Preface

Artificial neural networks (ANNs) offer a general framework for representing non-linear mappings from several input variables to several output variables, and they can be considered as an extension of the many conventional mapping techniques. In addition to many considerations on their biological foundations and their really wide spectrum of applications, constructing appropriate ANNs can be seen as a really hard problem. A distinguished task in building ANNs is the tuning of a set of parameters known as *weights*. This will be the main focus of the present book. The trained ANNs can be later used in classification (or recognition) problems, where the ANN outputs represent categories, or in prediction (approximation) problems, where the outputs represent continuous variables.

In the process of training the ANN (*supervised learning*), the problem is to find the values of the weights that minimize the error across a set of input/output pairs (patterns) called the training set. In a first stage, the training is an unconstrained nonlinear optimization problem, where the decision variables are the weights and the objective is to reduce the training error. However, the main goal in the design and training of ANNs is to obtain a model which makes good predictions for new inputs (which is termed as *generalization*). Therefore the trained ANN must capture the systematic aspects of the training data rather than their specific details. Hence, as it has been well documented, the optimization problem involved in the training/generalization process is of an extreme hardness.

Metaheuristics provide a means for solving complex optimization problems to obtain acceptable solutions or even global optima. These methods are designed to search for such global optima in complex problems where other mechanisms fail because: the problem is ill-defined, or has a

very large dimensionality, or a high interaction between variables exists, or require unaffordable computational efforts for exact methods. Experimental testing of metaheuristics show that the search strategies embedded in such procedures are capable of finding solutions of high quality to hard problems in industry, business, and science within reasonable computational time. The tools and mechanisms that have emerged from the creation of metaheuristic methods have also proved to be remarkably efficient, resulting in what has been coined as hybrid methods.

Apart from some sparse efforts to bring together metaheuristic techniques to train ANNs (which include conference sessions on this field), there is no a single source of reference for such goal. In this book we aim at giving a unified approach to the work of training ANNs with modern heuristics, given the overwhelming literature proving their appropriateness to escape local optima and to solve problems in very different mathematical scenarios (two features that encapsulate important shortcomings of other well-known algorithms specifically designed to train ANNs).

The book's goal is to provide successful implementations of metaheuristic methods for neural network training. Moreover, the basic principles and fundamental ideas given in the book will allow the readers to create successful training methods on their own. Apart from Chapter 1, in which classical training methods are reviewed for the sake of the book's completeness, we have classified the chapters in three main categories. The first one is devoted to *local search based* methods, in which we include Simulated Annealing, Tabu Search, and Variable Neighborhood Search. The second part of the book presents the most effective *population based* methods, such as Estimation Distribution algorithms, Scatter Search, and Genetic Algorithms. Finally, the third part includes other advanced techniques, such as Ant Colony Optimization, Co-evolutionary methods, GRASP, and Memetic algorithms. All these methods have been shown to work out high quality solutions in a wide range of hard optimization problems, while in this book we restrict our attention to their application to the ANN training problem.

This book is engineered to provide the reader with a broad coverage of the concepts, methods, and tools of this important area of ANNs within the realm of continuous optimization. In fact, many applications dealing with continuous spaces could profit from the advances described in it. The chapters can be addressed separately depending on the reader's necessities. It would be of interest to researchers and practitioners not only in neural networks but also in management science, economics, and engineering in general. Besides, it can be used as a textbook in a master course, a doctoral seminar, or as a reference book for computer science in areas such as enterprise resource planning and supply chain management.

Chapter 1

CLASSICAL TRAINING METHODS

Emilio Soria[1], José David Martín[1] and Paulo J. G. Lisboa[2]
[1] *Grupo de Procesado Digital de Señales, Dpt. Enginyeria Electrònica, Escola Tècnica Superior d'Enginyeria, Universitat de València, Spain.*
[2] *The Statistics and Neural Computation Research Group, School of Computing and Mathematical Sciences, Liverpool John Moores University, United Kingdom.*

Abstract: This chapter reviews classical training methods for multilayer neural networks. These methods are widely used for classification and function modelling tasks. Nevertheless, they show a number of flaws or drawbacks that should be addressed in the development of such systems. They work by searching the minimum of an error function which defines the optimal behaviour of the neural network. Different standard problems are used to show the capabilities of these models; in particular, we have benchmarked the algorithms in a non-linear classification problem and in three function modelling problems.

Key words: Multilayer perceptron; delta rule; cost function.

1. INTRODUCTION

There are two main approaches to describe Artificial Neural Networks (ANNs). Some authors describe ANNs as biological models that can be applied to engineering problems (Arbib, 2003). However, other authors consider ANNs as mathematical models related to statistical models, either linear or non-linear (Bishop, 1995; Ripley, 1996; Duda, et al., 2001). These two approaches have coexisted in the theory of

neural models from the very beginning, so that advances in this theory have come from both approaches. Biological models have provided the inspiration for the development of new artificial neural models while mathematical and statistical frameworks have consolidated their practical value. This chapter is focused on the mathematical approach of artificial neural models. In this approach, an artificial neural system can be described as it is shown in Figure 1-1.

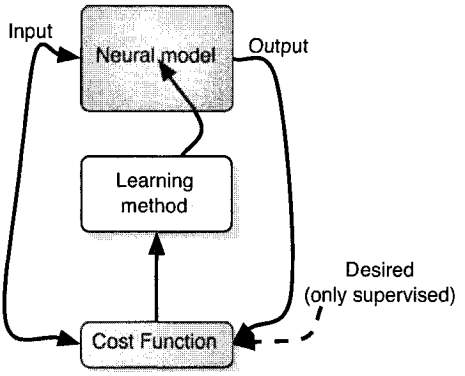

Figure 1-1. Schematic of a neural model.

Figure 1-1 shows a neural model. Three main parts can be observed:

- *Neural model.* It is the structure used to solve a certain problem. This model can be either linear or non-linear, it can have one or more than one outputs, it can consist of a combination (linear, non-linear, hierarchical, etc.) of simple neural models, etc. The performance of this model depends on several parameters which determine the complexity of the neural model.
- *Cost function.* It provides an evaluation about the quality of the solution obtained by the neural model. If an external signal to carry out this evaluation is available, then the neural model is called supervised model. If an external signal is not avaliable, then the neural model is called unsupervised model (Haykin, 1999).
- *Learning method.* The task of the learning algorithm is to obtain those parameters of the neural models that provide a solution to the tackled problem. If this solution does not exist, then the task of the learning algorithm is to find an optimal solution according to the criterion set by the cost function.

Summarizing, there are three basic elements in an artificial neural model. Since all these elements can change independently to the others, there is a huge amount of models/cost functions/ learning algorithms (Arbib, 2003). In

this chapter, we focus on the most widely used neural model, the Multilayer Perceptron (MLP) trained by supervised algorithms (in the case of unsupervised training, the resulting model is the so-called *Nonlinear Principal Component Analysis, NPCA*). We also analyse the most common cost functions associated with MLPs. For a further analysis of the most important neural models, supervised and unsupervised, the reader is encouraged to consult the excellent text (Haykin, 1999).

2. MULTILAYER PERCEPTRON

In this Section, the Multilayer Perceptron (MLP) is described. It is probably the most widely used neural network due to its interesting characteristics: universal function approximator and non-linear classifier. The MLP has shown excellent results in many different applications. In Subsection 2.1, the elementary units which form the MLP, the so-called neurons, are presented. Next subsections are devoted to explain how it works; in particular, in Subsection 2.2, the MLP architecture is analysed, in Subsections 2.3 and 2.4., the cost function, i.e., the error function that must be minimised is described, and in Section 2.5., a Bayesian approach to MLP learning is presented. Section 3 analyses the learning algorithms used to carry out the cost function minimisation.

2.1 Neurons

An MLP is a neural network made up by multiple, similar, non-linear processing units, termed neurons by analogy to the integrate-and-fire action of neural cells in biological nervous systems. Each neuron carries out a many-to-one mapping from its inputs to a single scalar output. A general schematic of a neuron model is shown in Figure 1-2 (Hecht-Nielsen, 1989).

