Chapter 3

Fundamentals of Neural Network

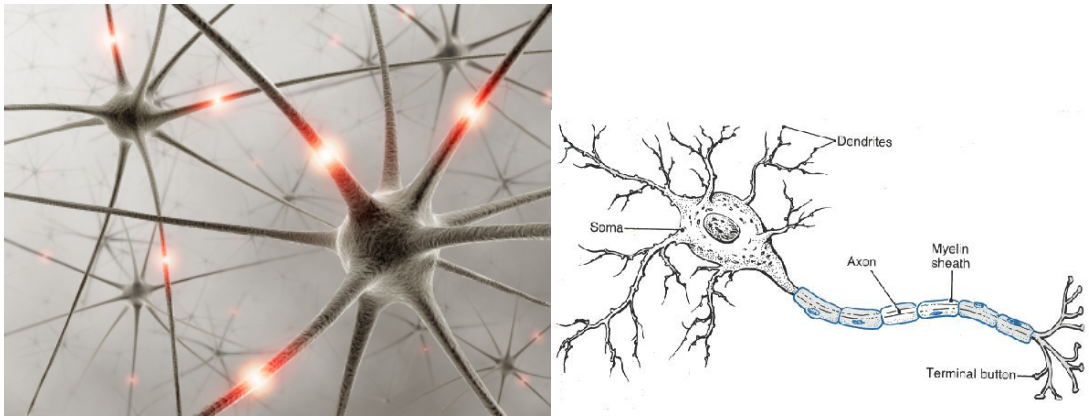
One of the main challenge in actual times researching, is the construction of AI (Artificial Intelligence) systems. These systems could be understood as any physical or logical device capable of carrying out the task for it was designed.
The Artificial Neural Network (ANN) belong to a subgroup of the AI systems. It is called *bottom-up,* due to the fact that the initial systems are simples and identical, and they generate more complex systems through a learning process.
The field of Neural Networks has arisen from diverse sources, ranging from the fascination of mankind with understanding and emulating the human brain, to broader issues of copying human abilities such as speech and the use of language, to the practica commercial, scientific, and engineering disciplines of pattern recognition, modelling, and prediction.
In Neural Networks, as in AI, the excitement of technological progress is supplemented by the challenge of reproducing intelligence itself.

Neural Networks approaches combine the complexity of some of the statistical techniques with the machine learning objective of imitating human intelligence. However, it is done at a level where there is no accompanying ability to make learned concepts transparent to the user, as it is with other techniques like decision trees [27].

3.1 Artificial Neural Networks (A.N.N.)

3.1.1 Biological Fundamentals

ANN are inspired by the biological nervous system in humans and animals. The neural communication system conformed by the nervous and hormonal system, connected to sense and effector organs is conformed by three parts [26]:



(a) Nervous System Sample **(b)** Biological Neurone

Figure 3.1: Biological Fundamentals of a Nervous Cell

1. Receptors, which are in sensorial cells, they collect the information by means of stimulus, either from medium or from inside the organism.
2. Nervous System. It recieves informations, and it sends them, already manufactured to effector organs and other places in the nervous system. Figure 3.1a.
3. Effector Organs (muscles and glands), which receive the information, and they interpret in the form of motor or hormonal actions.

The more essential structural and functional, in the communication neural system is the nervous cell or neurone (Figure 3.1b). They establish communications each other by means of chemical signals between axons, which send, and dendrites, which receive signals.

The mision from neurones covers approximately five functions:

- They collect the information from other neurones or receptors in the form of impulses.
- They integrate the information in an activation code.
- They transmit the codified information, in form of frequency impulses through their Axon.
- The axon carries out the distribution through its ramifications.
- In its terminals, it transmits the impulses to the next neurones or the effector cells.

Given a signal, the synaptic process might increase (excite) or decrease (inhibit) electrical potential. The neuron will fire (send), when its electrical potential reaches a threshold. Learning process might occur by changes to synapses. Our nervous system, and from the rest of animals, work in this way. And these are the mechanisms, that ANN try to incorporate throughout its history.
The conectivity between neurones is too elevated. The brain has about 10^{11} neurones and 10^{14} synapses, what is an enormous network, and impossible to implement (until now) with the present technology.

