CHAPTER II

THEORETICAL BACKGROUND

An overview of Neural Networks

Human brain is highly complex as it consists of billions of the specially built cells called neurons. Each neuron consists of four major components: soma, axon, dendrite and synapse as shown in Figure 2.1.

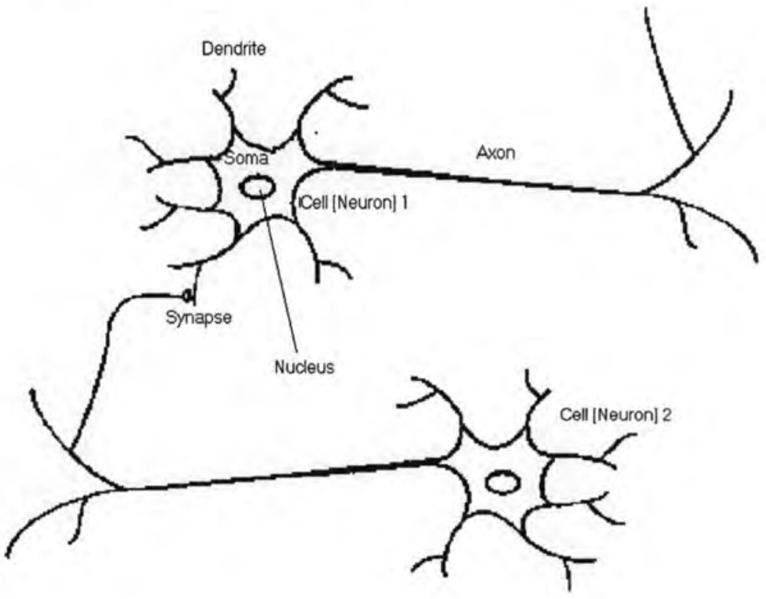

Figure 2.1: Two interconnected biological cells of neural network.

The input and output signals to the soma of a neuron are transmitted along the axon and dendrite, while the synaptic resistance controls the strength of the signal. A neuron learns to generate a particular signal by adjusting the synaptic resistance.

Neurons are connected to an enormous network called neural network. The synaptic resistance is referred as the weight of a neuron. There are three types of biological learning mechanisms which are.

1. **Supervised learning:** A neuron is forced to generate a target signal associated with a specific input pattern and to reproduce this target signal whenever the specific input pattern occurs.

2. **Unsupervised learning:** There is no target signal generated with a particular input pattern. A neuron competitively adjusts its weight equal to the value of the input pattern.

3. **Reinforcement learning:** It is a mixture of the supervised and unsupervised learning under the environment that the target cannot be explicitly known.

For artificial neural network, it is notably resemble to the biological learning, as each neuron is designed to have the same components similar to a biological neuron in the human brain as follows:

Soma	corresponds to	neuron node
Axon	corresponds to	output
Dendrite	corresponds to	input
Synapse	corresponds to	weight.

Theoretical Background

Following the objectives of this research, the neural network can be implemented using five different fundamental principles as follows.

1. The architecture of feedforward multilayer perceptron.

2. Learning Algorithm of neural network.

3. Activation Function.

4. Cost function.

5. Neural Cellular Automata.

2.1 Feedforward Multilayer Perceptron

Figure 2.2 shows a standard three-layered feedforward multilayer perceptron (MLP). This type of architecture is a part of a large class of feedforward neural networks with the neurons arranged in cascaded layers. The neural network architecture in this class share a common feature that all neurons in a layer are connected to all neurons in the adjacent layers through unidirectional branches. This feature is the forward direction, i.e., the branches or links can only broadcast information in one direction. The branches have associated transmittances or synaptic weights that can be adjusted according to the set of predefined learning rules. Feedforward networks do not allow connections between neurons within any layer of the architecture.

At every neuron, the output of the linear combiner or the neuron activity level v_q, is an input for a nonlinear activation function $f(\bullet)$, whose output is the response of the neuron. The neurons in the network typically have activity levels in the range [-1,1], and in some applications the range [0,1] is used.

In Figure 2.2, there are three-layer neural networks. The input layer does not perform any computations, but only serves to feed the input signal to the neurons of the hidden layer. The outputs of the hidden layer are fed to the output layer. The output of the output layer is the network response vector. The network can perform the *nonlinear input/output mapping* $\Omega : \Re^{n\times1} \rightarrow \Re^{p\times1}$. In general, there can be any number of hidden layers in the architecture; however, from a practical perspective, only one or two hidden layers are typically used.

