Ref: Abdi, H. (1994). A neural Network Primer. *Journal of Biological System*, **2(3)**, 247–283.

A NEURAL NETWORK PRIMER

HERVÉ ABDI

School of Human Development: The University of Texas at Dallas,
MS: GR.4.1., Richardson, TX75083-0688, U.S.A.
&
Université de Bourgogne at Dijon, 21000, Dijon, France

Received July 1993
Revised March 1994

ABSTRACT

Neural networks are composed of basic units somewhat analogous to neurons. These units are linked to each other by connections whose strength is modifiable as a result of a learning process or algorithm. Each of these units integrates independently (*in parallel*) the information provided by its synapses in order to evaluate its state of activation. The unit response is then a linear or nonlinear function of its activation. Linear algebra concepts are used, in general, to analyze linear units, with eigenvectors and eigenvalues being the core concepts involved. This analysis makes clear the strong similarity between linear neural networks and the general linear model developed by statisticians. The linear models presented here are the *perceptron*, and the *linear associator*. The behavior of nonlinear networks can be described within the framework of optimization and approximation techniques with dynamical systems (*e.g.*, like those used to model spin glasses). One of the main notions used with nonlinear unit networks is the notion of *attractor*. When the task of the network is to associate a response with some specific input patterns, the most popular nonlinear technique consists of using hidden layers of neurons trained with back-propagation of error. The nonlinear models presented are the *Hopfield* network, the *Boltzmann machine*, the *back-propagation network*, and the *radial basis function* network.

Keywords: neural networks, general linear model, perceptron, radial basis function, Hopfield network, Boltzmann machine, back-propagation network, eigenvector, eigenvalue, principal component analysis, attractors, optimization.

Even though research in neural modeling started *circa* 1940 (*i.e.*, McCulloch & Pitts in 1943 [48]), there was little active development of the field prior to the late fifties and early sixties when Rosenblatt introduced the *perceptron* in 1958 [69], (a close cousin of the perceptron was Widrow's *adaline* introduced in 1960 [84]). These early models already posess most of the

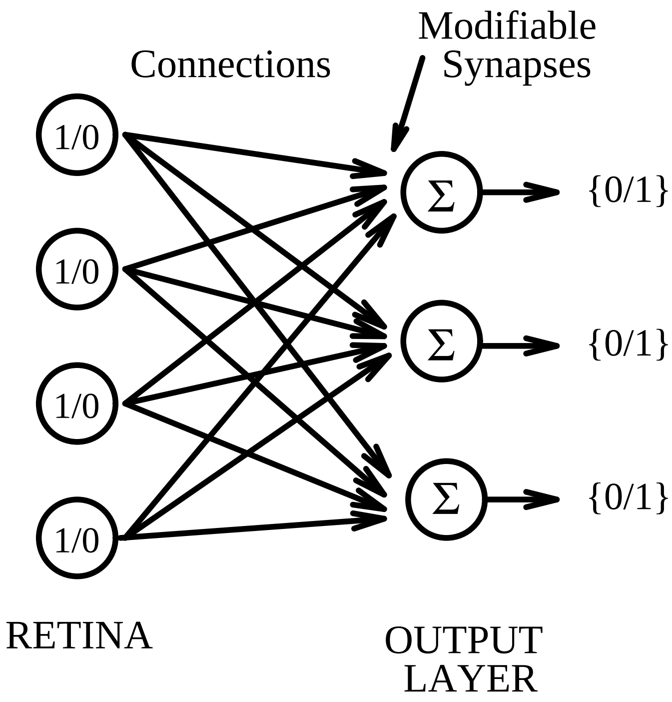

FIG. 1. — The architecture of a typical perceptron. The perceptron is composed of two layers of cells connected by synapses modifiable through learning. The input layer is named the *retina*. The cells can take only binary values (*e.g.*, 0 or 1).

essential features of more contemporary neural networks. They are composed of simple basic units loosely comparable to neurons. Perceptrons have essentially two layers of cells: an input layer which was called the (artificial) *retina* of the perceptron, and an output layer. Learning in these networks takes place at the synaptic junctions between the neurons of the input layer and the neurons of the output layer (in the original paper the output layer was called the "association layer"). At first, the performance of these early networks attracted quite a lot of attention. However, their limitations soon became clear.