3.1.2 General Model of A.N.N.

There exist many different models of ANN, which follow different design strategies, learning rules and output construction functions. A first classification, is made attending to the route that follow the information inside the network, and in this way, it can be distinguished two types: feed-forward networks and feed-backward networks.
Now it will be described the general computational model used to develop an ANN.

3.1.2.1 Artificial Neurone

An artificial neurone is a mathematical function which is the constitutive unit in an ANN. The first one was proposed by Warren McCulloch and Watter Pitts in 1943. Althought they were quite different to the actual neurone models, at this time with these Threshold Logic Units (as they were called) it was possible to implement any boolean function. Since its developed some variants were introduced to the units, like the Perceptron, developed by Fank Rosenblatt, which used a linear threshold function. In the last years, neurones with more continuous shapes started to be considered [31].

The actual artificial neurone receives one or more inputs (dendrites), the input values are ponderated by the weights of each connection and summed to produce a value. This sum could be understanding as the synaptic process. After that it is summed a bias value, that belong to the neurone which simulates the synaptic threshold in biological neurones. Finally is applied a function 'φ' (activation function) to obtain the output value. All this calculations are described in Equation 3.1, for m input connections to the neurone k.

This activation function can be linear, non-linear, threshold function, or even any function it could be conceived. But usually is recommended the use of non-linear functions, such as sigmoid or hyperbolic tangent.

$$y_j^k = \varphi(\sum_{i=0}^{n_{k-1}} w_{ij}^k x_i^k + b_j^k) \tag{3.1}$$

In Figure 3.2 can be observed the common model and the similarities with the biological neurone can be observed.

3.1.2.2 Basic Structure of the A.N.N.

Generally A.N.N. consist of layers of interconnected nodes, each node producing a non-linear function of its summed input. The input to a node may come from other nodes or directly from the input data. Also, some nodes are identified with the output of the network. The complete network therefore represents a very complex set of inter-dependencies which may incorporate any degree of nonlinearity, allowing very general

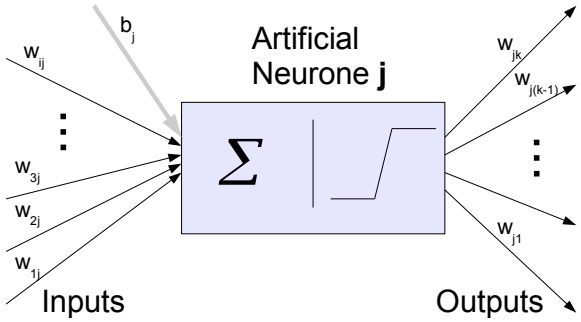


Figure 3.2: Artificial Neurone Schema

functions to be modelled [26].

The basic interconection structure between units is the Multilayer Network, shown in Figure 3.3. The first level is constituted by the input cells, these units receive the inputs to the Networks. Next to the input layer, there is a serie of intermediate layers, called hidden layers. There can be one or more hidden layers, where the last one, is the output level. The output of these units, is the output of the network.

Each connection between cells is like a communication route: through this connections flow numeric values, which are weighted by the weights on each link. These weights are adjusted during the learning phase to produce the final ANN.
All these can be resumed, like: *input units store the input vectors, hidden units transform the inputs into an internal numeric vector and output units transform the hidden values into the prediction.*
So if the ANN has *k-layers*, the output will result as in equation 3.2, where $\vec{X}$ is the input vector,φ the activation function, and W_{ij} the weight matrix of the k-hidden layer, where the weight belongs to the link between i-unit in (k-1)-layer and the j-unit in k-layer.

$$\vec{o} = \varphi(\ldots\varphi(\varphi(\vec{X}\cdot W_1)W_2)\ldots W_k) \tag{3.2}$$

