15. INTRODUCTION

Artificial neural networks can be classified by their architectures; the main classes of models are feedback networks (as the Hopfield net), layered networks (as the multilayer perceptron), and auto-organized networks (as the Kohonen map). All these networks realize associative memory processing; the Hopfield model corresponds to an autoassociative memory, perceptrons correspond to heteroassociative memories, and Kohonen maps to classifiers, although this distinction is quite artificial. Most current models of artificial neural networks are based on one of these three architectures; many differences in the structure, in the learning rule, or in the way the data are represented can, however, exist in other models. Some combine these concepts, for example, by adding feedback to multilayer perceptrons. The diversity of all models makes it difficult to build a complete list with their full characteristics.

What is important here is to point out the common features in all these models. The first characteristic relies on the common operations involved in nets. In the Hopfield model, a matrix product is performed between the input vector (or the output of the previous computation step) and a matrix of weights. In layered nets of perceptrons, a similar matrix–vector product is performed in each layer; they differ from the Hopfield model by the absence of feedback. In Kohonen maps, the first layer is devoted to the measure of distances between the input vector and the stored ones, which is also a matrix–vector product.

The second important common aspect between these models involves the precautions that must be taken regarding the precision and the dynamics of the values involved in the operations (synaptic weights, activation levels, ...), and the dimensions and cascaddability of chips. Except for some particular networks, interesting properties of neural nets can only be found when their size (number of neurons and synapses) is large. Real parallel computing can then be performed, and the speed of the computations can become significant in comparison to classical serial computing methods. The advantages of analog VLSI, in particular situations, will be discussed in a later section, but it will be seen that the problems mentioned above, especially the cascaddability, are not so simple to handle with analog design; again, the solutions proposed for one model can easily be transposed to others, and the topological layout of the net is of less importance.

This chapter emphasizes the Hopfield model. It has been chosen for several reasons. First, historically, it was one of the most used networks for associative memories; applications were lacking for self-organizing maps, and multiple-layer nets of perceptrons suffered from the complexity of their learning algorithms. Second, the simplicity of the Hopfield model makes it easier to study compared to more complex networks. Finally, as mentioned above, VLSI constraints for the implementation of neural networks are similar in the different models; it is thus natural to experiment with a simple model, keeping in mind that the results can easily be transposed to more complex ones.

15.2. HOPFIELD NETWORK

The Hopfield model is a simple recurrent network, as shown in Fig. 15.1. Refer to Chapter 1 for operation and details of this architecture.
measured as the number of random patterns that it is possible to store without error, is then reduced to about 0.11N vectors (compared to 0.15N for the unclipped Hebb rule). Morgenstern (1987) proposed a variant of this method, the zero model, which consists of setting the weight to zero if its absolute value is lower than some threshold $Z$, and applying the sign function for the other cases. The capacity can then be raised to a mean value of 0.12N.

### 15.3.2. Projection Rule

The projection rule (generalized-inverse rule) is covered and discussed in Chapter 1 (see also Personnaz, 1985). It is given by

$$W = \Psi^\top \cdot \Psi$$

where

$$\Psi^\top = (\Psi^\top \cdot \Psi)^{-1} \cdot \Psi^\top$$

and

$$\Psi = [\xi_1, \xi_2, \ldots, \xi_n]$$

Clipping the weights obtained with Eq. (3) leads to disastrous results. The reason is the matrix inversion involved in the rule; pseudo-inverse is indeed a generalization of matrix inversion, which involves many divisions in its computation. Depending on the vectors, the results can theoretically have infinite dynamics, and all the information contained in this dynamics is completely lost with any type of truncation. Clipping the projection rule will thus not be considered here.

### 15.3.3. Optimization Rule

The projection rule guarantees that patterns $\xi^k$ are fixed points as long as these patterns are linearly independent; but the method does not speak about the degree of stability of such patterns. Basins of attraction can be represented as in Fig. 15.2 for all patterns memorized with the projection rule; circles around each pattern define the minimum basins guaranteed by the rule. It must be mentioned that Fig. 15.2 is only a schematic 2-D representation of the attractiveness of vectors; the reality would require an N-dimensional representation with vectors situated on hypercube vertices, and Hamming distances instead of 2-D Euclidian ones.

Good behavior of the algorithm is shown for patterns situated on the left part of Fig. 15.2; however, when the size of the basins of attraction guaranteed by the rule is small in comparison with the distance between patterns, the limits of an attraction domain can be close to a pattern; this is nonoptimal, as shown in the right part of Fig. 15.2.

We propose here an original algorithm (Verleysen et al., 1989) which maximizes the size of the basins of attraction for a set of given patterns (Fig. 15.3). This learning rule will be termed the “optimization rule.”

If the Hopfield network of Fig. 15.4 is examined, it can be seen that a measure of the stability of a pattern is the difference between the activation of a neuron and its threshold $\Theta$. The activation $u_i$ is defined by

$$u_i = \sum_{j} w_{ij} y_j$$

A stability factor $S$ is defined here for a particular bit $i$ in a particular pattern $\xi_i$ as

$$S_i = |u_i - \Theta|$$

Without loss of generality, $\Theta$ will be set to 0 in the following, since a threshold can be formally replaced by a supplementary fixed input.