Rosenblatt [70] and Minsky and Papert [53] showed that these earlier neural networks were able to learn associations between a set of inputs and a set of outputs only if the output is a linear transformation of the input. Essentially, these networks are equivalent to linear regression and to discriminant analysis. As a consequence, their use as a *model* for human behavior was of moderate appeal. In addition, their computational power was only mediocre when compared to the digital computers available at the time. Considering the earlier successes of the symbolic approach (*i.e.*, Artificial Intelligence), it was tempting, then, to concentrate resources on the symbolic approach rather than on neural networks. This is, indeed, what happened (for a more detailed history of the subject, see [10, 3]). Despite the shortcomings of these early neural network models, they can solve important problems, and have the additional advantage of simplicity. As a consequence, I will use them in this paper to introduce neural networks and some of their main characteristics.

The late seventies and early eighties witnessed a renaissance of neural networks. The many reasons for this resurgence include: 1.) the general disappointment with the performance of the symbolic approach; 2.) the availability of cheap but powerful (micro) computers; 3.) the

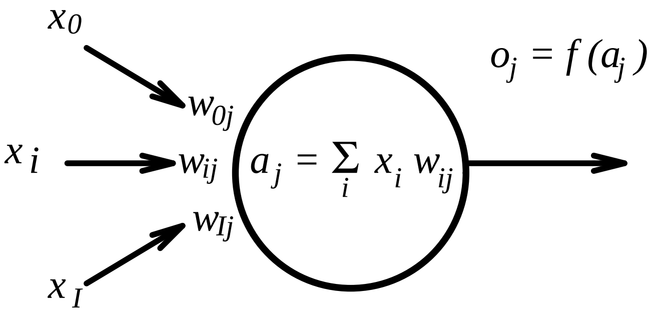

FIG. 2. — The output cell of a perceptron. The inputs are noted x_i, the synaptic weights are noted $w_{i,j}$, the total activation of the cell is noted a_j, its response or output, noted o_j, is equal to 0 if $a_j \leq 0$ and 1 if $a_j > 0$.

development of nonlinear models of neural networks; and 4.) the (re)discovery of techniques for training hidden layers of neurons.

The purpose of this introductory paper is to present some representative members of the main families of neural networks. I begin with the perceptron, which can be used to introduce the general architecture of a neural net. In the years since its introduction, several alternative architectures have been proposed to overcome the limitations of the perceptron. I describe two of these alternatives under the heading of nonlinear networks: the very popular back-propagation network, and the radial basis function network, a relative newcomer to the field of neural networks that is used by several papers in this volume. Most neural networks can be interpreted as associators: their task is to associate a given input pattern with a given response. A particular case of associators, called *auto-associators*, occurs when the input pattern is the same pattern as the response. In addition to the linear auto-associator, two nonlinear associators are described in this paper: the Hopfield network and the Boltzmann machine.

1. The Perceptron

The perceptron can be considered as the first important implementation of artificial neural networks. It was created in the fifties by Rosenblatt [68, 69]. As its name indicates, the perceptron was designed to mimic or to model perceptual activities. The main goal was to associate binary configurations (*i.e.*, patterns of $[0, 1]$ values) presented as inputs on a (artificial) retina with specific binary outputs. Hence, esssentially, the perceptron is made of two layers (see Figure 1): the input layer (*i.e.*, the "retina") and the output layer. The architecture, originally designed by Rosenblatt in 1957 [68], was more complex than this, but in fact is equivalent to the two-layer description given here (cf. also [3, 9, 10]). The activation of the cells of the input layer is transmitted to the cells of the output layer through connections ("synapses") which can change the intensity of the signal (by multiplying the incoming signal by a "weight" or "synaptic efficacy", see Figure 2). Hence, the response of the perceptron is a function of the stimulus applied to the cells of the input layer and of the weights of the connections. The cells of the output layer compute their state of activation as a function of the stimulation they receive through the synapses, and then give a (binary) response as a function of their state of activation.

More formally, the response of the output cells depends upon their level of activation which is computed as the sum of the weights coming from active input cells. The response is then obtained by thresholding the activation (*i.e.*, the cell will be in the active state only if its activation level is larger than a given threshold).

Specifically, the activation of the j-th output cell is computed as

$$a_j = \sum_i^I x_i w_{i,j} \ , \tag{1.1}$$

with:

- a_j: activation of the j-th output cell.
- x_i: state of the i-th cell of the retina (0 or 1).
- $w_{i,j}$: value of the weight connecting the i-th cell of the retina to the j-th output cell.

