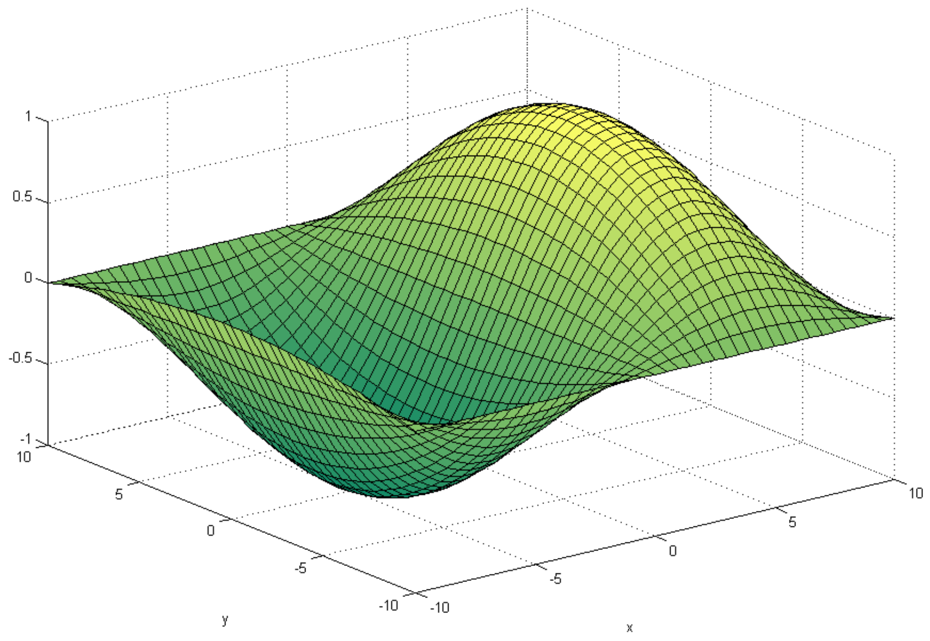


Figure 6. Function $f(x,y) = 1 - \frac{y^2}{100} sin(\frac{x}{3})$.

Given the size of the training set, and since the results for the previous benchmarks have demonstrated that the MBP algorithm is slower than the BP algorithm for the batch training mode, we decided to train all the tested networks using the online training mode. Several neural network topologies and learning algorithms were selected for experimentation. The configuration setup parameters u, d and r were set respectively to 1.1, 0.9 and 0.5. The networks were then trained until the RMS for the training examples was less than 0.01 and the two algorithms (BP and MBP) were compared in terms of speed convergence.

The results relative to epochs and training time required to train the networks are shown in table 8. Although the results are not so clear-cut as the ones obtained for previous benchmarks, they still indicate that the MBP requires a smaller number of epochs to train the networks than the BP. Actually the average time required to train the MBP$_{o7}$ network is lower than the time required to train any other network. Figure 7 depicts the approximation error between the desired output and the network output response.

Table 8. Average results for the two-variable function $f(x, y) = 1 - \frac{y^2}{100} sin(\frac{x}{3})$.

	Epochs				Training Time			
Net	Mean	Std.	Min.	Max.	Mean	Std.	Min.	Max.
BP_{o7}	2365	779	1536	4041	0:03:37	0:01:12	0:02:20	0:06:11
BP_{o9}	2360	921	908	4041	0:03:37	0:01:24	0:01:24	0:06:09
BP_{o11}	1958	1984	389	7456	0:03:34	0:03:37	0:00:43	0:13:36
MBP_{o7}	1513	750	363	2913	0:03:33	0:01:45	0:00:52	0:06:52
MBP_{o8}	1801	328	1301	2466	0:04:50	0:00:52	0:03:25	0:06:30
MBP_{o9}	1857	540	1133	2583	0:05:31	0:01:36	0:03:24	0:07:41

