Chapter 4

Neural Networks and the Backpropagation Technique

4.1 Introduction

The purpose of this chapter is to introduce the neural-network principles and architectures that are discussed in incoming chapters in connection to active control. We also describe the backpropagation training technique, which plays an important role in the training strategies discussed in this thesis.

4.2 Neural Networks

A neural-network controller is a non-linear arrangement that combines the elements of a certain input-data vector in order to generate a convenient control law. The arrangement is organized in layers as we describe below. We use a two-layer architecture for all the neural-network controllers described in the following chapters (see Figure 4-1).

The input-datum vector, **p**, is a set of state and dynamic variables of the system to be controlled. The elements of the input data are multiplied by different sets of so-called weights or gains. Each weighted set is added and fed as the input of a non-linear function currently referred to as a *squashing* function. Each weighted set and its corresponding squashing function constitute a neuron. We express this as:

$$v_j^I(t) = \sum_{i=1}^{N^0} w_{ji}^I \; p_i(t) \qquad \text{for} \quad j = 1,\ldots, N^I \tag{4.1}$$

$$y_j^I(t) = A^I \tanh\left(\beta^I v_j^I(t)\right) \qquad \text{for} \quad j = 1,\ldots, N^I \tag{4.2}$$

where N^0 is the dimension of vector **p**, N^I is the number of neurons of the first layer, and w_{ji}^I are the components of the weight of the first layer, where for each value of j a different set of weights (for each neuron) is associated with the same input data **p**. The hyperbolic tangent of equation (4.2) is the squashing function of neuron j. The constants A^I and β^I are parameters which are tuned heuristically. The outputs from the squashing functions of the first layer, $\mathbf{y}^I$, constitute the input vector for the second layer. Following the same procedure, the weighting process and output of the second layer can be expressed as:

$$v^{II}(t) = \sum_{j=1}^{N^I} w_j^{II} \; y_j^I(t) \qquad \text{for} \quad j = 1,\ldots, N^I \tag{4.3}$$

$$u_c(t) = A^{II} \tanh\left(\beta^{II} \; v^{II}(t)\right) \tag{4.4}$$

where w_j^{II} is the single set of weight coefficients corresponding to the second layer. The control command is u_c, which is also the output of the neural network. The constants β^{II} and A^{II} are analogous to those described for the first layer.

4.3 The Backpropagation Algorithm for a Two-Layer, Single-Output Neural Network

The next step is to train the neural network to emphasize and mix some of the stimuli, practically ignore others, and consider the rest in moderation in order to generate the desired output. The backpropagation technique is a very powerful method to adjust the weights of the neural network. For a single-output, two-layer neural network the weights are updated at every time-step according to:

Layer *I*:

$$w_{ji}^{I}(k) = w_{ji}^{I}(k-1) + \Delta w_{ji}^{I}(k) \tag{4.5}$$

Layer *II*:

$$w_{j}^{II}(k) = w_{j}^{II}(k-1) + \Delta w_{j}^{II}(k) \tag{4.6}$$

In the following, we describe how to estimate $\Delta w_{ji}^{I}(k)$ and $\Delta w_{j}^{II}(k)$.

We start by defining an *error* $e = e(t)$ as the difference between the desired output from the neural network, $\delta_d = \delta_d(t)$, and the current output the neural network is producing with the current weights, $\delta_c = \delta_c(t)$. We can write this error as

$$e(t) = \delta_d(t) - \delta_c(t) \tag{4.7}$$

To abbreviate the notation, the time argument will be omitted in the future. A measure of performance or performance index Θ is defined as half of the square of the error *e*, as

$$\Theta = \frac{1}{2} e^2 = \frac{1}{2} (\delta_d - \delta_c)^2 \tag{4.8}$$

The backpropagation algorithm (Rumelhart *et. al*. [81]), estimates the gradient in the weight space of the function Θ, and seeks to minimize it. Following Haykin [27], Figure 4-2 gives a general schematic representation of a two-layer neural network. There $y^{II} = \delta_c$ represents the output from the second layer, which is the neural-network output, and y_j^I are the outputs from the neurons on the first layer, which are the inputs to the second layer. For the output layer we define a *sensitivity factor* as

$$\frac{\partial \Theta}{\partial w_j^{II}} = \frac{\partial \Theta}{\partial e} \frac{\partial e}{\partial y^{II}} \frac{\partial y^{II}}{\partial v^{II}} \frac{\partial v^{II}}{\partial w_j^{II}} \tag{4.9}$$