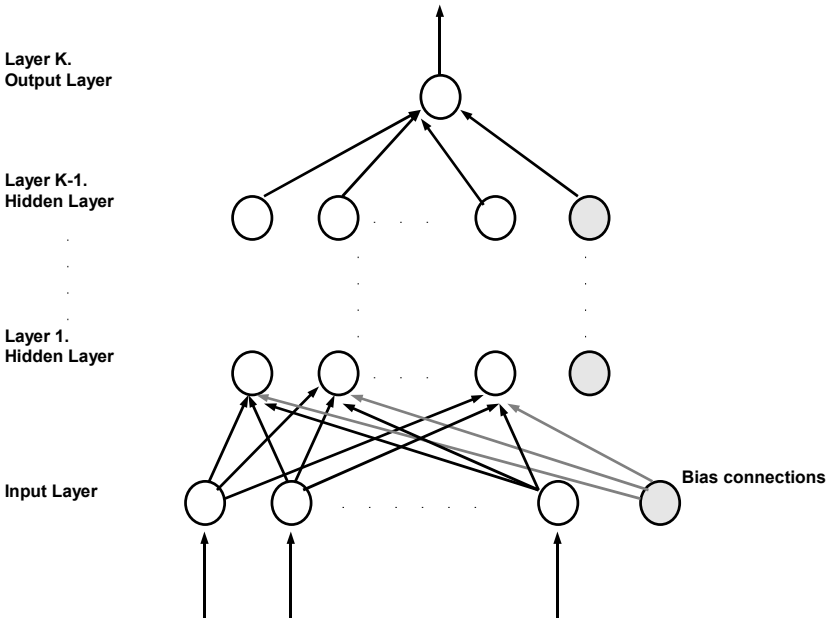


Figure 3.3: General structure of a Multilayer Network

3.1.2.3 Learning Stage

Maybe the most important part in an ANN. The learning schema in a network, is what establish the problems that the network will be able to solve. The ability of a network to solve a problem will be linked, to the available examples (patterns) in the learning stage.

The learning dataset should be significant (if it is limited, the network will not adjust the weights/bias well) and representative (there must exist diversity of examples in the dataset). Anyway in Chapter 4, it will be explained/carried out in great details, the optimization phase of a dataset.

Learning algorithms fall into three groups with respect to the sort of feedback that the learner has access to:

- Supervised learning: for every input, the network is provided with a target or class; that is, the environment tells the network what its response should be. The network then compares its actual response to the target and adjusts weight/bias values in such a way that it is more likely to produce the appropriate response the next time it receives the same input.
- Unsupervised learning: the network receives no feedback from the environment. Instead the network's task is to represent the inputs in a more efficient way, as clusters or categories or using a reduced set of dimensions. Unsupervised learning is based on the similarities and differences among the input patterns.
- Reinforcement learning: is much closer to supervised than unsupervised learning, the network receives feedback about the appropriateness of its response.

The avaliable dataset in this research is provided, with the output attribute values, so it means that in this case it will be worked with Supervised Learning Algorithms. Some of these algorithms are Backpropagation, Batch Backpropagation, Reasilient Backpropagation and some others, that will be explained in great details in the last section of this chapter. It would be interesting to mention, that all the learning algorithms that will be implemented in this research, are based in this backpropagation idea, which is simply the retropropagation of a gradient of the output error through the hidden neurones. Anyway it will be described in the next sections.

The leaning stage in ANN consists in determinate the most accurated weights/bias values for all the connections, which solve more efficiently the proposed problem. The normal learning process consists of: introducing the learning samples, and then modifying the weights/bias, following a particular learning algorithm. Once all the patterns have been introduced, it is checked if the convergence criterion was achieved, and if not the process is repeated, and all the patterns are again introduced.
The convergence criterion can be implemented in three forms:

- **Fix number of epochs**. It is decided how many times will be introduced the learning samples.
- **Goal definition**. It is established a performance value (normally Mean Absolut Error or Mean Squared Error), and when a goal value is reached, the learning phase will end.

- **Early stop**. It will be defined a validation set, from the dataset. And if there is no improvement in a determinated number of epochs in this validation set, then, the learning will end.