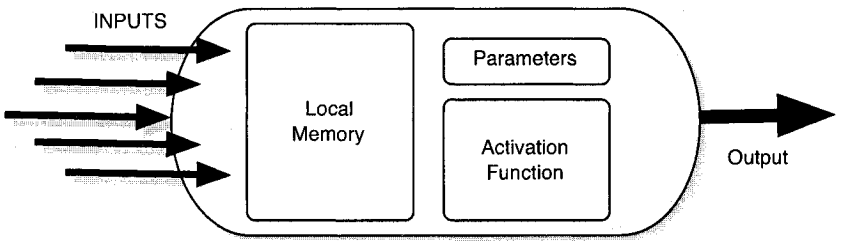

Figure 1-2. General schematic of a neuron.

This neuron model is formed by the following components:

- *Parameters.* These parameters are usually known as *synaptic weights.* They are used in combination with the inputs according to a certain function. Considering input vectors and coefficient vectors, the most used functions are the scalar product and the Euclidean distance.
- *Activation function.* This function is very important since the neural network capabilities to solve complex problems stem from it. It is a non-linear function whose argument is the aforementioned combination between synaptic weights and input vector. The most used functions are the *sign function,* the *hyperbolic tangent,* the *sigmoidal function* (hyperbolic tangent modified by restricting the range of values between 0 and 1), and the *Gaussian function.*
- *Local memory.* This component is used when neural networks are designed to time series modelling. Local memories can store either previous inputs or previous outputs, so that the neuron "remembers" previous behaviours. There are many different possibilities to implement this kind of memories (Weigend & Gershenfeld, 1993).

In the remainder of the chapter, we will focus on the neuron model shown in Figure 1-3:

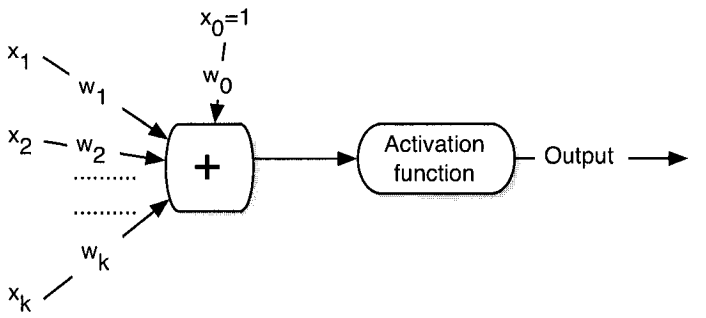

Figure 1-3. Schematic of the most used neuron model.

This model is formed by the following components:

- *Sum function.* This component computes the sum of the product between input vectors and synaptic weights. Let be $\boldsymbol{w}=[w_0, w_1, \ldots, w_k]$ the synaptic weight vector and $\boldsymbol{x}=[1, x_1, \ldots, x_k]$ the input vector, this sum can be given by the scalar product of both vectors. From a biological point of view, this function represents the action of the inputs produced in the axons of the biological neurons (Arbib, 2003). The coefficient w_0 plays a

relevant role in the processing of the neuron, and it is called threshold or bias.

- *Activation function.* Among all the possible choices for this activation function, the following ones should be emphasised:

  Sign function. It was the first used activation function in an artificial neural model (Arbib, 2003). This function proposes a crisp separation of the input data, so that they are classified as ±1. This function is defined as follows: $f(x) = \begin{cases} -1 \text{ si } x < 0 \\ 1 \text{ si } x \geq 0 \end{cases}$.

  Sigmoidal function. Since many learning algorithms need a differentiable activation function, and the sign function can not be differentiated in the origin, the sigmoidal function was proposed. This function is given by the following expression: $f(x) = \frac{a}{1+e^{-b \cdot x}}$ where *a* stands for the amplitude of the function and *b* is the slope in the origin; the higher the value of *b*, the closer the sigmoidal function to the sign function (but with the outputs ranging between *0* and *a*). The bipolar version of the sigmoidal function (outputs ranging between *–a* and *+a*) is given by the hyperbolic tangent: $f(x) = a \cdot \frac{1-e^{-b \cdot x}}{1+e^{-b \cdot x}}$.

  Gaussian function. An alternative activation function is used in a different type of neural networks, known as *Radial Basis Function (RBF)*. Its expression is given by: $f(x) = K_1 \cdot e^{-K_2 \cdot (x-c)^2}$, being K_1, K_2 and *c*, values which determine the amplitude of the Gaussian function, its width and its centre, respectively. The main characteristic of this activation function, which makes different from others, is its locality. The function tends to zero from a certain value *x* on; this way, only in a reduced range of input values, the output of the activation function is considerably different from zero. Gaussian functions have their centre and width values typically heuristically determined from the data.

2.2 Architecture

The architecture of a neural model gives information about how neurons are arranged in the neural model. In the case of MLP, as shown in Figure 1-

4, neurons are arranged in layers: one input layer, one output layer, and one or more than one hidden layers (Arbib, 2003). This arrangement in layers is due to the fact that the outputs of the neurons of a certain layer are used as inputs to the neurons of next layer (network without feedback) and/or to neurons of previous layers (networks with feedback or recurrent networks).

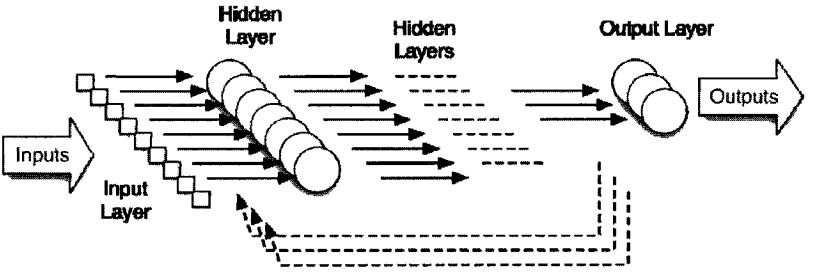

Figure 1-4. Multilayer perceptron. Dotted arrows stand for recurrent neural systems.

The number of neurons in both input and output layers is given by the problem to solve, whereas the number of hidden layers, and the neurons in each layer should be chosen by the designer of the neural network. It is shown that an MLP with one hidden layer is a universal approximator of continuous functions; if the function to model is not continuous, then two hidden layers become necessary (Cybenko, 1988). Although there are several rules to assess the approximately optimal number of hidden neurons, there is no precise methodology to determine this number exactly. Therefore, trial-and-error procedures are usually carried out to estimate the number of hidden neurons, often relying on cross-validation[1] or other evaluation procedures.

2.3 Cost Function

A neural system should be designed to present a desired behaviour, hence, it is necessary to define what is desired, being used a cost function for this task. Two points of view can be considered for cost functions: operational and probabilistic.

- *Operational.* This point of view consists of implementing the final goal of the neural system in the cost function. This is the case of supervised

[1] The cross-validation procedure is described in Section 3.1.

systems; in these systems, the model is designed to provide an output as similar as possible to a certain desired signal (Figure 1-1). An error signal is defined from the difference between the neural model output and the desired values. The goal of the neural model is to obtain an error signal equal to zero; therefore, if an error function is defined provided that its minimum corresponds to a zero error, then the goal is transformed in a problem of function minimisation (Hassoun, 1995; Bishop, 1995).

- *Probabilistic*. Two kind of problems can be solved by using this point of view: function modelling and pattern classification (Haykin, 1999). In the former case, the probability distribution of the desired signals conditioned by the input variables to the neural model must be modelled. In a pattern classification problem, the aim is to model the conditioned probabilities of every class, also by the input variables to the neural model (Bishop, 1995; Ripley, 1996).