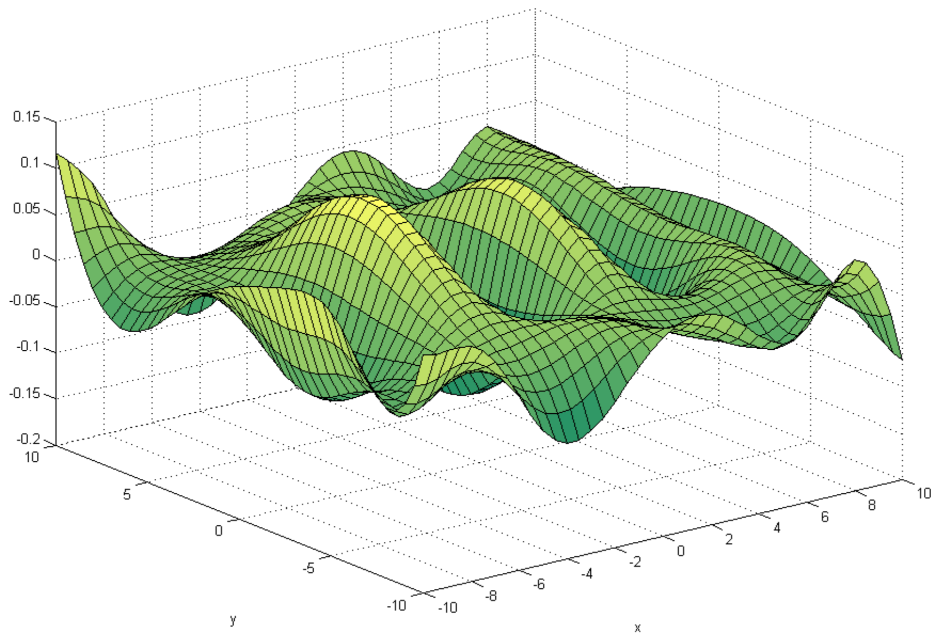


Figure 7. Approximation error for $f(x, y) = 1 - \frac{y^2}{100} sin(\frac{x}{3})$ obtained with the MBP_{o7} network.

4.7. The Box & Jenkins Gas Furnace Problem

The Box-Jenkins gas furnace benchmark (Box and Jenkins, 1970) is a time series prediction problem often used for non-linear system identification. This benchmark data set consists of 296 input-output measurements of gas flow rate $x(t)$ (input) and CO_2 concentration $y(t)$ (output). In order to predict $y(t)$, 10 inputs were considered: $x(t), ..., x(t-6), y(t-1), ..., y(t-3)$. This reduces the number of examples to 290, of which the first 150 were used in the training set and the remaining 140 in the test set.

In the training setup the values of u, d and r were set respectively to 1.1, 0.9 and 0.85. The networks were then trained until the RMS for the training examples was less than 0.005 and the two algorithms (BP and MBP) compared in terms of speed of convergence and generalization capabilities.

Table 9 illustrates the number of epochs and training time required to train the networks, to predict the CO_2 concentration in the gases expelled by a furnace, until the RMS error for the training data was less than 0.005. In this case, contrary to the previous benchmarks, the BP algorithm converges faster than the MBP algorithm, even for the online training mode. The reason might be that the difference between the number of neurons in the main MFF network and those in the MLP network is negligible.

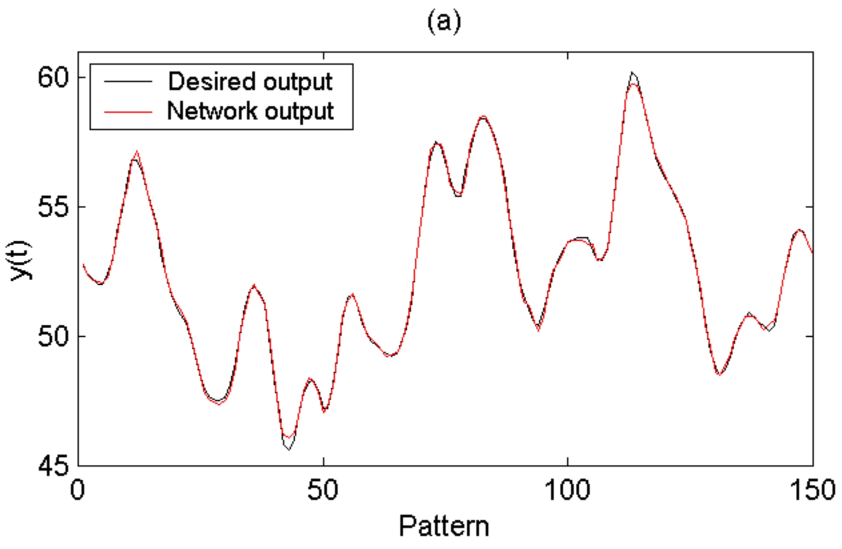
Table 9. Results for the Box-Jenkins gas furnace benchmark training data.