In this research all these criterions where implemented in the simulations. It will be also necessary to distingish between the possible learning strategies. Here were implemented the two most common. The first one consists on to make a partition in the dataset, where a percentage (normally 60%) was for learning tasks, and the rest for validation tasks. And the second one, was Cross-Validation.
K-fold-Cross-Validation is generally used when one wants to estimate how accurately a predictive model will perform in practice. The original sample is partitioned into K subsamples. Of the K subsamples, a single subsample is retained as the validation data for testing the model, and the remaining K-1 subsamples are used as training data. The cross-validation process is then repeated K times (the folds), with each of the K subsamples used exactly once as the validation data. The K results from the folds then can be averaged to produce a single estimation. The advantage of this method over repeated random sub-sampling is that all observations are used for both training and validation, and each observation is used for validation exactly once, and hence the generalization capabilities of the A.N.N. will be increased, but at the expense of more computational cost [3].

What is expected to improve using this second strategie, is the avoiding of the overfitting phenomenon, that takes place when the result model does not fit so well the learning samples as the validation samples.

Finally to resume this subsection the learning stage could stay as follows:

1. `Initialize all the parameters (weights/bias)`
2. `Selection of training and validation set`
   (a) `Introduce the training set, and adjust parameters according to the learning algorithm`
   (b) `Introduce the validation set, and evaluate performance of the network`
   (c) `If the convergence criterion is achieved the network model is finished, if not go back to 2.a`

3.2 Feed Forward Networks

3.2.1 MultilayerPerceptron

A MLP (Multilayer Perceptron) or as it is also known, Multilayer Network with forward connections, is a generalization of the Simple Perceptron, and it came up due to the limitations of the latter one. In 1986, Rumenhart, Hinton and Willians, they presented a way to retropropagate the output errors (Backpropagation) measured of the network to the hidden neurons of the network, this was the generalized Delta Rule. Some authors have demostrated [10] that the MLP is an universal aproximator, in the sense that any continuous function over a $\Re^n$ space can be approximated with a MLP, with at least one hidden layer.

Actually the MLP is one of the most used architectures to sove learning machine problems, due to their capability as universal approximator, as well as to their easy use and aplicability. They have been applied to solve problems in different areas like speech recognition, character recognition, control of processes, medical diagnoses, time series prediction...

However, it could be important to point out, that although it is one of the most used and known networks, it does not involve that it will be one of the most powerful or with the best results in the different application areas. In fact, the MLP possess some limitations, as its long learning process for complex problems or the dificulty to make a theoretical analysis of the network due to the presence of non-linear components and to the big connectivity.

3.2.1.1 M.L.P. Architecture

As it was mentioned before, the M.L.P. as the general Multilayer Network, it is formed by three types of different layers: input layer, hidden layer(s) and output layer (see Figure 3.3). The neurones in the input layer only propagate the signals (input vector) arriving to them, it can be understood then as if their activation function were linear. They send the signals to all the neurones in the next layer, which carry out a non-linear processing (or also linear with saturation in this research, see Figure 3.4), and then the signals travel (each hidden neurone sends the signal to all the neurones in the next

layer) through the *(k-1)hidden layers*, until they arrive to the output layer, it is then said that the network has total connectivity.

In the output layer is obtained the output of the network. With these output values or vectors a non-linear processing can be made or not (giving the direct result). For this research, as it will be later mentioned, the labelled data was normalized in the range [-1,1], so what it was made was to incorporate the hyperbolic tangent as the output layer activation function, to avoid output values greater or lower as the maximum or minimum expected.

Each connection in the network has associated a real number, called weight of the connection. And each neurone has associated a threshold or bias value, which for the case of MLP, it is threated like another connection to the neurone with input constant "1". These two parameters are the essence of the learning stage, w_{ij}, b_j.

3.2.1.2 Propagation from patterns

Now it is the time for introducing the background theory in this MLP. What is going to be first made, is to define all the parameters involved in this Neural Network with their appropiate indexes.
The Multilayer Perceptron (see Figure 3.3) defines a non-linear relationship between the input and the output patterns, which is obtained propagating forwards the inputs. For this task, each neurone process the information received, and generates the activation which is again propagate towards the connections with the next layer.

Although almost all the simulations will be carried out with one or two hidden layers, here the MLP equations will be defined in the K-layers case.