The two points of view can be related using the *maximum likelihood principle*, so that assuming different probabilistic models for the error between the network output and the target values, different cost functions are obtained (Bishop, 1995). For instance, if the error is assumed to follow a Gaussian distribution, then the most used cost function is obtained, the mean-square error (L_2 in Table 1-1). The most used cost functions are shown in Table 1-1 (Cichocki & Amari, 2002; Hassoun, 1995; Bishop 1995).

Table 1-1. Most used cost functions. The parameter β controls the error ranges in which a certain cost function is used; this parameter is used in those cases in which different subfunctions are defined for a certain cost function. In the case of the logistic cost function, α controls the robustness to outliers. In the remainder of the chapter, the desired signal will be denoted by d, the output of the neural model by o and the error signal by e, being $e=d-o$

Name	Cost Function
L_2	e^2
L_1	$\lvert e \rvert$
L_p	$\frac{1}{p} \cdot \lvert e \rvert^p$

Name	Cost Function
Entropic	$d \cdot \log\left(\frac{d}{o}\right) + (1-d) \cdot \log\left(\frac{1-d}{1-o}\right)$ $0 \le o \le 1$
Logistic	$\frac{1}{\alpha} \cdot \log(\cosh(\alpha \cdot e))$
Huber	$\begin{cases} 0.5 \cdot e^2 & for \lvert e \rvert \le \beta \\ \beta \cdot \lvert e \rvert - 0.5 \cdot \beta^2 & otherwise \end{cases}$
Talvar	$\begin{cases} 0.5 \cdot e^2 & for \lvert e \rvert \le \beta \\ 0.5 \cdot \beta^2 & otherwise \end{cases}$

2.4 Relevance of an Adequate Cost Function

The MLP architecture may be configured in two ways, namely for regression, and classification. In the first instance, the hidden layer is nonlinear but the output layer is normally linear and the appropriate cost function is the sum of square errors, reflecting the assumption that the noise in that data is homoscedatic (i.e. uniform across the range of input values) and normally distributed about the origin (Ripley, 1996; Bishop, 1995).

However, if the model is intended for classification, then the output layer becomes non-linear. For binary classification this will be a sigmoid, while for multi-class assignments it will be a softmax, which is a multivariate extension of the sigmoid function. In either case, the appropriate cost function to use is entropic (Ripley, 1996; Bishop, 1995). This can be seen by means of the following example.

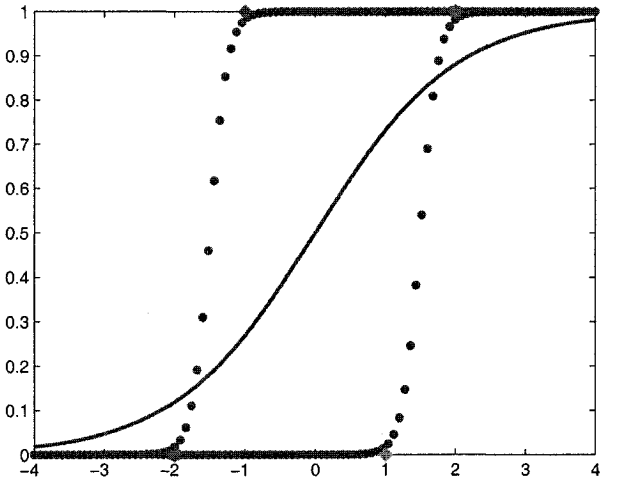

Figure 1-5. The solid line represents the optimal response function for binary classification in the presence of noise, while the dotted lines are two local minima that arise if the incorrect cost-function is used. These local minima may have less cost than the correct fit to the data, leading to inconsistency and inaccurate results.

Consider a model of a single neuron with a single input node directly linked to a single output node. Since, this model has only two parameters, its weight and bias terms, the cost function can be represented for the complete parameter space as a 3D plot. Now, consider the typical problem of classification with noisy data, shown in Figure 1-5. In this case there are 'true' class data, on-class to the right of the origin and off-class to the left, but there are also noisy data that overlap into the wrong side of the decision boundary, as is often the case in practice.

It is now a simple matter to plot the cost function for a sum-of squares error and for a log-likelihood error, as a function of the model parameters. These plots are in Figure 1-6a and 1-6b. It is straightforward to show that as the ratio of true class data to noisy data increases, the local minima along the two ravines spreading diagonally across the plot become closer in value to the correct minimum at the centre of the graph, while retaining a lower cost (i.e. being more optimal) than the true minimum. These ravines correspond to the functions shown as dotted lines in Figure 1-5. Fig. 1-6 (a) also shows the presence of plateaus even in this simple 1-D problem, a situation that can become much more acute in higher dimensions.

This example clearly shows that choosing an appropriate cost function is not an option, but a necessity, in order to obtain reproducible results that generalise well to out-of-sample data.

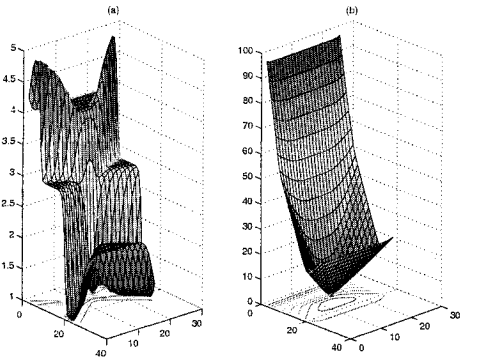

Figure 1-6. Complete maps of the cost function for the data in Fig. 1-5, using a sum of squares error (a) and a log-likelihood error (b). Two quite different surfaces are obtained, one with local minima and the other without.

2.5 Bayesian Approach to MLP Learning

MLP learning can be analysed from a Bayesian pint of view (MacKay, 2003; Bishop 1995), in which the learning process can be seen as an inference process. Since MLP learning is based on mapping an input space into an output space, Bayes' Theorem can be formulated, as follows

$$P(w \mid D) = \frac{P(D \mid w) \cdot P(w)}{P(D)}$$

where D is the desired data set and w is the set of network parameters (synaptic weights). It is necessary to assume a distribution for the neural model parameters; usually, the following expression is used (MacKay, 2003):

$$P(w)=\frac{e^{(-\alpha \cdot E_w)}}{Z_W(\alpha)}$$

E_w is an increasing function of the synaptic-weight values and α is a parameter known as *hyperparameter*. Function $Z_w(\alpha)$ is only used for normalisation purposes. If $P(D|w)$ is a similar distribution to that shown by synaptic weights in Bayes' expression, then:

$$P(D \mid w)=\frac{e^{(-\beta \cdot E_D)}}{Z_D(\beta)}$$

And replacing in the first expression:

$$P(w \mid D)=\frac{e^{(-\alpha \cdot E_W-\beta \cdot E_D)}}{Z_D(\beta)\cdot Z_W(\alpha)}=\frac{e^{(-M(w))}}{Z_M}$$

Taking logarithms and maximising this expression, the conclusion is that the Bayesian objective becomes the minimisation of a cost function, which is given by E_D, together with a regularisation term of the network parameters (E_W). The Bayesian approach has several advantages over approaches based on cost functions (MacKay, 2003; Bishop, 1995).

3. CLASSICAL LEARNING ALGORITHMS

A learning algorithm is a procedure which adapts the parameters and/or architecture of the neural model in order to solve a certain problem. The algorithms to adapt the architecture of the MLP are known as pruning methods or growing methods, depending on the strategy to carry out this adaptation (Haykin, 1999). Pruning methods have been more widely used than growing methods.

Learning algorithms to adapt the parameters of the neural model tend to be based on the minimisation of the cost function chosen to solve the posed problem. First neural models were formed by only one neuron and they used L_2 as cost function, so that minimum was obtained by solving a system of linear equations (Haykin, 1996). However, the practical application of these models involved solving this system of equations in every instant of processing; taking into account the huge number of unknown quantities and the technological state of those years (1950-1960), other approaches were researched in order to solve these systems of equations.