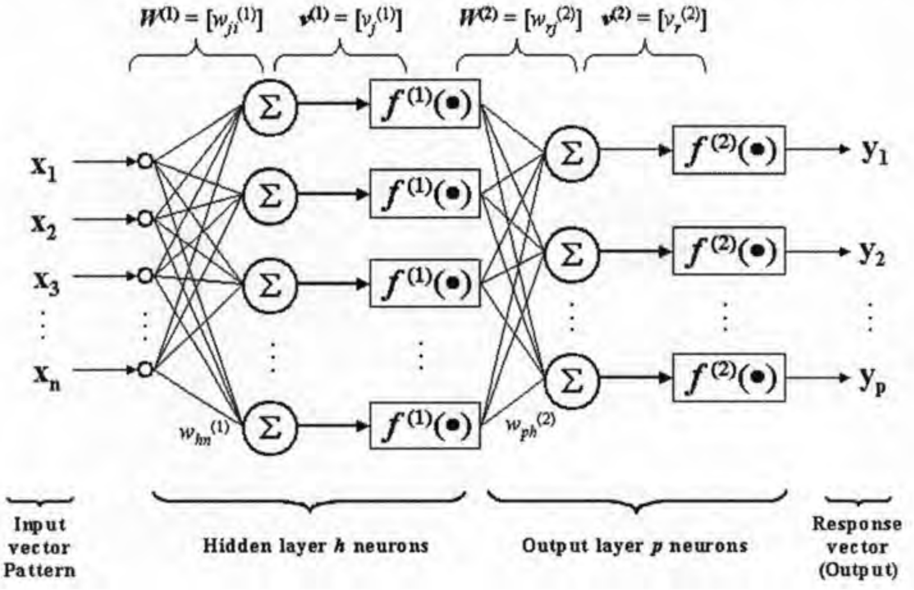

Figure 2.2: Feedforward three-layer perceptron architecture.

Each layer has a synaptic weight matrix associated with all the connections made from the previous layer to the next layer, i.e., $\mathbf{W}^{(\ell)}$, where $\ell = 1, 2$. The first layer has the weight matrix $\mathbf{W}^{(1)} = \left[w_{ji}^{(1)}\right] \in \Re^{h\times n}$, the second layer's weight matrix is $\mathbf{W}^{(2)} = \left[w_{rj}^{(2)}\right] \in \Re^{p\times h}$, for $i = 1, 2, \ldots, n; j = 1, 2, \ldots, h; and\ r = 1, 2, \ldots, p;$. The nonlinear input-output mapping $\Omega : \Re^{n\times1} \rightarrow \Re^{p\times1}$ can be determined directly from Figure 2.2 as follows.

Firstly, define $f^{(1)}(\bullet)$ as the nonlinear activation function in the hidden layer and $f^{(2)}(\bullet)$ as the nonlinear activation function in the output layer. Given the network input vector $\mathbf{x} \in \Re^{n\times1}$, the output of the hidden layer $\mathbf{x}_{out1} \in \Re^{h\times1}$ can be written as

$$\mathbf{x}_{out1} = f^{(1)}(\mathbf{v}^{(1)}) = f^{(1)}(\mathbf{W}^{(1)}\mathbf{x}) \tag{2.1}$$

which is the input to the output layer. The output of the output layer, which is the response of the network $\mathbf{y} = \mathbf{x}_{out2} \in \Re^{p\times1}$ can be written as

$$\mathbf{x}_{out2} = f^{(2)}(\mathbf{v}^{(2)}) = f^{(2)}(\mathbf{W}^{(2)}\mathbf{x}_{out1}) \tag{2.2}$$

Substituting (2.1) into (2.2) for x_{out1} gives the final response of the network as

$$\mathbf{y} = f^{(2)}(\mathbf{W}^{(2)}f^{(1)}(\mathbf{W}^{(1)}\mathbf{x})) = \Omega[\mathbf{x}] \tag{2.3}$$

A training process must be carried out by adjusting the weights to perform a desired mapping, for example, to solve a function approximation problem. Details concerning the training of MLPs using backpropagation are discussed in the backpropagation neural network section.