Be a MLP with $K - layers$ ($K-1$ hidden layers) and n_k neurones in layer k, being $k = 1, 2, \ldots, K$. Be $W^k = (w_{ij}^k)$ the weight matrix ($k = 1, 2, \ldots, K$) with (w_{ij}^k) representing the weight from neurone i (layer $k-1$) to the neurone j (layer k); and be $B^k = (b_i^k)$ the bias (threshold) vector from neurones in layer k being $k = 1, 2, \ldots, K$. Finally it is neccesary to describe the output or activation value of each neurone, defined as a_i^k with the same meaning as the index for b, but this time $k = 0, 1, 2, \ldots, K$. This activation value is calculated as follows:

$$a_i^k = \begin{cases} x_i & \text{if } k = 0 \\ \varphi\Big(\sum_{j=1}^{n_k-1} w_{ji}^{k-1} a_j^{k-1} + b_i^k\Big) & \text{if } k = 1, 2, 3, \ldots, K \end{cases} \tag{3.3}$$

Where x_i are the elements of the input pattern vector $\vec{X} = (x_1, x_2, \ldots, x_{n0})$. When $k = K$, a_i^k are the output values, that will be named as y_i, the components of the output vector $\vec{Y} = (y_1, y_2, \ldots, y_{nK})$.

φ is the activation function common to all the neurones in the layer. As is shown in Figure 3.4 there are three possibilites to choose, both of them with saturation areas from [-1,1] or [0,1] for the output values. These are:

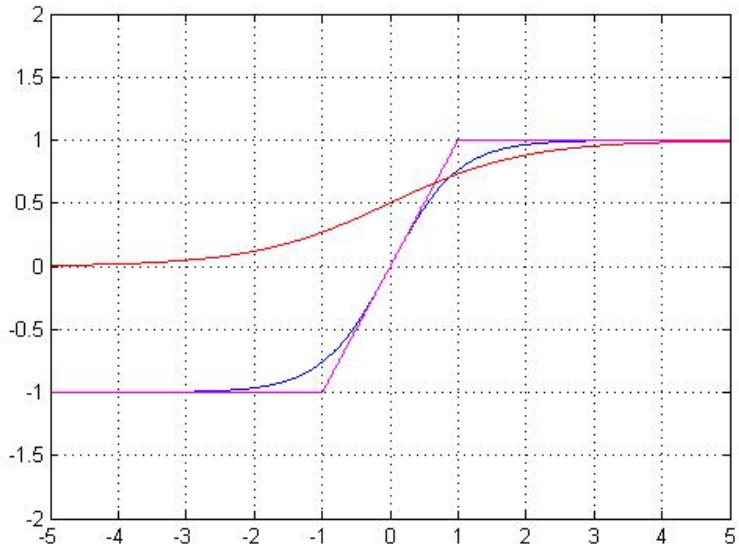


Figure 3.4: Typical Activation/Transfer Functions

- Sigmoidal function:

$$\varphi(x) = \frac{1}{1 + e^{-x}} \tag{3.4}$$

- Hyperbolic tangent function:

$$\varphi(x) = \frac{1 - e^{-x}}{1 + e^{-x}} \tag{3.5}$$

- Lineal saturation function:

$$\varphi(x) = \begin{cases} -1 & \text{if } x < 1 \\ x & \text{if } -1 \leq x \leq 1 \\ 1 & \text{if } x > 1 \end{cases} \tag{3.6}$$

Normally the activation function is common to all neurones in the network, but this time, during the searching of the best configuration, it will be contempled different functions for the hidden layers but keeping the output layer with the hyperbolic tangent function for convenience.

The election of one activation function, or a determinated number of neurones in each layer, will be discussed in Chapter 5. Now it is the time to describe the basic learning rule or algorithm.

3.2.1.3 BACKPROPAGATION Algorithm

The learning algorithm is the mechanism by means of which the parameters in the A.N.N. are adjusted and modified in the search of the best results. The most common algorithm for this task is the *BackPropagation Algorithm* ([11]).
It is an algorithm belonging to the class of supervised algorithms, so for each input pattern, the network must be also provided with a desired output pattern or, as it will be called since now, a target, t_i.
So what it must be faced, is a minimation problem, which objective is to obtain an output the most similar possible to the target. This problem could be expresed as in Equation 3.7.

$$Min_{w,b}E \tag{3.7}$$

Where E is a funtion of the error (see Equation 3.8), which can be evaluated as the sum of Mean Absolut Error of all the patterns, Mean Squared Error, or another performance function, that can be also contempled. The index D indicates the number of patterns in the training set.