The proposed optimization rule relies on the following intuitive view. A change in one bit of the vector $y$ (for example $-1$ instead of $+1$) can induce a change of a maximum of 2 (or $-2$) in the activation value $u_i$ (assuming that the weights $w_{ij}$ are in $[-1, +1]$). If the stability factor $S$ is constant for all bits of all stored patterns, and is set to 1 as in the projection rule, it means that one wrong bit in the pattern $y$ can lead to a wrong calculation of any other bit in the pattern during the next iteration of the network. A remedy to this problem would be to increase the stability factor $S$, again for all bits $i$ and patterns $k$. If $S$ is set to $S_i$, a maximum number of $C/2$ erroneous
The study of the algorithm is a "simulation" of its behavior under idealized conditions. The behavior is observed to be quite robust, and the results obtained from this study are considered to be valid for a wide range of practical applications.

The algorithm is based on the following assumptions:

1. The system is stable and the weights are adjusted in a consistent manner.
2. The rule for updating the weights is linearly proportional to the difference between the desired output and the actual output.
3. The system is not sensitive to small changes in the input.
4. The system is not affected by noise.

The algorithm is implemented in the following steps:

1. Initialize the weights with small random values.
2. For each input pattern, calculate the actual output and the error.
3. Update the weights using the following formula:
   
   \[ \Delta w_{ij} = \eta (d_{i} - y_{i}) x_{j} \]

   where:
   - \( \Delta w_{ij} \) is the change in the weight.
   - \( \eta \) is the learning rate.
   - \( d_{i} \) is the desired output.
   - \( y_{i} \) is the actual output.
   - \( x_{j} \) is the input.

4. Repeat steps 2 and 3 for all input patterns.

5. Repeat the entire process until the desired output is achieved or a maximum number of iterations is reached.

The algorithm is repeated for each iteration, and the weights are adjusted to minimize the error. The process is repeated until the error is minimized or a maximum number of iterations is reached. The final weights are then used to make predictions on new data.

The algorithm is a "simulation" of the behavior of a real-world system, and the results obtained from this simulation are considered to be valid for a wide range of practical applications.
15.5. CURRENT-MODE APPROACH

The property of fault-tolerant behavior of neural networks can be used to simplify some implementations. In this case, the information is represented as a function of the number of current units, with the current flowing through the network. This allows for the use of simple, low-power, and low-latency implementations. The current values can be used to represent the state of the network, and the behavior of the network can be controlled by changing the currents. The currents can be controlled by adjusting the conductance of the current sources, which can be done using simple, low-power, and low-latency techniques.

The advantage of using currents for the implementation of neural networks is that the currents can be easily adjusted and controlled, and the behavior of the network can be changed by changing the currents. The currents can be used to represent the state of the network, and the behavior of the network can be controlled by changing the currents. The currents can be controlled by adjusting the conductance of the current sources, which can be done using simple, low-power, and low-latency techniques.

The currents can be used to represent the state of the network, and the behavior of the network can be controlled by changing the currents. The currents can be controlled by adjusting the conductance of the current sources, which can be done using simple, low-power, and low-latency techniques.

15.5. Analog and Digital Implementations

The advantage of using currents for the implementation of neural networks is that the currents can be easily adjusted and controlled, and the behavior of the network can be changed by changing the currents. The currents can be used to represent the state of the network, and the behavior of the network can be controlled by changing the currents. The currents can be controlled by adjusting the conductance of the current sources, which can be done using simple, low-power, and low-latency techniques.

The currents can be used to represent the state of the network, and the behavior of the network can be controlled by changing the currents. The currents can be controlled by adjusting the conductance of the current sources, which can be done using simple, low-power, and low-latency techniques.

The currents can be used to represent the state of the network, and the behavior of the network can be controlled by changing the currents. The currents can be controlled by adjusting the conductance of the current sources, which can be done using simple, low-power, and low-latency techniques.

The currents can be used to represent the state of the network, and the behavior of the network can be controlled by changing the currents. The currents can be controlled by adjusting the conductance of the current sources, which can be done using simple, low-power, and low-latency techniques.
currents flow through N-type and P-type current sources, respectively. The mismatch between the two types of transistors makes it impossible to obtain exactly the same currents for the two states of the synapses; as a consequence, systematic errors occur in the neuron decision function, and the convergence process of the network can be altered. Mismatch between N-type and P-type current sources comes from the mobility differences between the two types of transistors. The size of the sources can of course be adjusted to compensate for these differences, but the exact ratio between the mobilities depends on the technological process, and cannot be known exactly before chip fabrication. Size compensation is thus not sufficient to ensure correct behavior of the circuit.

This mismatch problem can be avoided by using a two-line system with only one type of current source (Verleysen et al., 1989). The architecture is shown in Fig. 15.6. Its purpose is to use only one type of current source, in order to avoid the mismatching effects between N- and P-type transistors. However, even with one type of source, the currents will be slightly different from another; this will be discussed later. As described in Fig. 15.6, all excitatory currents are summed on one current line, and all inhibitory currents are summed on another one. The role of the neuron will be to compare the two total currents, and to apply the nonlinear function to the difference.