2.2 Learning Algorithm of neural network

2.2.1 Backpropagation Neural Network

Artificial neural networks with backpropagation was firstly developed by Werbos in 1974 [21], but this work remained unknown for many years until it was rediscovered in 1982 by Parker, in 1985 by LeCun, and by Rumelhart et al. in 1986. It was Rumelhart who presented this algorithm and made it well known in the areas of science and engineering. Feedforward multilayer perceptron (MLP) is trained by subjecting two sets of data to the backpropagation algorithm. The MLP's synaptic weights are adjusted by minimizing the difference between the actual and target outputs. Eventually, the trained network provides an association task for function approximation, classification, pattern recognition, diagnosis, etc. [22].

Standard backpropagation algorithm

Weight Initialization

Initialize each weight to a small random value.

Calculation of activation function

1. Determine the activation level of an input unit and feed to the network.
2. Determine the activation level O_j of a hidden and output unit by using the following equation.
$$O_j = F(\sum W_{ji}O_i + \theta_j) \tag{2.4}$$

where, O_i refers to an input
W_{ji} refers to the weight of input from unit i to unit j
θ_j refers to the node threshold
F refers to the activation function.

Weight Training

1. Work backward from the output units to the hidden layers recursively, and adjust the weight by using the following equation.
$$W_{ji}(t+1) = W_{ji}(t) + \Delta W_{ji} \tag{2.5}$$

where, $W_{ji}(t)$ refers to the weight at time t (or t the iteration)
ΔW_{ji} refers to the weight adjustment.

2. Compute the change of weight by using the following equations.
$$\Delta W_{ji} = \eta \delta_j O_i \tag{2.6}$$

where, η refers to a trial-independent learning rate (0< η <1, e.g., 0.3)
δ_j refers to the error gradient at unit j.

Convergence can be accelerated by adding the momentum term.
$$W_{ji}(t+1) = W_{ji}(t) + \eta\delta_j O_i + \alpha[W_{ji}(t) - W_{ji}(t-1)] \tag{2.7}$$

where 0< α <1

3. Determine the error gradients.

   - Determine the error gradient of the output units using the following equation.

$$\delta_j = (T_j - O_j)F(net_j) \tag{2.8}$$

where, T_j refers to the target output activation
O_j refers to the actual output activation at output unit j and $net_j = \sum_i W_{ji}O_i$.

   - Determine the error gradient of the hidden unit by the following equation.

$$\delta_j = F(net_j)\sum_k \delta_k W_{kj} \tag{2.9}$$

where, unit j refers to a hidden unit
δ_k refers to the error gradient at unit k to a connection points from hidden unit j.

4. Repeat iterations until convergence in terms of the selected error criterion or a maximum number of iterations is reached.

2.2.2 Levenberg-Marquardt Training Method

The Levenberg-Marquardt (LM) training algorithm [10] has been one of the learning methods most successfully imported from the optimization field to the neural network (NN). The concept of the LM algorithm is derived from Newton's method, the calculus and inversion, at each training cycle, of an approximation to the Hessian Matrix, which is the second derivatives of the performance function, in terms of the Jacobian matrix ($J(w)$) composing of all weight vectors in the network. It gives a good exchange between the speed of the Newton algorithm and the stability of the steepest descent method. The performance index ($F(w)$) is aimed to minimize the sum of squared errors (e) between the desired targets and the actual outputs. The performance index $F(w)$ is defined by the following equation.

$$F(w) = e^T e \tag{2.10}$$

where, $w = [w_1, w_2, \ldots, w_n]$ consists of all weight and bias vectors of the network
$e = [e_1, e_2, \ldots, e_p]$ refer to the error vector for all the training data (p).

The Jacobian matrix $J(w)$ composes of all weight vectors in the network defined by the following matrix.

$$J(w) = \begin{bmatrix} \frac{\partial e_1}{\partial w_1} \frac{\partial e_1}{\partial w_2} \frac{\partial e_1}{\partial w_3} \cdots \frac{\partial e_1}{\partial w_n} \\ \frac{\partial e_2}{\partial w_1} \frac{\partial e_2}{\partial w_2} \frac{\partial e_2}{\partial w_3} \cdots \frac{\partial e_2}{\partial w_n} \\ \vdots \\ \frac{\partial e_i}{\partial w_1} \frac{\partial e_i}{\partial w_2} \frac{\partial e_i}{\partial w_3} \cdots \frac{\partial e_i}{\partial w_n} \\ \vdots \\ \frac{\partial e_p}{\partial w_1} \frac{\partial e_p}{\partial w_2} \frac{\partial e_p}{\partial w_3} \cdots \frac{\partial e_p}{\partial w_n} \end{bmatrix} \tag{2.11}$$

Levenberg-Marquardt (LM) algorithm

Weight Initialization

Initialize each weight to a small random value.
Initialize the learning rate parameter μ and the decay rate β $(0 < \beta < 1)$.