Note 2 *All the equations are represented in the case of one neurone in the output layer.*

$$E = \frac{1}{D}\sum_{d=1}^{D} e(d) \tag{3.8}$$

$$e(d) = \text{ Performance Function}\Big(t_i(d) - y_i(d)\Big) \tag{3.9}$$

Being $Y(d) = (y_1(d))$ and $T(d) = (t_1(d))$ the output of the network and the desired target, respectively, for the pattern d.
So if W, B, are a minimum of the error function E, the error will be next to zero, what entails that the network output is similar to the desired output, the learning objective. Due to the fact that there are non-linear functions, this minimization problem will be non-linear too. What means, that it should be neccesary to use non-linear optimization techniques for its resolution.
These techniques are based in and adjustment of the parameters following a particular search direction. In the context of ANN, and more particularly for the MLP, the searching direction commonly used is the negative direction of the error function gradient, "*Gradient Descent Method*". Anyway they have been developed other methods to find these error function minimums, like random search method or evolutive techniques. An algorithm belonging to the latter possibility, is described in Section 5.2, and it was applied, and the results shown in the same section, to find the best configuration of the Neural Network. In Section 6.2 are shown the results when they were used as a learning algorithm.

In this moment the adjusting of the parameters can be understood as a simple actualization of weights and bias by adding Δw. It will be differenced two possibilities now from this Backpropagation Algorithm, these are online Backpropagation and Batch Backpropagation. The difference is that whereas the Online algorithm consists in a sucesive minimization of the error function for each presented pattern, $e(d)$, the Batch algorithm carries out a minimization of the total error E. It will be compared which of these algortihms provide better results during the simulations, in the Chapter 5.

Both online and batch backpropagation are presented in Algorithms 1 and 2, it can be observed how they work and the result equations which were before discussed. The parameter α is the learning rate, which has influence in the displacement on the error surface. ΔW represents the matrix of increments to all the weights and biases in the network, where the index i denotes the corresponding neurone. In the presented algorithm, the propagation of the error through the neurones (δ function), is not explained for simplicity.

Algorithm 1 Gradient Descent Algorithm. Online Backpropagation

initialize w
while stopping criterion is false **do**
  for each $(x_d, t_d) \in D$ **do**
    for each $\Delta w_{ij} \in \Delta W$ **do**
      $\Delta w_{ij} \leftarrow -\alpha \frac{\partial e(n)}{\partial w_{ij}}$
    end for
    for each $w_{ij} \in W$ **do**
      $w_{ij} \leftarrow w_{ij} + \Delta w_{ij}$
    end for
  end for
end while
return w

Algorithm 2 Gradient Descent Algorithm. Batch Backpropagation

initialize w
while stopping criterion is false **do**
  for each $\Delta w_{ij} \in \Delta W$ **do**
    $\Delta w_{ij} \leftarrow 0$
  end for
  for each $(x_d, t_d) \in D$ **do**
    for each $\Delta w_{ij} \in \Delta W$ **do**
      $\Delta w_{ij} \leftarrow \Delta w_{ij} - \alpha \frac{\partial e(n)}{\partial w_{ij}}$
    end for
  end for
  for each $w_{ij} \in W$ **do**
    $w_{ij} \leftarrow w_{ij} + \Delta w_{ij}$
  end for
end while
return w

Note 3 *The bias values are not included in the algorithm because they are assumed to be another weight with input (output from the previous neurone) constant and equal to "1".*

Due to the fact that the neurones are grouped in different layers and connected between them, it is possible to apply the gradient method in an efficient form, resulting this Backpropagation algorithm or as it is also known generalized delta rule. Backpropagation term is used due to the form of implement this method in the MLP, since the output error is propagated backwards, transforming it in an error for each neurone in the network.
This algorithm is one of the most used in A.N.N. simulations but, as it was before said, it has many limitations, on of the most importants is the long convergence time.