From equation (4.3) we have $y_j^I = \frac{\partial v^{II}}{\partial w_j^{II}}$. Substituting this result into equation (4.9) yields

$$\frac{\partial \Theta}{\partial w_j^{II}} = \frac{\partial \Theta}{\partial e} \frac{\partial e}{\partial y^{II}} \frac{\partial y^{II}}{\partial v^{II}} y_j^I \tag{4.10}$$

Let's define the local gradient of layer *II* as

$$\delta^{II} = \frac{\partial \Theta}{\partial e} \frac{\partial e}{\partial y^{II}} \frac{\partial y^{II}}{\partial v^{II}} \tag{4.11}$$

Then equation (4.10) can be written as

$$\frac{\partial \Theta}{\partial w_j^{II}} = \delta^{II} y_j^I \tag{4.12}$$

If we differentiate equations (4.8) and (4.7) with respect to e and y^{II}, respectively, we obtain

$$\frac{\partial \Theta}{\partial e} = e \tag{4.13}$$

$$\frac{\partial e}{\partial y^{II}} = -1 \tag{4.14}$$

Substituting equations (4.13) and (4.14) into equation (4.11), gives

$$\delta^{II} = -e \; \varphi'^{II}\left(v^{II}\right) \tag{4.15}$$

where $\varphi'^{II}\left(v^{II}\right) = \frac{\partial y^{II}}{\partial v^{II}}$. Based on the Hebbian Principle, (Hebb [29]), we can apply the Δ-*Rule* whereby, at every time step, the change in the corresponding weight Δw_j^{II} is made proportional to the sensitivity function:

$$\Delta w_j^{II}(k) = \lambda^{II} \frac{\partial \Theta}{\partial w_j^{II}}(k) \tag{4.16}$$

or

$$\Delta w_j^{II}(k) = \lambda^{II}\ \delta^{II}(k)\, y_j^I(k) \tag{4.17}$$

where λ^{II} is commonly referred to as the learning parameter or learning coefficient for layer *II*.

For the first layer, we define its *sensitivity function* as

$$\frac{\partial \Theta}{\partial\ w_{ji}^{I}} = \frac{\partial\ \Theta}{\partial\ y_j^{I}} \frac{\partial\ y_j^{I}}{\partial\ v_j^{I}} \frac{\partial\ v_j^{I}}{\partial\ w_{ji}^{I}} \tag{4.18}$$

From equation (4.1) we have $p_i = \frac{\partial\ v_j^{I}}{\partial\ w_{ji}^{I}}$. Substituting this result into equation (4.18) yields

$$\frac{\partial \Theta}{\partial\ w_{ji}^{I}} = \frac{\partial \Theta}{\partial\ y_j^{I}} \frac{\partial\ y_j^{I}}{\partial\ v_j^{I}}\ p_i \tag{4.19}$$

Let's define the local gradient for layer *I* as

$$\delta_j^{I} = \frac{\partial \Theta}{\partial\ y_j^{I}} \frac{\partial\ y_j^{I}}{\partial\ v_j^{I}} \tag{4.20}$$

Then equation (4.19) can be written as

$$\frac{\partial \Theta}{\partial\ w_{ji}^{I}} = \delta_j^{I}\ \ p_i \tag{4.21}$$