The most important characteristics that a learning algorithm must show, are the following (Haykin, 1999):

- *Efficiency*. Ability to solve the problem with the minimum computacional burden.
- *Robustness*. The algorithm should present immunity to undesired noise.
- *Independence on the initial conditions*. The algorithm should show similar solutions independently of the values used to initialise the algorithm.
- *Generalisation capabilities*. The algorithm should provide the adequate outputs when inputs different to the training data set, are used.
- *Scalability with the size and complexity of the data*. The algorithm should have a computational burden that does not strongly depend on the dimensionality and size of the problem to solve.

3.1 Backpropagation Algorithm (BP)

The BP was the first algorithm used for MLP adaptation (Werbos, 1974; LeCun, 1985; Rumelhart, 1986), and it is still the most known and used nowadays. The goal pursued by the algorithm is to obtain the minimum of the cost function (Section 2.3 of this chapter), which is denoted as J. The actual solution of J is computationally unfeasible in many applications. Moreover, the solution of J in a certain instant may be uninteresting in many cases since the properties of the input signals to the neural model may be time-dependent and the neural models must adapt to these changes. An iterative solution is proposed:

$$w_{n+1} = w_n + \Delta w_n \tag{1}$$

where w are the parameters of the neural model and the subscript stands for the time instant. Many algorithms are designed to accomplish Eq. (1); within these algorithms, there are two possibilities for learning (Haykin, 1999):

- *On-Line.* The MLP is fed during all the training process with the input of every pattern and the corresponding desired output. Then, the error is measured and the synaptic weights are adapted depending on this error by using the chosen algorithm.
- *Batch.* In this case, the error between the network output and the desired values is computed for all the patterns. Then, the model parameters are adapted depending on the average error for all the patterns. The computation of all the outputs of the neural model for all the available input patterns is known as *epoch*.

One of the keys of Eq. (1) is to find out the optimal increase/decrease in the parameters that enables to find the minimum of the function. A low computational burden approach is based on a geometrical analysis of the problem. This analysis is based on finding the direction of the minimum from a certain point w_n in the parameter space of the neural model. Since the function gradient points to the direction of the function maximum, the approach will be based on finding the function minimum by moving the synaptic weights in the opposite gradient direction (Bishop, 1995; Luenberger, 1984):

$$w_{n+1} = w_n - \alpha \cdot \nabla_{w_n} J \tag{2}$$

where α is the so-called learning rate or adaptation constant.

The BP algorithm, based on Eq. (2), is a gradient-descent algorithm which backpropagates the error signals from the output layer to the input layer, thus optimising the values of the synaptic weights through an iterative process. Therefore, two stages can be considered:

- ***Feed-forward propagation***: The output of the neural network is obtained, and then, the error is computed by comparing this output with the desired signal.
- ***Backpropagation***: Depending on the error between the network output and the desired values, the algorithm optimises the values of the synaptic weights by means of error backpropagation from the output layer to the input layer, and through the hidden layers.

A schematic of a general connection between two neurons (i,j) is shown in Figure 1-7.

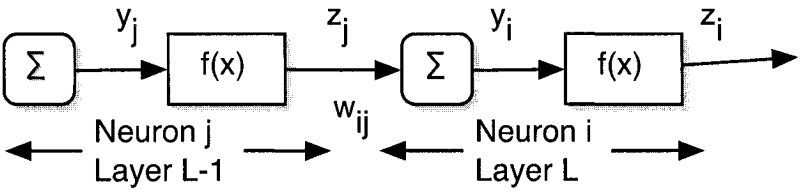

Figure 1-7. Schematic of a connection between two neurons.

The gradient of the cost function must be computed with respect to the parameters of the neural network, i.e., the synaptic weights. The following notation will be used: $y_s^{L-1}; z_s^{L-1}$ stands for the output of the sum function and the activation function, respectively (neuron s and layer L-1). Therefore, taking all these facts into account:

$$\frac{\partial J}{\partial w_{ij}^L} = \frac{\partial J}{\partial y_i^L} \cdot \frac{\partial y_i^L}{\partial w_{ij}^L} = \delta_i^L \cdot \frac{\partial}{\partial w_{ij}^L} \sum_s w_{is}^L \cdot z_s^{L-1} = \delta_i^L \cdot z_j^{L-1}$$
$$\text{where } \delta_i^L = \frac{\partial J}{\partial y_i^L} \tag{3}$$

δ is known as local gradient, an can be obtained as follows:

$$\delta_i^L = \frac{\partial J}{\partial y_i^L} = \sum_m \frac{\partial J}{\partial y_m^{L+1}} \cdot \frac{\partial y_m^{L+1}}{\partial y_i^L} = \sum_m \delta_m^{L+1} \cdot \frac{\partial y_m^{L+1}}{\partial y_i^L} \Rightarrow$$
$$\delta_i^L = \sum_m \delta_m^{L+1} \cdot \frac{\partial}{\partial y_i^L} \cdot \sum_j w_{mj}^{L+1} \cdot z_j^L \Rightarrow \delta_i^L = \frac{dz_i^L}{dy_i^L} \cdot \sum_m \delta_m^{L+1} \cdot w_{mi}^{L+1} \tag{4}$$

Therefore, an iterative procedure is used to compute local gradients of first layers from local gradients of last layers. The output layer gradient is given by:

$$\delta_k = \frac{\partial J}{\partial e_k} \cdot \frac{de_k}{dz_k} \cdot \frac{dz_k}{dy_k} = -\frac{\partial J}{\partial e_k} \cdot \frac{dz_k}{dy_k} \tag{5}$$

Output gradient depends on the error and activation functions. A very attractive advantage of using a sigmoidal/hyperbolic tangent activation

function stems from the fact that their derivatives can be expressed by using the own activation functions (Haykin, 1999).

In spite of the advantages offered by the BP algorithm, it also shows a number of drawbacks that should be known (Haykin, 1999; Arbib, 2003; Bishop 1995; Orr & Müller, 1998):

- *Neuron saturation.* Since the derivative of the activation function appears in the weight update, and this derivative equals zero for the most used activation functions (hyperbolic tangent and sigmoidal) in the function extremes, weights are not updated in these zones although the modelling error is different from zero.
- *Weight initialisation.* Weight initialisation is basic in order to achieve a good modelling. Since the learning algorithm is based on finding the minimum of the error function that is closest to a certain initial point, this minimum may be a local minimum, not a global one, and therefore this initial point is fundamental for ensuring that the network finally achieves a global minimum. Weight initialisation affects algorithm behaviour in three main factors a) *Convergence speed;* the convergence speed depends not only on the learning rate, but also on the initial distance to the minimum. b) *Minimum achieved*; the algorithm finds the closest minimum to the initial point, which may be a local minimum. c) Neuron saturation; large values of the weights can involve a neuron saturation.
- *Plateaus.* Weight update is proportional to the derivative of the error function. This derivative equals zero or a very low value on the flat parts of the error function. Therefore, weights are hardly updated.
- *Choice of the learning rate.* Too high values may involve instabilities while too low values may make the converge speed very slow. There are many algorithms which propose strategies to find an optimal learning rate. They tend to be based on the following claim: *"the value of the learning rate should be high far away from the minimum and should be low near the minimum".*
- *Early stopping.* There are different criteria to carry out the learning stopping, such as, to fix a number of epochs in advance, an error threshold, to find plateaus in the cost function, etcetera. Data are usually split into two sets: a training set and a generalisation set. The former is used to train the network, whereas the latter is used to check the behaviour of the network with patterns different from those used in the training process. The goal is to obtain a model with good generalisation properties. The generalisation error tends to decrease as the learning process is progressing until a certain epoch, in which the generalisation error starts to increase because the network is overfitting the patterns of the training data set. When this change in the tendency of the

generalisation error is observed, the learning must be stopped. This procedure is known as cross-validation. There are also different criteria to decide the rate of patterns that should be assigned to each data set: 66% of the patterns to the training set, a small percentage if there is a large amount of data available, and there are also dynamic processes to carry out the selection of these sets (Haykin, 1999; Bishop, 1995).