The output cells will then take either the active state (*i.e.*, give the response 1) or the inactive state (*i.e.*, give the response 0) if their level of activation is greater or less than their threshold noted ϑ_j (quite often ϑ_j is set to 0). Precisely, the response of the j-th output cell is given as

$$o_j = \begin{cases} 0 & \text{for } a_j \le \vartheta_j \\ 1 & \text{for } a_j > \vartheta_j \end{cases} \ . \tag{1.2}$$

The threshold ϑ_j is modifiable by learning (like the weights), as a consequence its function is almost equivalent to a weight. Actually, thresholding is often implemented by setting an input cell always active (the 0-th input cell) so that the weight $w_{0,j}$ is equal to $-\vartheta_j$. The term of *response bias* is often used as a synonym for threshold. This works because Equation 1.2 is then rewritten as (with $w_{0,j}$ being equal to $-\vartheta_j$):

$$o_j = \begin{cases} 0 & \text{for } a_j + w_{0,j} \leq 0 \\ 1 & \text{for } a_j + w_{0,j} > 0 \end{cases} . \tag{1.3}$$

Equivalently, ϑ_j can be computed as $w_{0,j}$ if cell 0 is "clamped" to the value -1.

1.1. Learning rule

The main problem for the perceptron is to learn how to adjust the synaptic weights in order to give the desired response for a given stimulus. For example, a perceptron with two input cells and one output cell can be used to implement logical functions if the stimuli given as input consist of the set

$$\begin{array}{cc} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{array}$$

and the output is constituted by the 4 binary responses corresponding to the function to be implemented. For example, the logical OR function is given by this pattern of association

$$\begin{array}{ccccc} 0 & 0 & \longmapsto & 0 \\ 1 & 0 & \longmapsto & 1 \\ 0 & 1 & \longmapsto & 1 \\ 1 & 1 & \longmapsto & 1 \, . \end{array} \tag{1.4}$$

Learning in perceptrons takes place at the synaptic level by changing the weights of the connections between the cells of the retina and the output cells. There are several possible procedures that a perceptron can use in order to change iteratively the synaptic weights to produce a set of desired responses to a particular set of inputs. The most famous one is known under several names: the *Widrow-Hoff* learning rule (cf. [84, 75, 56, 21]), the *Delta* rule [71], or more simply the *perceptron* learning rule [30].

In order to use the Widrow-Hoff learning rule, the output cells of the perceptron must be provided with the correct response so that they can compute an *error signal*. This type of learning is referred to as *supervised learning*. The output cells must also know the state of the input cells (*i.e.*, they need to know what synapses are activated by the current stimulation). The output cells, however, do not need to know the response of the other output cells. Hence this learning rule can be seen as purely local, because the cells need to know only the local information in order to learn.

For a perceptron, the Widrow-Hoff learning rule is very simple. First, a cell will learn only if it makes a mistake (*i.e.*, the error signal is not zero). Because the output is binary, there are only two possible mistakes. Either the cell should be *off* and it is *on*, or the cell is *off* when it should be *on*. If the cell is *on* instead of *off*, this means that the cell's activation is too high, and therefore, that it has assigned too much importance (*i.e.*, too large a weight) to the cells from the retina that are in the *on* state. An obvious solution to this problem is to decrease the weights associated with these cells from the retina. Likewise, if the cell is *off* when it should be *on*, its activation is too low, and so, the weights of the retinal cells that are *on* should be increased to correct the problem.

Learning for a set of stimuli is implemented by "presenting" (*i.e.*, computing the output associated with a given input pattern) the individual stimuli to the perceptron several times in a random order. The weights are adjusted as described previously and the learning procedure terminates as soon as the perceptron makes no error. The synaptic weights will stop changing as soon as the perceptron performs perfectly, because the perceptron learns only when making a mistake.

More formally, the Widrow-Hoff learning rule is written as:

$$w_{i,j}^{(t+1)} = w_{i,j}^{(t)} + \eta(t_j - o_j)x_i = w_{i,j}^{(t)} + \Delta w_{i,j} \ , \qquad \mathbf{(1.5)}$$

with

- $\Delta w_{i,j}$: correction to add to the weight $w_{i,j}$.
- x_i: value (0 or 1) of the i-th retinal cell.
- o_j: response or output of the j-th output cell.
- t_j: target response (or correct desired response).
- $w_{i,j}^{(t)}$: weight of the synapse between the i-th retinal cell and the j-th output cell at time (t) [*i.e.*, the exponent (t) give the number of the current iteration]. The values of $w_{i,j}^{(0)}$ are generally initialized to small random values.
- η: a small positive constant generally referred to as the *learning constant*. Its function is examined in greater detail in the discussion of linear associative memory. It should be noted, however, that choosing η is a delicate problem in training a neural network. In several cases, the value of η is dependent upon the learning history. Learning will start with relatively large values of η which will be decreased as learning progresses. So, at first the system makes important corrections to the synaptic weights. When the values of η get relatively small this amounts to making fine changes to the synaptic weights.