15.6.1. Synapse

Each synapse in Fig. 15.6 is a programmable current source controlled by a differential pair (see Fig. 15.7). Three connection values are allowed in each synapse. If mem0 = 1, current is sunk to one of the two lines; mem0 and the input determines to which of them the current is sunk. If mem0 = 0, no connection exists between neurons i and j, and no current flows to either the excitatory or the inhibitory line. This synapse is designed in accordance with the optimization learning rule described previously, where only 2-bit precision is necessary in the synaptic weights; the synapse input is here a logic (1-bit) value.

15.6.2. Neuron

Depending on the state of the XOR function (see Fig. 15.7), the current may be sunk either from the line current + or from the line current --. The comparison between the two total currents must be achieved in the neuron; this is done by means of the current mirror shown in Fig. 15.8. The currents on the two lines are converted into voltages across transistors T3 and T4; these voltages themselves are compared in the operational amplifier shown in the figure. To obtain digital values at the output of a neuron, its gain must be very large; the opamp will thus be a simple amplifier with no feedback, as shown in Fig. 15.9.

15.7. PRECISION IN NEURON ELEMENTS

The design of the neuron clearly determines the sum-of-products precision achieved. Since each synaptic current is relatively small, the current mirror, and the other components in the neuron, have to be designed accurately to avoid computation errors. Several mismatch errors can be quantified in the neuron of Fig. 15.9 (Verleysen et al., 1989). First, threshold voltages of the two transistors in the mirror (T3 and T4) may differ (Verleysen et al., 1991b). For first order, the effect of this difference is directly related to the transconductance of a transistor in saturated mode:

$$\Delta I_m \approx \sqrt{2 \mu C_{ox} \frac{W}{L} L V_{th}}$$

(16)

where $\Delta I_m$ is the current error, $\mu$ is the mobility of carriers in the transistors, $C_{ox}$ is their oxide capacitance, $W/L$ is their size, and $V_{th}$ is the difference of threshold voltages between T3 and T4. This leads to

$$\Delta I_m \approx \frac{2}{V_{th}} \Delta V_{th} I_1$$

(17)

where $V_{th}$ is the gate-to-source voltage of transistor T3, and $I_1$ its threshold voltage.

A second effect is the variation of $\beta$ factors between the two transistors. The resulting current difference is expressed by

$$\Delta I_e \approx \frac{\Delta \beta}{\beta} I_1$$

(18)

A third matching error in the mirror is due to the difference of the Early voltages between the two transistors. Taking the Early effect into consideration, and neglecting the Early effect of the N-type current sources, the expression for the current in the transistors is

$$I = \mu C_{ox} \frac{W}{L} \frac{V_{ds} - V_{th}}{2}$$

(19)

where $V_{ds}$ is the drain-to-source voltage of the transistor, and $V_{th}$ its pinch-off voltage. If we define

$$\lambda_p = \frac{1}{V_{th}}$$

(20)

then the current in the two transistors becomes

$$I_{th1} = \mu C_{ox} \frac{W}{L} \frac{V_{ds1} - V_{th}}{2} [1 - \lambda_p (V_{ds1} - V_{th})]$$

(21)

$$I_{th2} = \mu C_{ox} \frac{W}{L} \frac{V_{ds2} - V_{th}}{2} [1 - \lambda_p (V_{ds2} - V_{th})]$$

(22)

Equation (22) leads to

$$V_{ds2} = \frac{1}{\lambda_p} \left( 1 + \frac{2 I_{th2}}{\mu C_{ox} \frac{W}{L} (V_{ds1} - V_{th})^2} \right)$$

(23)

As proposed by Hoekstra (1990), the derivative of $V_{ds1}$ can be computed by differentiating (21) implicitly with respect to $\lambda_p$. This leads to

$$0 = \frac{\partial V_{ds1}}{\partial \lambda_p} [2 - \lambda_p (V_{ds1} - V_{th})] - (V_{ds1} - V_{th}) V_{ds1}$$

(24)

$$\frac{\partial V_{ds1}}{\partial \lambda_p} = \frac{(V_{ds1} - V_{th}) V_{ds1}}{2 - \lambda_p (V_{ds1} - V_{th})}$$

(25)

Differentiating $V_{ds2}$ in (23) with respect to $\lambda_p$ results in

$$\frac{\partial V_{ds2}}{\partial \lambda_p} = \frac{V_{ds2}}{\lambda_p}$$

(26)
requirements of VLSI circuits. The basic principles of analog VLSI architectures are presented, together with most precision limitations encountered with such technology. It is also shown how to overcome such limitations by appropriate design of the neuron.

The principles explained in this chapter are illustrated for the Hopfield memory network. They are, however, more general, and can be used in most neural network implementations where the basic operation is the sum-of-product (i.e., the product of a matrix and a vector), and where binary weights may be used.

This chapter also presents a learning rule for the Hopfield memory adapted to such circuits; the rule shows good performance even when the weights’ dynamic range is restricted to one or two bits.

REFERENCES