- *Architecture choice.* The number of hidden neurons and layers is a difficult choice, being determined in many cases by using trial-and-error procedures. Nonetheless, there also pruning and growing methods that are used to find optimal structures (Reed, 1993; Reed & Marks, 1999).

3.2 Variants of the BP Algorithm

Some variants have been proposed in order to overcome the different problems observed in BP algorithm. In this section, we will focus on four of the most classical variants, namely, *Momentum term, Silva-Almeida, Delta-Bar-Delta* and *Rprop*.

3.2.1 Momentum Term

This variant (backpropagation with momentum, BPM) is very similar to the classical backpropagation algorithm. The difference is an additional term, which provides information about the weight change in the previous epoch. Therefore, weight update is given by (Haykin, 1999):

$$\Delta w_{n+1} = -\alpha \cdot (\nabla J)_n + \mu \cdot \Delta w_n \tag{6}$$

This new term controls the speed of convergence, speeding up the process when far from the minimum, and slowing it down when close to the minimum. The momentum coefficient μ gives more or less importance to the momentum term. This algorithm shows instabilities near the minimum.

3.2.1.1 Silva-Almeida

This variant adapts the value of the learning rate depending on the distance to the minimum. This distance is evaluated through two consecutive signs of the gradient of the error function. These gradients should show the same sign far away from the minimum (the algorithm is approaching the minimum in the same direction) whereas the signs should be different near the minimum since the algorithm should be oscillating around the minimum.

Weight update is identical to that carried out by the classical *backpropagation* but taking into account that learning rate is determined by (Silva & Almeida, 1990):

$$\alpha(n)=\begin{cases}\alpha(n-1)\cdot u \Leftrightarrow (\nabla J)_n \cdot (\nabla J)_{n-1} > 0\\ \alpha(n-1)\cdot d \Leftrightarrow (\nabla J)_n \cdot (\nabla J)_{n-1} < 0\end{cases} \tag{7}$$

being $d<1$ and $u>1$.

Moreover, this algorithm incorporates a sort of "*pocket technique*" since if an increase in the network error is observed, then the previous weights are retrieved.

3.2.2 Delta-Bar-Delta

This variant is similar to Silva-Almeida, since the learning rate is also adapted. In this case, the adaptation is given by (Jacobs, 1988):

$$\alpha(n)=\begin{cases}\alpha(n-1)+u \Leftrightarrow (\nabla J)_n \cdot (\delta)_{n-1} > 0\\ \alpha(n-1)\cdot d \Leftrightarrow (\nabla J)_n \cdot (\delta)_{n-1} < 0\end{cases} \tag{8}$$
$$(\delta)_{n-1} = (1-\theta)\cdot(\nabla J)_{n-1} + \theta \cdot (\delta)_{n-2}$$

provided that $0<\theta<1$.

Equation (8) shows two main differences with regard to *Silva-Almeida* method:

- The increase in the learning rate to speed up the convergence is not an exponential increase, but a linear one. Therefore, it is less likely the presence of instabilities in the algorithm due to an excessively high value of the learning rate.
- The increase or decrease of the learning rate does not depend only on two consecutive gradients, but it is carried out by comparing the gradient and a weighted average of previous gradients given by the parameter δ.

A variation in weight update is proposed in (Minai & Williams, 1990):

1. An exponentially decreasing function of δ_{ij}^n is used instead of increasing the learning rate by using a constant factor.
2. A momentum term is added. It is updated likewise the learning rate.

3. Maximum and minimum values are imposed for the coefficients.
4. The "pocket technique" is used in order to ensure that the minimum error is obtained.

3.2.2.1 Rprop (Resilient Backpropagation)

This algorithm also proposes an adaptation of the learning rate. Moreover, weight update is somewhat different to the other variants, and it is given by (Riedmiller & Braun, 1993):

$$\Delta w_{n+1} = -\alpha \cdot sign(\nabla J)_n \tag{9}$$

Using the sign of the gradient involves less computational burden. The learning rate is adapted as follows:

$$\alpha(n) = \begin{cases} \min[\alpha(n) \cdot u, \alpha_{\max}] \Leftrightarrow (\nabla J)_n \cdot (\nabla J)_{n-1} > 0 \\ \max[\alpha(n) + d, \alpha_{\min}] \Leftrightarrow (\nabla J)_n \cdot (\nabla J)_{n-1} < 0 \end{cases} \tag{10}$$

being $u>1$ and $d<1$.

Therefore, the learning rate can show two different values, depending on the sign of the two last gradients. The aim is to have a low value of the learning rate near the minimum and a higher value far away from the minimum, thus controlling the convergence speed. The possible values of the learning rate are limited in order to avoid either an excessively high value which can lead to instabilities or a too low value which can lead to a very slow convergence speed.

Another first order algorithm is that proposed in (Chan & Fallside, 1987). This algorithm analyses the relationship between the directions defined by the gradient of the error function and the previous weight increase. In particular, it is used the following parameter:

$$\cos(\theta(n)) = -\frac{(\nabla J)_n \cdot \Delta w_{n-1}}{\|(\nabla J)_n\| \cdot \|\Delta w_{n-1}\|} \tag{11}$$

where $\|\cdot\|$ stands for the norm. This algorithm helps in overcoming the irregular parts of the error surface. This is because if there is not an appreciable change in the direction between the previous weight increase and the search direction (the gradient of the error function with opposite sign), then it means that the search process stays in a stable situation.

This Section has shown a small illustration of the most important variants of the BP algorithm. There are many other variants. A deeper review and comparison among these algorithms can be found in (Schiffmann et al., 1994), (Moreira & Fiesler, 1995).

3.3 Other Algorithms

In spite of BP algorithm and its variants are usually chosen to train MLPs, more complex algorithms have also been proposed. These algorithms tend to show a higher computational burden and faster convergence speed than BP.

3.3.1 Conjugate Gradient Algorithms

These algorithms show a convergence speed faster than that obtained by BP, in exchange for a small increase in the computational burden (Luenberger, 1984). Weight update is given by the following expression (Bishop, 1995):

$$w_{n+1} = w_n + \alpha(n) \cdot d(n) \tag{12}$$

where $\alpha(n)$ is the learning rate and $d(n)$ stands for the search direction of the minimum. This search direction is a linear combination of the gradient of the function (with opposite sign) and the previous search direction. Therefore, $d(n+1)$ is given by:

$$\begin{aligned} &d(n+1) = -g(n+1) + \beta(n) \cdot d(n) \\ &\textit{where } \mathrm{d}(0) = -g(0) \end{aligned} \tag{13}$$

$g(k)$ is the gradient of the cost function at instant k. These gradients are computed using the procedure followed in the BP algorithm. It should be pointed out that parameter $\beta(n)$ is used to define the relationship between the two directions in weight update. There are several variants of this algorithm depending on the value of this parameter (Bishop, 1995; Luenberger, 1984). The most used variants are the following (superscript t means transposition):

Fletcher-Reeves (FR). $$\beta(n) = \frac{g^t(n) \cdot g(n)}{g^t(n-1) \cdot g(n-1)} \tag{14}$$

Polak-Ribiere (PR). $$\beta(n)=\frac{g^t(n)\cdot[g(n)-g(n-1)]}{g^t(n-1)\cdot g(n-1)} \quad (15)$$

All these algorithms obtain the learning rate $\alpha(n)$ dynamically, using a line search procedure (Luenberger, 1984).

3.3.2 BFGS Algorithm

The goal of the learning of a neural network can be stated as a function minimisation problem. One of the most known methods of function minimisation is the Newton's method, which is faster than the other methods previously depicted. Weight update is given by the following expression:

$$w_{n+1}=w_n-[H(n)]^{-1}\cdot g(n) \quad (16)$$

The main problem of this algorithm lies in the requirement of knowing the *Hessian matrix H(n)*. This matrix contains the second derivatives of the cost function with respect to the parameters of the problem, the synaptic weights in this case. An approximation can be calculated, with an additional problem, which is the requirement of the approximation to be positive definite in order to guarantee algorithm convergence (Luenberger, 1984; Press et al., 1992). Usually, far away from the minimum and if the cost function is not quadratic, the matrix is not positive definite.