If we apply the Δ-Rule to the first layer we get

$$\Delta \mathbf{w}^{I}_{ji} = \lambda^{I} \frac{\partial \Theta}{\partial \mathbf{w}^{I}_{ji}} \tag{4.22}$$

or

$$\Delta \mathbf{w}^{I}_{ji} = \lambda^{I} \delta^{I}_{j} p_{i} \tag{4.23}$$

which is analogous to equation (4.17). Here λ^{I} is the learning parameter used in the first layer, and δ^{I}_{j} is the corresponding local gradient which is given by

$$\delta^{I}_{j} = \frac{\partial \Theta}{\partial y^{I}_{j}} \frac{\partial y^{I}_{j}}{\partial v^{I}_{j}} = \left[\delta^{II} w^{II}_{j}\right] \left[\varphi'^{I}(v^{I}_{j})\right] \tag{4.24}$$

where $\frac{\partial y^{I}_{j}}{\partial v^{I}_{j}} = \varphi'^{I}\left(v^{I}_{j}\right)$. The term $\frac{\partial \Theta}{\partial y^{I}_{j}} = \delta^{II} w^{II}_{j}$ comes from

$$\frac{\partial \Theta}{\partial y^{I}_{j}} = \frac{\partial \Theta}{\partial e} \frac{\partial e}{\partial y^{II}} \frac{\partial y^{II}}{\partial v^{II}} \frac{\partial v^{II}}{\partial y^{I}_{j}} = \delta^{II} w^{II}_{j} \tag{4.25}$$

where $\delta^{II} = \frac{\partial \Theta}{\partial e} \frac{\partial e}{\partial y^{II}} \frac{\partial y^{II}}{\partial v^{II}}$ and $w^{II}_{j} = \frac{\partial v^{II}}{\partial y^{I}_{j}}$. The learning parameters of each layer (λ^{I} and λ^{II}) are determined heuristically.

To avoid local minima (see Figure 4-3) the concept of numerical momentum may also be applied (see *e.g.*, Haykin [27]). To that end a momentum coefficient μ is defined such that $0 < \mu < 1$. Then a certain memory of the evolution of every weight can be considered by computing each new weight according to

$$w_{ji}^{I}(k+1) = w_{ji}^{I}(k) + \mu^{I}\left[w_{ji}^{I}(k) - w_{ji}^{I}(k-1)\right] \tag{4.26}$$

$$w_{j}^{II}(k+1) = w_{j}^{II}(k) + \mu^{II}\left[w_{j}^{II}(k) - w_{j}^{II}(k-1)\right] \tag{4.27}$$

where k indicates the current time step. The moment coefficients μ^{I} and μ^{II} are determined heuristically.

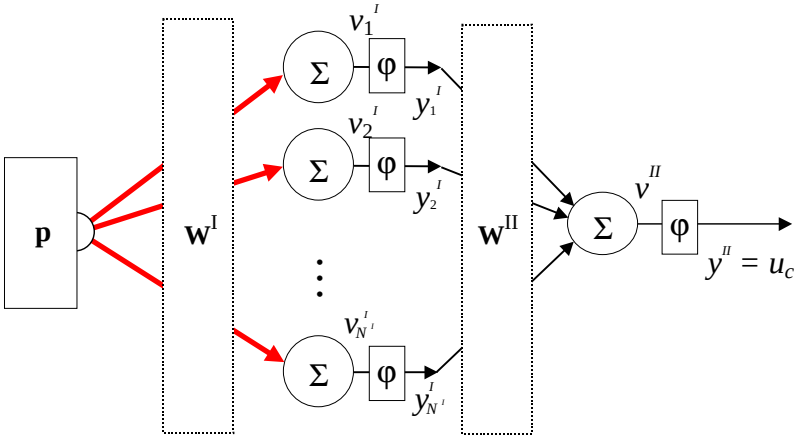

Figure 4-1: Schematic representation of a two-layer neural network. Thick arrows account for a set of axons with their origin at each input datum. Functions φ are the squashing functions of each neuron.