In Section 3.3 some other algorithms, variants of this Backpropagation Algorithm, are described. In the simulations all these algorithms will be implemented using Matlab, so it will not be neccesary to programme all these functions at any Programming Language, due to that Matlab have all them already implemented in the Neural Network Toolbox [24].

3.3 Learning Algorithms

In this section will be introduced some variants from the Backpropagation Algorithm, which were also implemented in the simulations. These are: Gradient Descent with Momentum Backpropagation, Levenberg Marquardt Backpropagation and Resilient Backpropagation.

3.3.1 Gradient Descent with Momentum Backpropagation

This method emerge to solve the possible instabilities in the Backpropagation Algorithm due to the learning rate. As it was already mentioned this parameter is which take care of controlling how much the weigths are moved around the network through the error surface. Big values of this parameter could achieve a rapid convergence at cost of the possibility of oscillate around a minimun or even jump this minimum of the error. And small values could avoid this problem but they cause a slower convergence [26].

What this method makes is to modify the Backpropagation learning rule, by adding a second member, which is called *Momentum*, obtaining the next learning rule:

$$w(n) = w(n-1) - \alpha \frac{\partial e(n)}{\partial w} + \gamma \Delta w(n-1) \tag{3.10}$$

Note 4 *The biases parameters are considered as another weight but with input (output from previous neurone) constant and equal to "1".*

Where Δw is the increase of the parameter in the previous iteration, and γ is a new configurable parameter called *Momentum Constant* ($0 \leq \gamma < 1$), which controls the assigned importancy to the previous increase. This rule was again proposed by Rumelhart [11], and keeps the Backpropagation properties, attending to modify the network's parameters minimizing the Error function (Equation 3.8). The new term gives to the original method some inertia, which could avoid oscillations.
Applying successively the Equation 3.10 the learning rule can be expressed in another form:

$$w(n) = w(n-1) - \alpha \sum_{t=0}^{n} \gamma^{n-t} \frac{\partial e(t)}{\partial w} \tag{3.11}$$

where it can be observed that the new addition to a parameter depends on all the previous variations, leading to a more stable method without abrupt oscillations.

3.3.2 Levenberg Marquardt Backpropagation

This is a second order technique to solve optimization problems, and it uses to be more efficient than the classic Backpropagation method, although it requires much more memory. It was applied for the first time to a MLP by Hagan y Menhaj [13], which demonstrated that it is much more efficient than other techniques when the network contains no more than a few hundred weights.
The performance function for the network is:

$$V(w) = \frac{1}{2} \sum_{i=1}^{D} \sum_{i=1}^{n_K} [t_i - y_i(\mathbf{w})]^T [t_i - y_i(\mathbf{w})] = \frac{1}{2} \sum^{N} e(\mathbf{w})^T e(\mathbf{w}) \tag{3.12}$$

with $\mathbf{w} \in \Re^{1 \times L}$ is the vector with all the parameters in the network (weights/biases), and $N = D \times n_K$. While backpropagation is a steepest descent algorithm, the Marquardt-Levenberg algorithm is an approximation to Newton's method. The idea is to minimize

the function $V(w)$ with respect to the parameter vector $\mathbf{w}$. Then Newton's method would stay as:

$$\Delta\mathbf{w} = -[\nabla^2 V(\mathbf{w})]^{-1}\nabla V(\mathbf{w}) \tag{3.13}$$

where $\nabla^2 V(\mathbf{w})$ is the Hessian matrix and $\nabla V(\mathbf{w})$ is the gradient. Then being $V(\mathbf{w})$ as in Equation 3.12, it can be shown:

$$\begin{aligned} \nabla V(\mathbf{w}) &= J^T(\mathbf{w})e(\mathbf{w}) \\ \nabla^2 V(\mathbf{w}) &= J^T(\mathbf{w})J(\mathbf{w}) + S(\mathbf{w}) \end{aligned} \tag{3.14}$$