There is a kind of algorithms based on the Newton's method, the so-called quasi-Newton methods. These methods estimate the inverse Hessian matrix, forcing this matrix to be positive definite in every step (Press et al., 1992). Weight update is stated as follows:

$$w_{n+1}=w_n-\alpha(n)\cdot M(n)\cdot g(n) \quad (17)$$

M(n) is an estimation of the inverse Hessian matrix, which is updated every iteration. The most widely used quasi-Newton method is BFGS, acronym of the names of the authors who proposed the method (Broyden-Fletcher-Goldfarb-Shanno). This algorithm carries out the following processing (Luenberger 1984):

1) Algorithm initialisation; *M(0)* is considered to be any positive definite matrix, the identity matrix, for instance.
2) Search direction is calculated as follows:

$$d(n) = -M(n) \cdot g(n) \tag{18}$$

3) $\alpha(n)$ is optimised within the cost function $J[w+\alpha(n) \cdot d(n)]$ by means of a line search procedure (Luenberger, 1984; Press et al., 1992).
4) The following expressions are calculated:

$$\begin{aligned} p(n) &= \alpha(n) \cdot d(n) \\ q(n) &= g(n+1) - g(n) \\ M(n+1) &= M(n) + \frac{p(n) \cdot [p(n)]^t}{[p(n)]^t \cdot p(n)} - \frac{M(n) \cdot q(n) \cdot [q(n)]^t \cdot M(n)}{[q(n)]^t \cdot M(n) \cdot q(n)} \end{aligned} \tag{19}$$

5) Go to next iteration $n=n+1$, and go to step 2.

3.3.3 Levenberg-Marquardt

The Levenberg-Marquardt (LM) algorithm was proposed to be used with the quadratic cost function (Bishop, 1995):

$$J = \frac{1}{2} \cdot \sum_k (e_k)^2 \tag{20}$$

being e_k the error for pattern k and appearing the factor ½ for the sake of simplicity (after differentiation, all the constant terms are cancelled). If the vector of components e_k is denoted by $\boldsymbol{e}$, and if small perturbations of the synaptic weights are considered (Bishop, 1995):

$$\mathbf{e}\big|_{new} = \mathbf{e}\big|_{old} + L \cdot \left[w\big|_{old} - w\big|_{new} \right] \tag{21}$$

where L is a the following matrix

$$L\big|_{st} = \frac{\partial e_s}{\partial w_t} \tag{22}$$

The cost function shown in Eq. (20) can be written as:

$$J = \frac{1}{2} \cdot \sum (e_k)^2 = \frac{1}{2} \cdot \|\mathbf{e}\|^2 = \frac{1}{2} \cdot \left\| \mathbf{e}|_{old} + L \cdot \left[w|_{old} - w|_{new} \right] \right\|^2 \quad (23)$$

The minimisation of this function with respect to the new weights leads to the following weight update:

$$w|_{new} = w|_{old} - \left(L^t \cdot L \right)^{-1} \cdot L^t \cdot \mathbf{e}|_{old} \quad (24)$$

The matrix L is easy to be obtained since it only needs the first derivatives of the cost function. This procedure depends on the requirement of small changes of the synaptic weights in Eq. (21), so that if this condition is not true, then the algorithm can become instable. LM algorithm solves this problem forcing the change of the weights to be small by means of a cost function:

$$J = \frac{1}{2} \cdot \left\| \mathbf{e}|_{old} + L \cdot \left[w|_{new} - w|_{old} \right] \right\|^2 + \lambda \cdot \left\| w|_{new} - w|_{old} \right\|^2 \quad (25)$$

The minimisation of Eq. (25) with respect to the new weights leads to:

$$w|_{new} = w|_{old} - \left(L^t \cdot L + \lambda \cdot I \right)^{-1} \cdot L^t \cdot \mathbf{e}|_{old} \quad (26)$$

being I the identity matrix. If λ takes a very high value, then the BP algorithm is obtained. There are iterative procedures to obtain this factor (Bishop, 1995).

The main drawback of this algorithm lies in the need of saving the inverse matrix of L. The size of this matrix is the square of the number of synaptic weights of the network. Therefore, this method is not a good choice when the number of weights is large (Nelles, 2001).

4. EXPERIMENTAL RESULTS

In this section, the algorithms presented in previous sections are benchmarked in two different kind of problems: a classification problem and three modelling problems. Algorithms are compared in terms of accuracy and convergence speed. The accuracy achieved by the algorithm is obviously important since it is a measure of the capability of the algorithm to solve the problem. Moreover, in many practical applications (channel equalisation, for

instance), the necessary time to find the solution can be almost as important as the accuracy achieved. Experiments were carried out using the hyperbolic tangent as activation function in the hidden neurons. One hundred tests were run for every experiment, using different weight initialisations; however, the same one hundred different initialisations were used in all the networks in order to obtain unbiased results. The value for the learning rate was equal to *0.5/N*, being *N* the number of hidden neurons, and the value of the momentum term was chosen as equal to *0.8*.

4.1 Classification Problems

Channel equalisation is a typical problem of application of neural networks in Communications (Qureshi, 1985; Chen et al., 1990). A general communication system is shown in Figure 1-8.

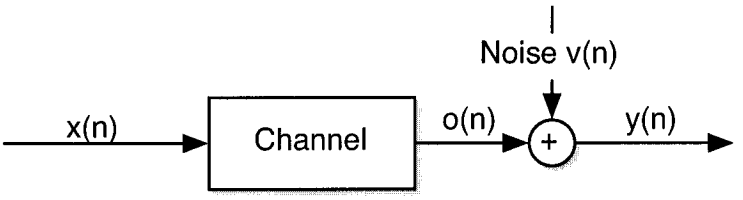

Figure 1-8. General schematic of a communication system.

Figure 1-8 shows a message *x(n)* which is sent through a communication channel. This message is modified and corrupted by the transmission channel and by the ambient noise, *v(n)*, which is modelled by a Gaussian distribution with mean 0, being used its variance to characterise the noise. The goal is to decode the emitted message from the received message, as it is shown in Figure 1-9.

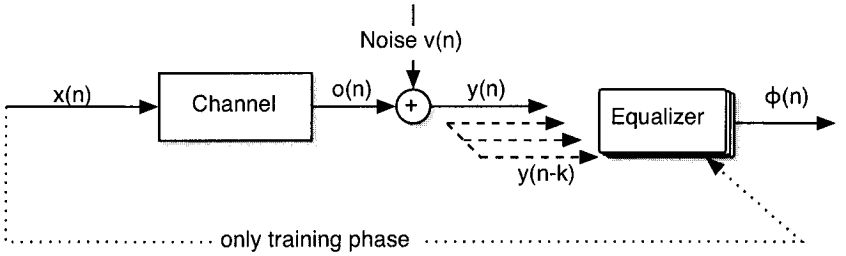

Figure 1-9. Schematic of a communication channel with equaliser.

Figure 1-9 shows how the equaliser works. During the training process, a sequence is transmitted, being this sequence known by both the emitter and the receiver (dotted line). This way, the desired signal needed by every supervised system is available. The emitted message is decoded by the equaliser, using as inputs the message received in the current instant, and also that received in previous instants, denoted as *y(n-k)* in Figure 1-9.

We chose a widely used channel, whose difference equation is the following (Gibson et al., 1991):

$$o(n) = 0.5 \cdot x(n) + x(n-1) \tag{27}$$

The message emitted in our simulations was ±1 (this signal is known in Engineering as 2-PAM), with equal probabilities. The noise variance was varied. Signal-to-Noise Ratio (SNR) was used to characterise the transmitted signal and the environmental noise (Proakis, 2001):

$$SNR = -20 \cdot \log_{10}(\sigma_v) \tag{28}$$

where σ_v is the standard deviation of noise signal *v(n)*; this SNR is measured in dB. A representation of emitted data in the space of received data is shown in Figure 1-10 for different values of SNR.