Calculation of activation function

1. Determine the activation level of an input unit and feed to the network.

2. Determine the activation level O_j of a hidden and output unit by using the following equation.

$$O_j = F(\sum W_{ji} O_i + \theta_j) \tag{2.12}$$

where, O_i refers to an input
W_{ji} refers to the weight of input from unit i to unit j
θ_j refers to the node threshold
F refers to the activation function.

Weight Training

1. Compute the performance index $F(w)$ by the following equation.

$$F(w) = e^T e \tag{2.13}$$

where, e refers to the error vector composing the sum squared error for all the training data.

2. Determine the weight adjustment.

   - Save the current value of weights and determine the weight adjustment by the following equation.

$$W_{ji}(t+1) = W_{ji}(t) + \Delta W_{ji} \tag{2.14}$$

   where, $W_{ji}(t)$ refers to the weight at time t (or t the iteration)
   ΔW_{ji} refers to the weight adjustment.

   - Compute the change of weight by using the following equations.

$$\Delta w_{ji} = [J^T J + \mu I]^{-1} J^T e \tag{2.15}$$

   where, J refers to the Jacobian matrix
   μ refers to the learning rate

3. Recompute the performance index $F(w)$

4. Adjust the learning rate μ

   - If $F(w)$ obtained from the network decrease, μ is multiplied by the decay rate β.

$$\mu = \mu \cdot \beta \tag{2.16}$$

   - If $F(w)$ obtained from the network does not decrease, restore the weight values and divide μ by the decay rate β.

$$\mu = {}^{\mu}/_{\beta} \tag{2.17}$$

5. Repeat iterations until convergence in terms of the performance index $F(w)$ or a maximum number of iterations is reached.

2.3 Activation Function

An activation function, denoted by $f(v_k)$, defines the output of a neuron k in terms of the induced local field v_k. Here, we identify the three basic types of the activation functions:

1. **Threshold function**

   For this type of activation function, described in Fig. 2.3(a), we have

$$f(v_k) = \begin{cases} 1 & if \quad v_k \geq 0 \\ 0 & if \quad v_k < 0 \end{cases} \tag{2.18}$$