1.2. An example: learning the logical function OR

The purpose of this section is to provide a demonstration of the ability of the perceptron to resolve a linearly separable function. Let's suppose that we want to teach a perceptron the logical function OR, whose truth table is:

$$\begin{array}{ccccc} 0 & 0 & \longmapsto & 0 \\ 1 & 0 & \longmapsto & 1 \\ 0 & 1 & \longmapsto & 1 \\ 1 & 1 & \longmapsto & 1\,. \end{array}$$

This perceptron is composed of three input cells: two cells for the values of the argument of the logical function, plus one cell (*i.e.*, x_0) to implement thresholding (cf. Equation 1.3), and one output cell (see Figure 2). This is equivalent to implementing the following ternary truth table:

$$\begin{array}{ccccc} 1 & 0 & 0 & \longmapsto & 0 \\ 1 & 1 & 0 & \longmapsto & 1 \\ 1 & 0 & 1 & \longmapsto & 1 \\ 1 & 1 & 1 & \longmapsto & 1\,. \end{array}$$

Supposing that the synaptic weights (*i.e.*, the w_i's) are initialized to zero values, they can be represented by a 3×1 matrix $\mathbf{W}$:

$$\mathbf{W} = \begin{bmatrix} w_0 = -\vartheta \\ w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} . \qquad \mathbf{(1.6)}$$

Suppose, now, that the first randomly chosen association is the 4-th one (*i.e.*, the perceptron should produce the value 1 when its retinal cells are presented with the stimulus [1, 1, 1]). The activation of the output cell is given by:

$$a = \sum_i x_i w_i = (1 \times 0) + (1 \times 0) + (1 \times 0) = 0 \qquad \mathbf{(1.7)}$$

(because there is just one output cell in this example, the notation is somewhat simpler, for example a_j becomes a, $w_{i,j}$ becomes w_i, and so on). The (incorrect) response of the perceptron is

$$o = 0$$

(cf. Equation 1.2).

The error is the difference between the target value (1) and the response of the output cell (0). Suppose that the learning constant η is set to the value $\eta = .1$, and the Widrow-Hoff learning rule is used. Then, from Equation 1.5, the correction for the synapses from the retinal cells to the output cell is:

$$\begin{aligned} \Delta w_0 &= \eta(t-o)x_0 = .1 \times (1-0) \times 1 = .1 \\ \Delta w_1 &= \eta(t-o)x_1 = .1 \times (1-0) \times 1 = .1 \\ \Delta w_2 &= \eta(t-o)x_2 = .1 \times (1-0) \times 1 = .1 \ . \end{aligned} \tag{1.8}$$

When the correction is applied, $\mathbf{W}$ becomes:

$$\begin{bmatrix} w_0 = .1 \\ w_1 = .1 \\ w_2 = .1 \end{bmatrix} \ . \tag{1.9}$$

Suppose, now, that the first stimulus (*i.e.*, [1, 0, 0]) is presented to the perceptron. The activation of the output cell is given by:

$$a = \sum_i x_i w_i = (1 \times .1) + (0 \times .1) + (0 \times .1) = .1 \ , \tag{1.10}$$

and the response of the output cell is

$$o = 1 \ .$$

The correction to apply to the synaptic weights is

$$\begin{aligned} \Delta w_0 &= \eta(t-o)x_0 = .1 \times (0-1) \times 1 = -.1 \\ \Delta w_1 &= \eta(t-o)x_1 = .1 \times (0-1) \times 0 = 0 \\ \Delta w_2 &= \eta(t-o)x_2 = .1 \times (0-1) \times 0 = 0 \end{aligned} \tag{1.11}$$

When the error correction rule is applied, $\mathbf{W}$ becomes:

$$\begin{bmatrix} w_0 = 0 \\ w_1 = .1 \\ w_2 = .1 \end{bmatrix} \ . \tag{1.12}$$

With this set of weights, the perceptron is now able to give perfect answers for the OR problem.

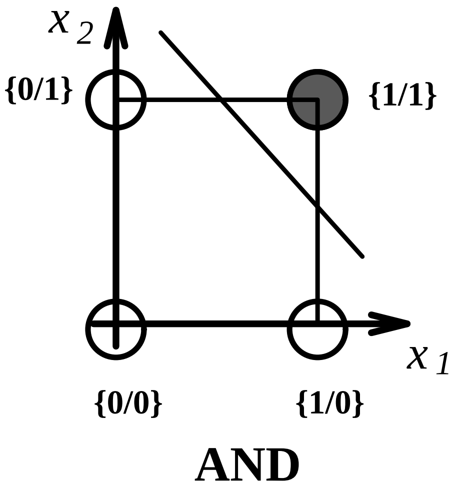

FIG. 3. — The logical function AND is linearly separable. It is possible to draw a line separating the white circles from the dark ones.