In Figure 1-10, the classification problem is not linearly separable, and its structure is similar to other standard classification problems, as those shown in (Ripley, 1996). The main advantage of using this problem lies in the easiness to change the problem conditions, and also in their actual practical application.

Convergence speed for different algorithms is compared in Tables 1-2 and 1-3. Two different architectures were considered, setting the SNR equal to 20 dB. In order to measure convergence speed, the algorithm was supposed to converge in that iteration in which, the mean square error (MSE) of the neural network was less or equal to 15% of the error of the neural network in the first epoch after its initialisation. Since it was a relative threshold, results should also be interpreted the same way, i.e., as a relative comparison of different algorithms. Tables 1-2 and 1-3 show the frequency of each relative position of the algorithms with respect to convergence speed; in case of draw (for instance, if three algorithms achieved the threshold at the same time), the best position was considered for all the algorithms (for instance, the three algorithms would be assigned to the first position).

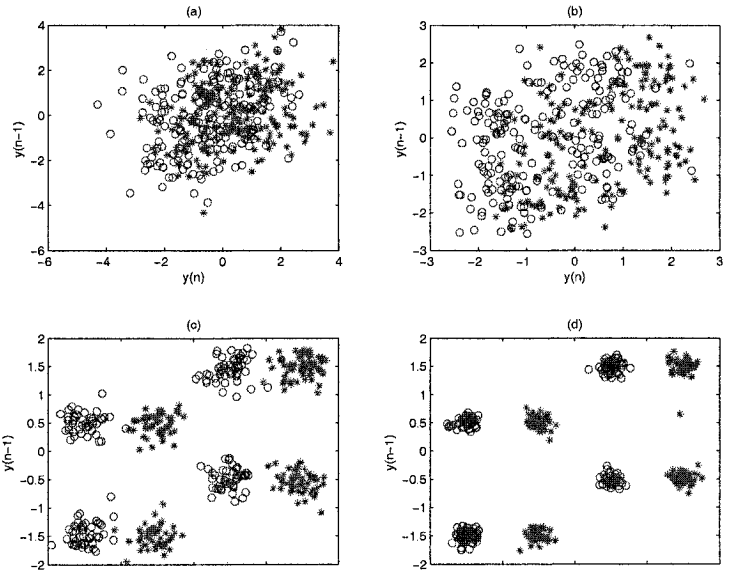

Figure 1-10. Representation of emitted data in the space of received data for different values of SNR. The transmitted symbols are '*' *x(n)=1* and 'o' *x(n)=-1*. (a) SNR=5 dB, (b) SNR=10 dB, (c) SNR=15 dB, (d) SNR=20 dB.

Table 1-2. Architecture 2×3×2×1. Frequency of relative positions of the different algorithms in terms of convergence speed. The most frequent algorithm for each relative position is highlighted in bold

	1º	2º	3º	4º	5º
BP	0	0	1	7	**92**
BPM	0	1	18	**79**	2
FR	6	27	**56**	11	0
BFGS	32	**52**	11	4	1
LM	**64**	21	12	1	2

Table 1-3. Architecture 2×10×1. Frequency of relative positions of the different algorithms in terms of convergence speed. The most frequent algorithm for each relative position is highlighted in bold

	1º	2º	3º	4º	5º
BP	0	0	0	6	**94**
BPM	2	4	23	**68**	3
FR	9	28	**41**	21	1
BFGS	31	**52**	11	4	2
LM	**66**	17	16	1	0

Tables 1-2 and 1-3 show that algorithms based on gradient descent exclusively, such as BP and BPM, were the slowest ones. In particular, the algorithm using the least information, BP, was the slowest algorithm. On the contrary, LM algorithm was the fastest algorithm for the selected architectures.

Tables 1-4 and 1-5 show a comparison of the accuracy achieved by different algorithms. This accuracy was measured by means of the so-called Bit Error Rate (BER). BER is a measure of the relationship between the transmitted bits (each symbol $x(n)$ is a bit) and the error between the network output and the desired values. Results show an average of 100 tests for different values of SNR and different architectures.

Table 1-4. Bit-Error-Rate. Architecture 2×4×1. Different values of SNR are shown in the different columns. The best results for each SNR are highlighted in bold

	5	10	15	20
BP	0.15±0.04	0.075±0.026	0.050±0.009	0.027±0.008
BPM	**0.14±0.04**	**0.054±0.017**	**0.016±0.010**	**0.007±0.007**
FR	**0.14±0.04**	0.056±0.021	0.017±0.013	**0.007±0.007**
BFGS	**0.14±0.04**	0.062±0.021	0.019±0.012	0.022±0.018
LM	**0.14±0.04**	0.056±0.018	0.031±0.016	0.028±0.015

Table 1-5. Bit-Error-Rate. Architecture 2×6×1. Different values of SNR are shown in the different columns. The best results for each SNR are highlighted in bold

	5	10	15	20
BP	0.14±0.04	0.065±0.023	0.020±0.013	0.013±0.011
BPM	**0.13±0.04**	**0.040±0.012**	**0.007±0.005**	**0.001±0.003**
FR	0.14±0.04	**0.040±0.011**	0.025±0.018	0.009±0.010
BFGS	0.14±0.04	0.046±0.014	0.013±0.007	0.021±0.015
LM	0.14±0.04	0.048±0.015	0.026±0.034	0.023±0.022

Although BPM was not the fastest algorithm, Tables 1-4 and 1-5 show that BPM did yield the most accurate results.

4.2 Modelling Problems

Three different function modelling problems were used to carry out a comparison of algorithms' modelling capabilities. First, two simple functions (f_1 and f_2) were used (Sexton et al., 2004); in addition, a more complex function was also used (f_3):

$$\begin{cases} f_1(x_1, x_2) = x_1 + x_2 \\ f_2(x_1, x_2) = x_1 \cdot x_2 \\ f_3(x_1, x_2) = \sin c(x_1) \cdot \sin c(x_2) \end{cases} \tag{29}$$

where $\sin c(x) = \dfrac{\sin(\pi \cdot x)}{\pi \cdot x}$.

Table 1-6. Problem x_1+x_2. Architecture 2×2×1. Frequency of relative positions of the different algorithms in terms of convergence speed. The most frequent algorithm for each relative position is highlighted in bold

	1º	2º	3º	4º	5º
BP	1	4	2	41	**52**
BPM	1	0	0	**54**	45
FR	29	28	**43**	0	0
BFGS	21	**61**	17	1	0
LM	**85**	6	7	1	1

Two hundred patterns formed by pairs (x_1,x_2) uniformly distributed in the range (-1,1) were used as training data set. Every experiment was carried out 100 times with the same initialisation for every neural network. As in the case of classification problems, convergence speed and accuracy was used to benchmark algorithms' performance. With respect to convergence speed, the convergence threshold was taken as a 15% of the initial MSE. Results are shown in Tables 1-6, 1-7 and 1-8.

Table 1-7. Problem $x_1 \cdot x_2$ Architecture 2×4×1. Frequency of relative positions of the different algorithms in terms of convergence speed. The most frequent algorithm for each relative position is highlighted in bold

	1º	2º	3º	4º	5º
BP	10	3	11	**43**	33
BPM	10	0	1	38	**51**
FR	**68**	23	9	0	0
BFGS	57	**31**	11	0	1
LM	61	**7**	**22**	3	7

Table 1-8. Problem $\text{sinc}(x_1)\cdot\text{sinc}(x_2)$. Architecture 2×4×3×1. Frequency of relative positions of the different algorithms in terms of convergence speed. The most frequent algorithm for each relative position is highlighted in bold

	1º	2º	3º	4º	5º
BP	33	5	10	22	**30**
BPM	33	6	12	**38**	11
FR	**49**	22	**29**	0	0
BFGS	59	5	11	8	17
LM	42	**37**	10	6	5

Note that LM is not the best choice always (Nelles, 2001); in fact, FR algorithm shows a faster convergence speed than LM in Tables 1-7 and 1-8.