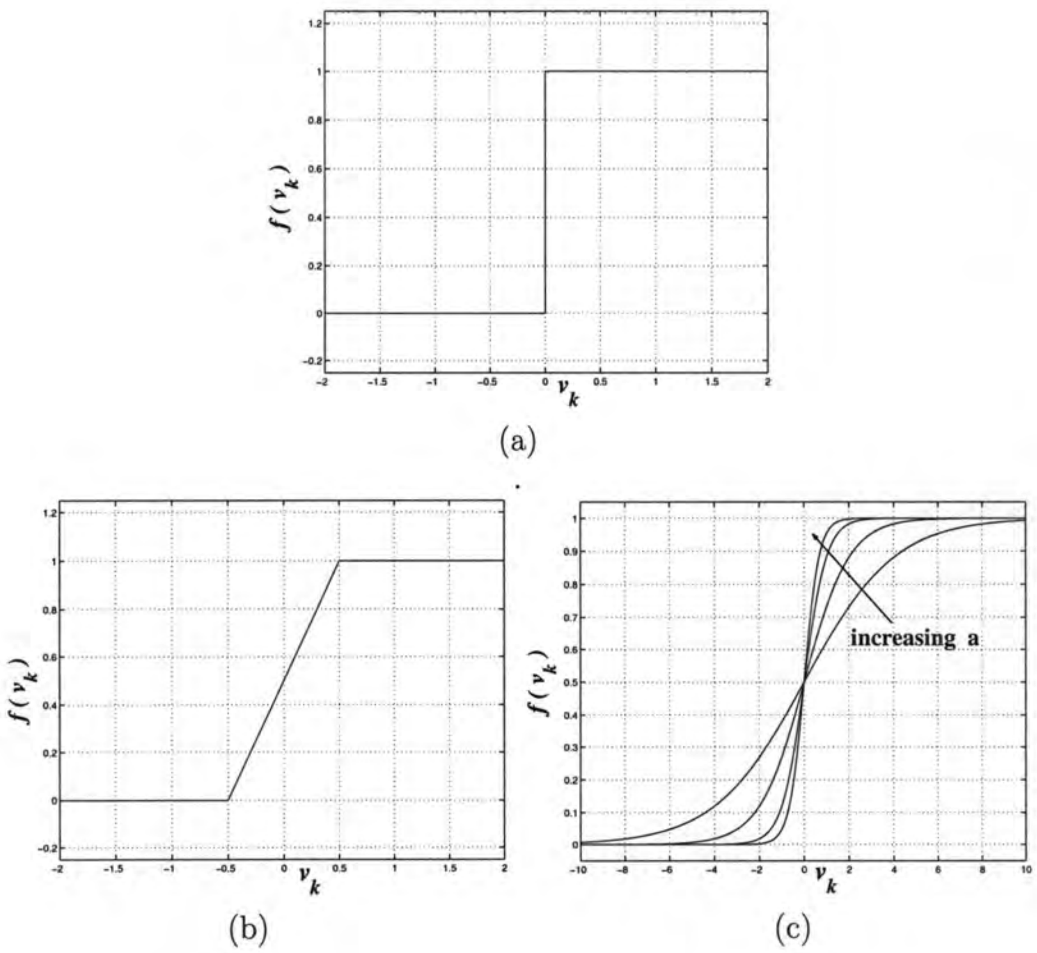

Figure 2.3: Three different types of activation Functions (a)Threshold function, (b)Piecewise-Linear Function and (c)Logistic Activation Functions.

This form of threshold function is commonly referred to as a Heaviside function. The value of a v_k is computed by

$$v_k = \sum_{j=1}^{m} w_{kj} x_j + b_k \tag{2.19}$$

where m is the dimension of input of the neuron k, w_{kj} is the synaptic weights at link j of neuron k, x_j is the input signals at link j and b_k is the bias of neuron k. Such a neuron is referred as the McCulloch-Pitts model, in recognition of the pioneering work done by McCulloch and Pitts(1943). In this model, the output of a neuron takes on the value of 1 if the induced local field of that neuron is nonnegative, and 0 otherwise.

2. **Piecewise-Linear Function**

   For the piecewise-linear function described in Fig. 2.3(b), we have

$$f(v_k) = \begin{cases} 1, & v_k \geq +\frac{1}{2} \\ v_k, & +\frac{1}{2} > v_k > -\frac{1}{2} \\ 0, & v_k \leq +\frac{1}{2} \end{cases} \tag{2.20}$$

   where the amplification factor, which is a slope of a linear region, is a constant value. This form of an activation function may be viewed as an approximation to a non-linear amplifier. Therefore, the piecewise-linear function becomes a threshold function if the amplification factor of the linear region is made infinitely large.

3. **Logistic Activation Function**

   A logistic activation function, whose graph is s-shaped, is by far the most common form of activation function used in the construction of artificial neural networks. It is defined by

$$f(v_k) = \frac{1}{1 + exp(-av_k)} \tag{2.21}$$

   where a is the slope parameter of the logistic activation function. By varying the parameter a, we obtain a logistic activation function of different slopes, as illustrated in Fig. 2.3(c). In fact, the slope at the origin equals $a/4$. When the slope parameter approaches infinity, the logistic activation function simply becomes a threshold function. Also notice that the logistic activation function is differentiable, whereas the threshold function is not.

The activation functions defined in Eqs. (2.18), Eqs. (2.20) and Eqs. (2.21) range from 0 to +1. It is sometimes desirable to have the activation function ranging from -1 to +1, in which case the activation function assumes an antisymmetric form with respect to the origin; that is, the activation function is an odd function of the induced local field. Specifically, the threshold function of Eqs. (2.18) is now defined as

$$f(v_k) = \begin{cases} 1, & if \;\; v_k > 0 \\ 0, & if \;\; v_k = 0 \\ -1, & if \;\; v_k < 0 \end{cases} \tag{2.22}$$

which is commonly referred to as a signum function. For the corresponding form of a logistic activation function, we may use a hyperbolic tangent function, defined by

$$f(v_k) = \tanh(v_k) \tag{2.23}$$

or

$$f(v_k) = \frac{2}{1 + e^{-2(v_k)}} - 1 \tag{2.24}$$

2.4 Cost Function

2.4.1 Mean Squared Error

Backpropagation is a gradient descent optimization procedure that minimizes the mean squared error between network output and the desired target of all input patterns. According to the principle of backpropagation algorithm, the learning performance is measured by using the following cost function.

$$E = \frac{1}{2P}\sum_{p=1}^{P}\sum_{h=1}^{n_s}(d_{ph} - out_{ph})^2 \tag{2.25}$$

where, P refers to the total number of training pattern
n_s refers to the number of neurons in the output layer
d_{ph} refers to the desired target of the h^{th} neuron in the output layer to the p^{th} training input
out_{ph} refers to the actual target of the h^{th} neuron in the output layer to the p^{th} training input.