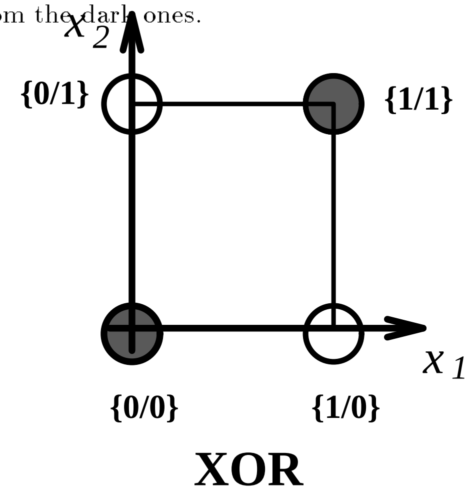

FIG. 4. — The logical function XOR is not linearly separable. There is no way to draw a line separating the white circles from the dark ones.

1.3. Evaluation of the perceptron

As the previous example makes clear, the perceptron is able to learn. The question addressed in this section is "What is it able to learn?" Because the activation of the output cell is a linear combination of the retinal input cells, the perceptron can learn only to discriminate linearly separable categories (this is illustrated in Figure 3). If the categories are linearly separable, then if the learning constant η is small enough, convergence is guaranteed (cf. [3, 12] for a "modern" proof of the perceptron convergence theorem).

As Minsky and Papert showed in their famous book [53], it is quite easy to find examples of interesting nonlinearly separable functions. Probably the most well known of these is the XOR function (the "exclusive or"). This problem is illustrated in Figure 4, and corresponds to the

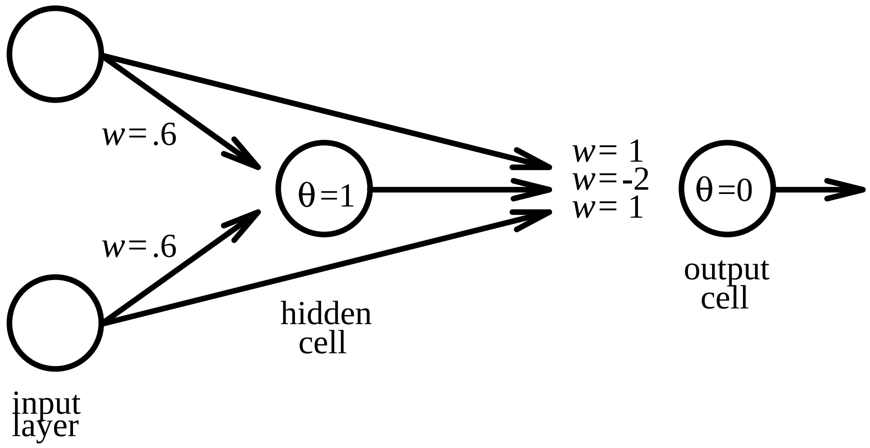

FIG. 5. — A perceptron with a hidden layer is able to solve the XOR problem.

following truth table:

$$\begin{array}{cccc} 0 & 0 & \longmapsto & 0 \\ 1 & 0 & \longmapsto & 1 \\ 0 & 1 & \longmapsto & 1 \\ 1 & 1 & \longmapsto & 0\ . \end{array} \tag{1.13}$$

In order to show that the logical function XOR is not linearly separable, suppose that we have a perceptron with 2 retinal cells and one output cell. The synaptic weights are noted w_1 and w_2 (to make the argument easy to follow, the threshold is supposed, without loss of generality, to be 0). The association of the input pattern [1, 0] to the response 1 implies that

$$w_1 > 0\ . \tag{1.14}$$

Similarily, the association of the input pattern [0, 1] to the response 1 implies that

$$w_2 > 0\ . \tag{1.15}$$

Adding together Equations 1.14 and 1.15 gives

$$w_1 + w_2 > 0\ . \tag{1.16}$$

Now if the perceptron gives the response 0 to the input pattern [1, 1], this implies that

$$w_1 + w_2 \leq 0\ . \tag{1.17}$$

Clearly, the last two equations contradict each other, hence no set of weights can solve the XOR problem.