	Epochs				Training Time			
NNet	Mean	Std.	Min.	Max.	Mean	Std.	Min.	Max.
BP_{b3}	9093	4524	6364	21528	0:00:50	0:00:25	0:00:35	0:01:59
BP_{b4}	7338	625	6618	8779	0:00:50	0:00:04	0:00:45	0:01:00
BP_{b5}	7101	266	6843	7543	0:00:57	0:00:02	0:00:55	0:01:01
MBP_{b2}	13364	1165	11838	14849	0:01:49	0:00:09	0:01:37	0:02:01
MBP_{b3}	12382	589	11821	13834	0:02:13	0:00:06	0:02:07	0:02:28
MBP_{b4}	12038	459	11757	13237	0:02:41	0:00:06	0:02:37	0:02:57
BP_{o3}	2658	1821	1213	6257	0:00:18	0:00:13	0:00:08	0:00:42
BP_{o4}	1657	1046	1005	4576	0:00:14	0:00:09	0:00:08	0:00:39
BP_{o5}	1800	1954	858	7314	0:00:18	0:00:20	0:00:09	0:01:14
MBP_{o2}	24069	24908	1164	71720	0:03:50	0:03:53	0:00:22	0:11:16
MBP_{o3}	2762	3025	892	11048	0:00:37	0:00:38	0:00:12	0:02:22
MBP_{o4}	1238	409	878	2048	0:00:21	0:00:07	0:00:15	0:00:36

The results in terms of generalization performance are given in table 10. From this table we observe that MBP_{b2} and MBP_{o2} networks have the smallest mean errors over the testing data, with the best error being 0.136, achieved by the MBP_{b2} network. Figures 8a and 8b show the output of the MBP_{b2} network for the training and testing data sets. Once again, these results emphasize that the generalization capabilities of the MFF networks, trained with the MBP algorithm, outperform the generalization capabilities of the MLP networks, trained with the BP algorithm.

Table 10. RMS error for the Box & Jenkins test data.

	RMS			
NNet	Mean	Std.	Min.	Max.
BP_{b3}	0.0163	0.0010	0.0141	0.0174
BP_{b4}	0.0164	0.0012	0.0145	0.0182
BP_{b5}	0.0163	0.0018	0.0137	0.0205
MBP_{b2}	0.0156	0.0020	0.0136	0.0204
MBP_{b3}	0.0165	0.0012	0.0152	0.0186
MBP_{b4}	0.0165	0.0013	0.0140	0.0179
BP_{o3}	0.0179	0.0012	0.0156	0.0197
BP_{o4}	0.0171	0.0011	0.0148	0.0186
BP_{o5}	0.0173	0.0012	0.0153	0.0198
MBP_{o2}	0.0157	0.0007	0.0143	0.0167
MBP_{o3}	0.0180	0.0018	0.0152	0.0210
MBP_{o4}	0.0186	0.0015	0.0165	0.0219