being $J(\mathbf{w})$ the Jacobian matrix

$$J(\mathbf{w}) = \begin{pmatrix} \frac{\partial e_1(\mathbf{w})}{\partial w_1} & \frac{\partial e_1(\mathbf{w})}{\partial w_2} & \cdots & \frac{\partial e_1(\mathbf{w})}{\partial w_L} \\ \frac{\partial e_2(\mathbf{w})}{\partial w_1} & \frac{\partial e_2(\mathbf{w})}{\partial w_2} & \cdots & \frac{\partial e_2(\mathbf{w})}{\partial w_L} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial e_{n_K}(\mathbf{w})}{\partial w_1} & \frac{\partial e_{n_K}(\mathbf{w})}{\partial w_2} & \cdots & \frac{\partial e_{n_K}(\mathbf{w})}{\partial w_L} \end{pmatrix} \tag{3.15}$$

and

$$S(\mathbf{w}) = \sum_{i=1}^{n_K} e_i \mathbf{w} \nabla^2 e_i \mathbf{w} \tag{3.16}$$

For the Gauss-Newton method $S(\mathbf{w}) \approx 0$, and after adding the Marquardt Levenberg modification to the Equation 3.13, the $\mathbf{w}$ increase will finally stay as follows:

$$\Delta\mathbf{w} = [J^T(\mathbf{w})J(\mathbf{w}) + \eta I]^{-1} J^T(\mathbf{w})\mathbf{e}(\mathbf{w}) \tag{3.17}$$

The parameter η is multiplied by β each time that a step will result in a positive $\Delta\mathbf{w}$. And in the other side if $\Delta\mathbf{w}$ is negative, η will be divided by β.
The algorithm would stay as follows:

1. Present all the training samples to the network and compute the outputs and errors. Then compute the sum of squares of errors for all the inputs $V(\mathbf{w})$.
2. Compute the Jacobian Matrix $J(\mathbf{w})$.
3. Solve Equation 3.17 to obtain $\Delta\mathbf{w}$
4. Compute again $V(\mathbf{w})$, but using $\mathbf{w} + \Delta\mathbf{w}$.
   (a) If the new $V(\mathbf{w})$ is smaller than in step 1, then reduce η by β, let $\mathbf{w} = \mathbf{w} + \Delta\mathbf{w}$ and go back to 1.
   (b) If the new $V(\mathbf{w})$ is not reduced, then increase η by β and go back to 3.
5. The algorithm will converge when norm of gradient becomes lower than a predetermined value or the output error becomes lower than goal.

3.3.3 Resilient Backpropagation (Rprop)

Rprop [9] is an adaptive learning rate neural network learning algorithm. The main purpose of these adaptive learning rate algorithm is to vary the learning rate with the general goal of speeding up the network convergence.
One of the main difficulties founded with the backpropagated neural networks is the long training times before convergence or even not convergence at all. This is often caused by poor choice of parameters, such as learning rate, momentum, etc... By dropping use of momentum and automatically adjusting the learning rate, then Rprop achieves faster convergence, and also requires less manual optimization of parameters.
Rprop introduces a time varying weight step Δ_{ij} to the standard backpropagation algorithm, for every weight. The operation is the same than with standard backpropagation. If the errors increasing, the weights are reduced, and when the errors decreasing the weight are increased. Nevertheless, unlike the backpropagation algorithm, the size of the adjustment is no more computed with a fixed constant, this time it is adjusted as follows:

$$\Delta_{ij}^{t} = \left\{ \begin{array}{ll} \mu^{+}\Delta_{ij}^{t-1} & \text{if } (\frac{\partial E}{\partial w_{ij}})^{t-1}(\frac{\partial E}{\partial w_{ij}})^{t} > 0 \\ \mu^{-}\Delta_{ij}^{t-1} & \text{if } (\frac{\partial E}{\partial w_{ij}})^{t-1}(\frac{\partial E}{\partial w_{ij}})^{t} < 0 \\ \Delta_{ij}^{t-1} & \text{else} \end{array} \right\} \tag{3.18}$$

where $0 < \mu^{-} < 1 < \mu^{+}$ and Δ_{ij}^{t-1} is the weight step for w_{ij} at instant t.