1.4. Linearly separable or not? Is it really a problem?

In the early days of neural networks, much was made of the perceptron's inability to solve such a seemingly simple nonlinear transformation as the XOR problem. More recently, however, several algorithms (the most popular of which is *back-propagation*) make it possible for a perceptron with at least one hidden layer to implement the XOR logical function. This is illustrated in Figure 5, and in Table 1. However, in the years following Rosenblatt's introduction of the perceptron, there were no algorithms available *to train* the hidden layer. In fact, Minsky and Papert [53], mistakenly believed that no such algorithm could exist.

Activation of the Cells				
input		Activation of the hidden cell ($\vartheta = 1$)	o	Activation of the output cell ($\vartheta = 0$)
0	0	$(0 \times .6) + (0 \times .6) = 0.0$	0	$(0 \times 1) + (0 \times 1) + (0 \times -2) = 0$
0	1	$(0 \times .6) + (1 \times .6) = 0.6$	0	$(0 \times 1) + (1 \times 1) + (0 \times -2) = 1$
1	0	$(1 \times .6) + (0 \times .6) = 0.6$	0	$(1 \times 1) + (0 \times 1) + (0 \times -2) = 1$
1	1	$(1 \times .6) + (1 \times .6) = 1.2$	1	$(1 \times 1) + (1 \times 1) + (1 \times -2) = 0$

(o is the response of the hidden cell after "thresholding")

TABLE 1:

Responses of the cells of the perceptron described in Figure 5 showing how to solve the XOR problem

In addition to training a perceptron with hidden layers to make nonlinear transformations between input and output stimuli, another simpler way of coping with a nonlinearly separable problem is to recode it in order to make it linearly separable. For example, the XOR problem can be solved if a third input cell is added and provided with the product of the two remaining cells, transforming learning the binary XOR into learning the ternary relation:

$$\begin{array}{ccccc} 0 & 0 & 0 & \longmapsto & 0 \\ 1 & 0 & 0 & \longmapsto & 1 \\ 0 & 1 & 0 & \longmapsto & 1 \\ 1 & 1 & 1 & \longmapsto & 0\,. \end{array} \tag{1.18}$$

The following set of weights:

$$w_1 = 1, \qquad w_2 = 1, \qquad w_3 = -2 \tag{1.19}$$

will then solve the XOR problem.

It has often been claimed that Minsky and Papert's [53] highly critical analysis of the perceptron's failure to cope with nonlinearly separable categorizations was responsible for supressing interest in neural network research until the late seventies. As Anderson and Rosenfeld [10] point out in their introductory comments to some classic papers in neural modeling, this claim is somewhat exaggerated. Several other factors that may account for the lack of interest in neural networks include: the relatively clumsy nature of the networks, and the fact that the numerical algorithms they applied were better implemented and analyzed as numerical techniques on standard computers. Also, the *Zeitgeist* at that time favored modeling memory storage as creating new proteins rather than synaptic modification (because of the success of DNA/RNA for the genetic code). So, at best (or at worst) Minsky and Papert gave the final blow to an already moribund field.

2. Linear associators

The models presented in this section are known as linear associators. They come in two forms: hetero- and auto-associators. The hetero-associators can be used to learn associations between input and output patterns. The auto-associator is a special case of the hetero-associator in which the association between an input pattern and itself is learned. This special case of associator is used widely as a pattern recognition and pattern completion devices in an auto-associator is able to reconstruct learned patterns when noisy or incomplete versions of the learned patterns are used as "memory keys" [41, 42] (this property is true also for non-linear auto-associators, the linear asociator has, in addition, the advantage of being easy to analyze mathematically as well as being very efficient computationally).

In what follows, matrix notation is used. A good introduction oriented toward neural modeling can be found in [36] and [3].

2.1. Notation

Stimuli are represented by $I \times 1$ vectors $\mathbf{f}_k$ where the index k indicates the stimulus number. The components of $\mathbf{f}_k$ specify the stimulus to be applied to the cells of the input layer for the k-th stimulus. In general, input vectors are normalized so that $\mathbf{f}_k^T \mathbf{f}_k = 1$. The responses of the network are given by $J \times 1$ vectors denoted $\mathbf{g}_k$. The complete set of K stimuli is represented by a $K \times I$ matrix denoted $\mathbf{F}$ in which the k-th row is $\mathbf{f}_k^T$. The set of the K responses is represented by a $K \times J$ matrix denoted $\mathbf{G}$ in which the k-th row is $\mathbf{g}_k^T$. The $J \times I$ synaptic weight matrix is denoted $\mathbf{W}$.