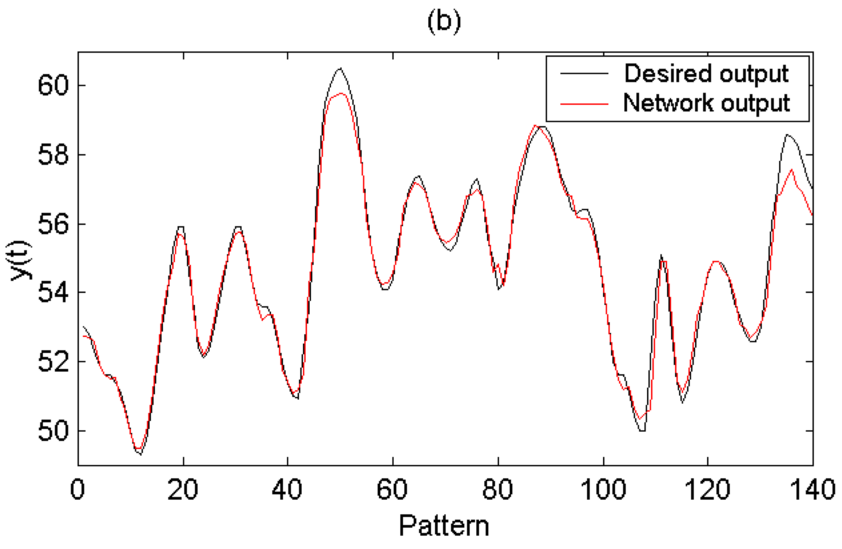


Figure 8. Desired output *versus* MBP_{b2} network output for the Box & Jenkins on (a) the training data and (b) on the testing data.

4.8. Discussion of Results

The number of hidden neurons used in an MLP network is one of the most critical parameters regarding the design of a neural solution. Basically, the more hidden units are used, the greater is the probability of avoiding local minima during the training phase. This, however, is accomplished by reducing the generalization capabilities of the network, leading to the bias-variance dilemma. By varying the number of hidden neurons, two extreme solutions, both leading to poor generalization cases, can be obtained: (*i*) a solution whereby the network accommodates both the underlying mapping function and the noise inherent to the training data (overfitting) and (*ii*) a solution where the local characteristics of the mapping function are ignored (underfitting), due to an insufficient number of hidden neurons.

One of the advantages of MFF networks over MLP networks relies on the fact that, in general they require fewer hidden neurons, in the main network, than the MLP networks require. In fact, learning a given mapping function is achieved in the MFF networks while they retain the capacity to approximate closely to more general, irregular, non-linear characteristics in localized regions. Thus, when using the new architecture, it is usually possible to obtain better solutions (networks that generalize better), than those attainable by MLP networks trained with the BP algorithm.

5. CONCLUSION

A new architecture, consisting of the MFF networks and the MBP algorithm, has been proposed and analyzed. The new class of networks (MFF) contain selective actuation neurons, whose contribution to the network output depends on the space distribution of input patterns. The MBP hybrid learning algorithm for the proposed architecture can be seen as a generalization of the well known BP algorithm, and it denotes good localization properties and parsimony of the function representation, exploring fine details in highly non-linear characteristics. The power of the MBP comes from its ability to approximate closely to more general, irregular, non-linear characteristics in localized regions, without too much effort. Benchmark experiments show that the MFF networks trained with the MBP algorithm have better generalization properties than the MLP networks trained with the BP. This is due to their ability to decompose the underlying mapping function into simpler sub-functions that require simpler NN, which is implicitly achieved by the role assigned to the selective actuation neurons. In fact, these neurons allow MFF networks to use fewer hidden neurons in the main network than are needed in an MLP network, while retaining the capacity to approximate closely to more general, irregular, non-linear characteristics. Thus, when using the new architecture, it is usually possible to obtain better design solutions than those given by the MLP networks trained by the BP algorithm. Furthermore, the MBP is usually faster than the BP for the online training mode, making this new architecture very attractive. While this method can provide a functionality that is similar to that of the RBF networks, it also offers other advantages. The MBP is more flexible in fitting arbitrary functions. Another advantage is the realization of the network in hardware, since sigmoidal type units are simpler and easier to implement. Also, the MBP is based on dot product calculation, which is much simpler to implement in hardware than the Euclidean distance calculation that is required with the RBF. Future work will attempt to test the proposed architecture with a different network structure and/or training algorithm relative to the space network.

ACKNOWLEDGMENTS

Center of Informatics and Systems of University of Coimbra (CISUC) and Institute Polithecnic of Guarda (IPG) are gratefully acknowledged for partial financing support.

REFERENCES

1. Almeida, L. B. (1997). *Handbook of Neural Computation*, chapter C1.2 Multilayer perceptrons, pages C1.2:1–C1.2:30. IOP Publishing Ltd and Oxford University Press.