2.4.2 Relative Entropy

The similarity between target and output values can also be measured by using the relative entropy or the Kullback - Leibler distance. Assume that target and output values are referred as event f and f' which are the same type characterized by their probability distributions. The measurement of similarity between event A and B is done by solving the following equation.

$$K(f||f') = \sum_{i=1}^{m} f_i \times ln\frac{f}{f'} \tag{2.26}$$

where, m refers to the number of levels of the variables
f and f'refers to two any probability distributions for a discrete random variables

The distance of relative entropy equation becomes zero if both distributions are equivalent ($f$and f'). The smaller distance of relative entropy means the more similarity between both events. On the other hand, the greater distance indicates the less similarity.

2.5 Neural Cellular Automata

Neural Cellular Automata (Markovian Process) is a massive parallel computing model which is an N-dimensional regular array of elements, and the cell grid can be a planar array with rectangular, triangular or hexagonal geometry, a $2-D$ or $3-D$ torus, a $3-D$ finite array, or a $3-D$ sequence of $2-D$ arrays. Cells are multiple input and single output processors, and characterized by an internal state variable which is sometimes not directly observable from outside the cell itself. It may have more than one connection networks with different neighborhood sizes, and its dynamical system is able to operate both in discrete and continuous time. The data and parameters for neural cellular automata are continuous values, and the model operates with more than one iteration, or so called, recurrent networks. Neural cellular automata differs from the other neural networks models in term of the locality of the connection between the units in which information is initially exchanged between the neighboring units, and then the non directly connected units communicate by passing information through other units, and finally create a global processing. From the mentioned principle, the block-scheme of a general neural cellular automata iteration is depicted by Figure 2.4.

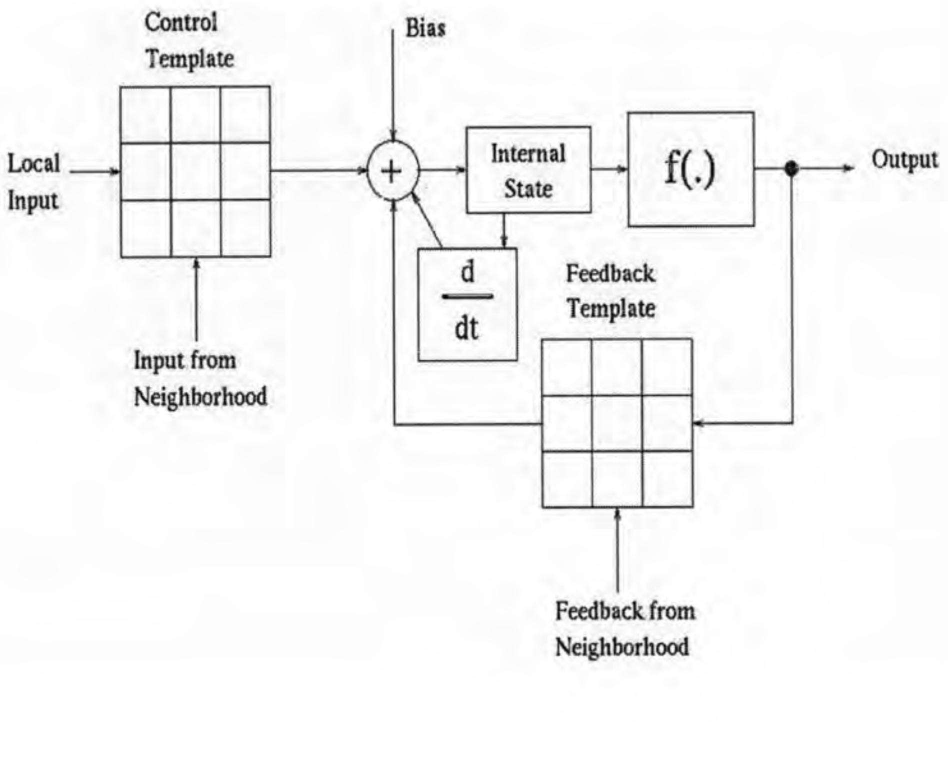

Figure 2.4: The block-scheme of a general neural cellular automata iteration.