The Hebbian learning rule [29] in linear associators sets the change in the synaptic weights to be proportional to the product of the input and the output. For example, if the association considered is between the k-th input and the k-th response, the weight $w_{i,j}$ corresponding to the connection between the i-th input cell and the j-th output cell should be proportional to $f_{k,i} \times g_{k,j}$. If the proportionality constant is set to 1, the association between the k-th stimulus and the k-th response leads to the creation of the weight matrix

$$\mathbf{W}_k = \mathbf{g}_k \mathbf{f}_k^T \ . \tag{2.1}$$

The response of the network is obtained by postmultiplication of $\mathbf{W}_k$ by the stimulus $\mathbf{f}_k$. The response of the associator is denoted $\hat{\mathbf{g}}_k$ and can be considered as the associator estimation of $\mathbf{g}_k$. With just one association stored, it is easy to show that the associator estimates $\mathbf{g}_k$ perfectly:

$$\hat{\mathbf{g}}_k = \mathbf{W}_k \mathbf{f}_k = \mathbf{g}_k \mathbf{f}_k^T \mathbf{f}_k = \mathbf{g}_k \tag{2.2}$$

To implement several associations, the $\mathbf{W}_k$ matrices are summed in order to give $\mathbf{W}$:

$$\mathbf{W} = \sum \mathbf{W}_k = \sum \mathbf{g}_k \mathbf{f}_k^T = \mathbf{G}^T \mathbf{F} \ . \tag{2.3}$$

The estimation of the k-th response by the associator is obtained as:

$$\hat{\mathbf{g}}_k = \mathbf{W} \mathbf{f}_k = \sum_{\ell=1}^{K} \mathbf{g}_\ell \mathbf{f}_\ell^T \mathbf{f}_k = \mathbf{g}_k + \sum_{\ell \neq k} \cos(\mathbf{f}_\ell, \mathbf{f}_k) \mathbf{g}_\ell \ . \tag{2.4}$$

From this last equation, it is clear that the response of the associator involves some cross-talk or interference between the stored patterns that may result in less than perfect recall of the learned associations. In general, the quality of the associator estimation is evaluated by comparing $\hat{\mathbf{g}}_k$ with $\mathbf{g}_k$. A popular estimator is the cosine between $\hat{\mathbf{g}}_k$ and $\mathbf{g}_k$: $\cos(\hat{\mathbf{g}}_k, \mathbf{g}_k)$. However, if the stimuli are pairwise orthogonal (*i.e.*, if $\mathbf{f}_k^T \mathbf{f}_{k'} = 0$, for all $k \neq k'$) then $\cos(\mathbf{f}_k, \mathbf{f}_{k'}) = 0$ and $\hat{\mathbf{g}}_k = \mathbf{g}_k$, and consequently, the responses of the associator estimate perfectly the target responses.

2.1.1. Linear auto-associator

As noted, the *linear auto-associator* (cf., Valentin *et al.* in this volume) is a particular case of the linear-associator. The goal of this network is to associate a set of stimuli to itself, (*i.e.*, $\mathbf{G}$ is equal to $\mathbf{F}$). The weight matrix $\mathbf{W}$ is now equal to $\mathbf{F}^T\mathbf{F}$, and is the familiar "cross-product" matrix of standard linear multivariate analysis.

When the stimulus set is composed of non-orthogonal stimuli, the associator will fail to reconstruct perfectly the stimuli that were stored (cf. Equation 2.4). On the other hand, some new patterns will be perfectly reconstructed by the associator, creating in a way, the equivalent of a "false alarm" or "false recognition." These patterns are defined by the equation:

$$\mathbf{W}\mathbf{u}_k = \lambda_k \mathbf{u}_k \quad \text{with: } \mathbf{u}_k^T \mathbf{u}_k = 1 \ . \tag{2.5}$$

They are the *eigenvectors* of $\mathbf{W}$ and λ_k is the eigenvalue associated with the k-th eigenvector. Within a multivariate analysis framework, the eigenvectors of $\mathbf{W}$ are the principal components of a Q-principal component analysis of the stimuli [38]. In general, principal component analysis is used to analyze the variables describing the stimuli, Q-analysis is obtained by transposing the data matrix which is equivalent to switching the *rôle* of the variables and the stimuli. These eigenvectors can be interpreted as *prototypes*, *macro-characteristics*, or, even, *hidden dimensions* [2, 8, 11, 57, 58].