2. Becker, S. and LeCun, Y. (1989). Improving the convergence of back-propagation learning with second-order methods. In Touretzky, D., Hinton, G., and Sejnowski, T., editors, *Proceedings of the 1988 Connectionist Models Summer School*, pages 29–37, San Mateo. Morgan Kaufman.

3. Box, G. E. P. and Jenkins, G. M. (1970). *Time Series Analysis: Forecasting and Control.* San Francisco, Holden-Day.

4. CMU (2001). Carnegie Mellon University neural networks benchmark collection [online]. Available from World Wide Web: http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/ai-repository/ai/areas/neural/bench/cmu/0.html.

5. Fahlman, S. E. (1988). An empirical study of learning speed in back-propagation networks. Computer Science Technical Report CMU-CS-88-162, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

6. Fahlman, S. E. and Lebiere, C. (1990). The cascade-correlation learning architecture. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems*, volume 2, pages 524–532, Denver 1989. Morgan Kaufmann, San Mateo.

7. Frasconi, P., Gori, M., and Tesi, A. (1993). Successes and failures of backpropagation: A theoretical investigation. In Omidvar, O., editor, *Progress in Neural Networks*, pages 205–242. Ablex Publishing.

8. Haykin, S. (1994). *Neural Networks: A Comprehensive Foundation.* Prentice-Hall.

9. Jordan, M. I. and Jacobs, R. A. (1994). Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, 6(2):181–214.

10. Kecman, V. (2001). *Learning and Soft Computing : Support Vector Machines, Neural Networks and Fuzzy Logic Models.* The MIT Press, Cambridge, MA.

11. LeCun, Y., Bottou, L., Orr, G., and Muller, K. (1998). Efficient backprop. In Orr, G. and K., M., editors, *Neural Networks: Tricks of the trade.* Springer.

12. Lopes, N. and Ribeiro, B. (2001). Hybrid learning in a multi neural network architecture. In *INNS-IEEE International Joint Conference on Neural Networks, IJCNN'01*, volume 4, pages 2788–2793, Washington D.C., USA.

13. Schiffmann, W., Joost, M., and Werner, R. (1994). Optimization of the backpropagation algorithm for training multilayer perceptrons. Technical report, University of Koblenz, Institute of Physics, Rheinau 1, 56075 Koblenz, Germany.

14. Schmidhuber, J. (1989). Accelerated learning in back-propagation nets. In R. Pfeifer, Z. Schreter, Z. F. and Steels, L., editors, *Connectionism in Perspective*, pages 429–438, Amsterdam, North-Holland. Elsevier.

15. Werbos, P. J. (1995). Backpropagation: Basics and new developments. In Arbib, M. A., editor, *The Handbook of Brain Theory and Neural Networks*, chapter Part III: Articles, pages 134–139. The MIT Press.

16. Lopes, N. (2001). Arquitecturas Híbridas de Aprendizagem em Redes Neuronais: Estudo e Aplicação a um Caso Prático. MSc. Thesis. University of Coimbra, Coimbra, Portugal.

NOTATION

f_m	Underlying mapping function of a given problem.
k	Neuron.
h	Hidden neuron.
o	Output neuron.
p	Current input pattern.
q	Last input pattern.
N_p	Number of input patterns.
N	Number of input connections of a neuron, not including the bias.
w_{jk}	Weight associated with the connection between neuron j and neuron k.
θ_k	Bias of neuron k.
a_k^p	Activation of the neuron k.
$\mathcal{F}_k$	Activation function of the neuron k.
m_k^p	Importance of neuron k, when the pattern p is submitted to the network.
y_k^p	Output of neuron k, when the pattern p is submitted to the network.
N_o	Number of outputs of the neural network.
d_o^p	Desired output of neuron o, when pattern p is submitted to the network.
E^p	Network error for the pattern p
$\Delta_p w_{jk}$	Adjust made to the weight, associated with the connection between neuron j and neuron k, when pattern p is submitted to the network.
γ	Learning rate or step size.
α	*momentum* term.
δ_k^p	Local gradient of neuron k, for pattern p.
$\Delta_p m_k^p$	Adjust made to the importance of neuron k for pattern p.
RMS	Root mean square error.
r	Reducing factor (robust training).