In order to test the accuracy, the error of the neural network after training for a data set different from the training set, was measured. Five thousand patterns uniformly distributed within the range (-1, 1) were used for this accuracy test. Average results of 100 tests are shown in Table 1-9.

Table 1-9. Accuracy yielded (MSE) by the different algorithms in the three proposed problems. The best result for each problem is highlighted in bold

	x_1+x_2 2x2x1; 500 epochs	$x_1 \cdot x_2$ 2x4x1; 1000 epochs	$\text{sinc}(x_1)\cdot\text{sinc}(x_2)$ 2x4x3x1; 1500 epochs
BP	$(8.12\pm1.26)\cdot10^{-3}$	$(12.15\pm1.34)\cdot10^{-3}$	$(19.83\pm3.37)\cdot10^{-3}$
BPM	$(1.01\pm0.09)\cdot10^{-3}$	$(2.15\pm0.32)\cdot10^{-3}$	$(1.82\pm0.39)\cdot10^{-3}$
FR	$(8.55\pm1.67)\cdot10^{-5}$	$(6.37\pm1.41)\cdot10^{-5}$	$(1.45\pm0.21)\cdot10^{-4}$
BFGS	$(1.17\pm0.09)\cdot10^{-6}$	**$(6.63\pm3.03)\cdot10^{-8}$**	**$(2.41\pm0.84)\cdot10^{-5}$**
LM	**$(1.30\pm0.67)\cdot10^{-8}$**	$(5.89\pm11.79)\cdot10^{-4}$	$(3.11\pm4.54)\cdot10^{-4}$

Table 1-9 shows that the fastest algorithms were also the most accurate ones. Nevertheless, results achieved by other algorithms were also quite accurate, hence they could also be a good choice in applications involving a high computational burden; for instance, applications involving a large number of inputs (high dimensionality).

5. CONCLUSIONS

A review of classical training methods has been provided in this chapter. It is mainly focused on the most widely used neural model, the so-called Multilayer Perceptron. It shows many attractive features; e.g., it is a universal function approximator and it is able to carry out non-linear classification. Classical training algorithms, based on the first and second derivatives, have been described. Several experiments applied to standard problems have been used to benchmark the capabilities of the different training algorithms.

Unfortunately, the reduced length of this chapter does not allow to summarise all the learning algorithms, new applications, theoretical developments and research directions related to neural models. Readers are encouraged to consult the excellent texts provided in the bibliography, as well as the following chapters of this book, as a nice way to get involved with the fascinating world of neural networks. More expert readers, rather concerned about last advances in this field, are encouraged to have a look at the updated issues of some excellent journals (*IEEE Transactions on Neural Networks*, *Neural Networks* and *Neurocomputing*, among others).

The practical implementation of the algorithms presented in this chapter is far less difficult than some years ago was. There are different software solutions for Statistics, e.g., SPSS® and Statistica®, which provide a neural network toolbox. Moreover, other software products used for numerical and symbolic computing, e.g. Matlab® and Mathematica®, also include a neural network toolbox. In addition, there is another software product (Neurosolutions®), which includes many different neural network implementations, thus allowing their use in a straightforward way.

REFERENCES

Arbib, M., 2003, *The Handbook of Brain Theory and Neural Networks*, MIT Press.

Bishop, C. M., 1995, *Neural Networks for Pattern Recognition*, Clarendon Press.

Chan, L. W., Fallside, F., 1987, An adaptive training algorithm for backpropagation, *Computer Speech and Language*, **2**:205-218.

Chen, S., Gibson, G. J., Cowan, C. F. N., Grant, P. M., 1990, Adaptive equalization of finite non-linear channels using multilayer perceptrons, *Signal Processing* **20**:107-119.

Cichocki, A., Amari, S., 2002, *Adaptive Blind Signal and Image Processing*, John Wiley & Sons.

Cybenko, G., 1988, Continuous valued neural network with two hidden layers are sufficient, Tech. Report, Department of Computer Science, Tufts University, Medford, USA.

Duda, R. O., Hart, P. E., Stork, D. G., 2001, *Pattern Classification*, Wiley.

Gibson, G. J., Siu, S., Cowan, C. F. N., 1991, The application of nonlinear structures to the reconstruction of binary signals, *IEEE Transactions on Signal Processing* **39**:1877-1881.

Haykin, S., 1996, *Adaptive Filter Theory*, Prentice-Hall.

Haykin, S., 1999, *Neural Networks: A Comprehensive Foundation*, Prentice Hall.

Hassoun, M. H.,1995, *Fundamentals of Artificial Neural Networks*, MIT Press.

Hecht-Nielsen, R., 1989, *Neurocomputing*, Addison-Wesley.

Jacobs, R. A., 1988, Increased rates of convergence through learning rate adaptation, *Neural Networks* **1**:295-307.

LeCun, Y., 1985, Une procedure d'apprentissage pour reseau a seuil asymmetrique (a Learning Scheme for Asymmetric Threshold Networks), *Proceedings of Cognitiva* **85**:599-604.

Luenberger, D. G., 1984, *Linear and Nonlinear Programming*, Addison-Wesley.

MacKay, D. J. C., 2003, *Information Theory, Inference and Learning Algorithms*, Cambridge University Press.

Minai, A. A., Williams, R. D., 1990, Acceleration of backpropagation through learning rate and momentum adaptation, in: *Proceedings of IJNCNN-90*, pp. 676-679.

Moreira, M., Fiesler, E., 1995, Neural networks with adaptive learning rate and momentum terms, Technical Report 95-04, IDIAP, Martigny, Switzerland.

Nelles, O., 2001, *Nonlinear System Identification From Classical Approaches to Neural Networks and Fuzzy Models*, Springer.

Orr, G. B., Müller, K. R., 1998, *Neural Networks: Tricks of the Trade*, Lecture Notes in Computer Science, Springer.

Press, W. H., Flannery, B. P., Teukolsky, S. A., Vetterling, W. T., 1992, *Numerical Recipes in C*, Cambridge University Press.

Proakis, J. G., 2001, *Digital Communications*, McGraw-Hill.

Qureshi, S. U. H., 1985, Adaptive equalization, *Procs. of the IEEE* **73**:1349 1387.

Reed, R., 1993, Pruning Algorithms: A Survey, *IEEE Transactions on Neural Networks* **4**(5):740-747.

Reed, R. D., Marks II, R. J., 1993, *Neural Smithing, Supervised Learning in Feedforward Artificial Neural Networks*, MIT Press.

Riedmiller, M., Braun, H., 1993, A direct adaptive method for faster backpropagation learning: the RPROP algorithm, in: *Proceedings of IEEE International Conference on Neural Networks*, pp. 586-591.

Ripley, B. D., 1996, *Pattern Recognition and Neural Networks*, Cambridge University Press.

Rumelhart, D. E., Hinton, G. E., Williams, R. J., 1986, Learning representations by back-propagating errors, *Nature* **323**:533-536.

Schiffmann, W., Joost, M., Werner, R., 1994, Optimization of the backpropagation algorithm for training multilayer perceptrons, Technical Report, University of Koblenz, Institute of Physics, Germany.

Sexton, R. S., Dorsey, R. E., Sikander, N. A., 2004, Simultaneous optimization of neural networks function and architecture algorithm, *Decision Support Systems* **36**:283-296.

Silva, F. M., Almeida, L. B., 1990, Acceleration techniques for the backpropagation algorithm, in: *Proceedings of the EURASIP Workshop*, Lecture Notes in Computer Science, vol. 412 of Lecture Notes on Computer Science, Springer-Verlag, pp. 110-119.

Weigend, A. S., Gershenfeld, N. A., 1993, *Time Series Prediction: Forecasting the Future and Understanding the Past*, Addison-Wesley.

Werbos, P. J., 1974, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*, PhD thesis, Harvard University, Cambridge, MA, USA.