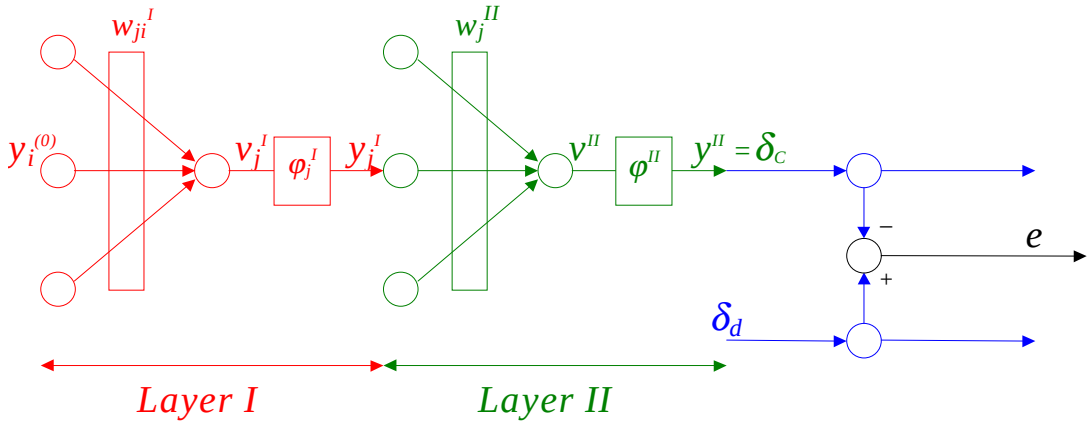

Figure 4-2: Schematic representation of a general two-layer neural network for a single-output system.

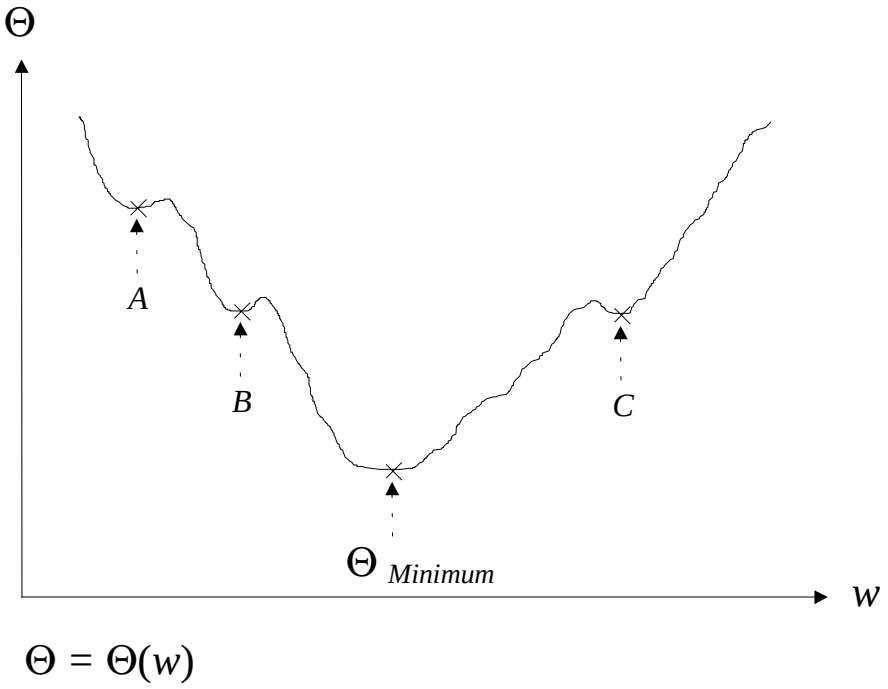

Figure 4-3: Local minima (*A*, *B*, and *C*) for a one-weight performance index Θ. If the weight adjustment Δw predicted by the training algorithm for weight *w* is too small, the training algorithm may get stuck at a local minimum. Numerical momentum helps backpropagation to overcome the local valleys at *A*, *B* and *C*.