2.1.2. Back to the general case

In order to improve the performance of linear associators (*i.e.*, in order to increase the cosine between the $\hat{\mathbf{g}}_k$'s and $\mathbf{g}_k$'s), several learning rules have been proposed. The most popular one is clearly the Widrow-Hoff learning rule (*alias* Delta learning rule) previously described for the perceptron. This is an iterative procedure correcting the error between the target response and the actual response of the network. In matrix notation, the Widrow-Hoff rule is:

$$\mathbf{W}_{(t+1)} = \mathbf{W}_{(t)} + \eta(\mathbf{g}_k - \mathbf{W}_{(t)}\mathbf{f}_k)\mathbf{f}_k^T \;, \tag{2.6}$$

with $\mathbf{W}_{(t)}$ being the weight matrix at step t, η being a (small) positive constant, and k being randomly chosen.

This algorithm will (eventually) converge if η is properly chosen. Actually, the analysis of the Widrow-Hoff learning rule involves only the matrix of eigenvectors of $\mathbf{F}^T\mathbf{F}$ denoted $\mathbf{U}$, the diagonal matrix of their eigenvalues denoted $\mathbf{\Lambda}$, and the matrix of eigenvectors of $\mathbf{F}\mathbf{F}^T$ denoted $\mathbf{V}$, as well as η and t (cf. [3]). The analysis is facilitated by using the *singular value decomposition* of $\mathbf{F}$ which expresses the matrix $\mathbf{F}$ as:

$$\mathbf{F} = \mathbf{V}\mathbf{\Lambda}^{\frac{1}{2}}\mathbf{U}^T \;. \tag{2.7}$$

With these notations, $\mathbf{W}_{(t)}$ can be computed as:

$$\mathbf{W}_{(t)} = \mathbf{G}\mathbf{U}\left\{\mathbf{\Lambda}^{-\frac{1}{2}}\left[\mathbf{I} - (\mathbf{I} - \eta\mathbf{\Lambda})^t\right]\right\}\mathbf{V}^T \;. \tag{2.8}$$

From Equation 2.8, it can be seen that convergence is assured if

$$0 < \eta < 2\lambda_{\max}^{-1} \qquad \text{(with } \lambda_{\max} \text{ being the largest eigenvalue of } \mathbf{\Lambda}). \tag{2.9}$$

When convergence is reached, the weight matrix is then (cf. [42]):

$$\widetilde{\mathbf{W}} = \mathbf{G}\mathbf{F}^{+} \tag{2.10}$$

with:

$$\mathbf{F}^{+} = \mathbf{V}\mathbf{\Lambda}^{-\frac{1}{2}}\mathbf{U}^T \tag{2.11}$$

denoting the Moore-Penrose pseudo inverse of $\mathbf{F}$.

For an auto-associative memory, the Widrow-Hoff learning rule will converge to:

$$\widetilde{\mathbf{W}} = \mathbf{F}\mathbf{F}^{+} = \mathbf{U}\mathbf{U}^T \quad \text{with: } \mathbf{U} \text{ being the matrix of eigenvectors of } \mathbf{W}. \tag{2.12}$$

The matrix $\mathbf{W}$ is said to have been *sphericized* (*i.e.*, all its non-zero eigenvalues are now equal to one).

3. Non-linear systems (auto-associator)

As its name indicates, a linear associator (cf. also Equation 2.4) will give a response that is a linear combination of *all* the input values. This property can be undesirable sometimes. As with the perceptron, several nonlinear extensions of the auto-associator that can remedy this problem have been developed. These models include the Hopfield net [33, 34], and Anderson's *Brain-State-in-a-Box* (abreviated as BSB, [8]). While these models originated within very different traditions, they are in fact quite similar [27]. Introduced from a physics perspective, the popularity of the Hopfield model, was perhaps one impetus of the current resurgence of interest in neural networks [8, 9]. Acting as content-addressable memories, these nonlinear auto-associators try to find, among the patterns stored, the one *closest* to the stimulus. In the present paper, I will describe Hopfield nets in some detail as an example of a nonlinear auto-associator. It is worth noting, however, that the BSB model, published prior to the Hopfield model, was proposed as a psychological model of memory and shares many of the essential features of the Hopfield model. Readers interested in this perspective on neural networks will find a detailed presentation in the Anderson and Rosenfeld collection of papers [8].

3.1. Hopfield Nets

The Hopfield net is an asynchronous nonlinear auto-associator. This network is named for J.J. Hopfield (a physicist from CALTECH) who gave a detailed analysis of these types of networks in two papers which have since become classic papers ([33, 34]). The k-th stimulus to be stored is represented by an $I \times 1$ binary vector denoted $\mathbf{f}_k$, taking values 0 or 1. The set of the thresholds for the I units is denoted $\boldsymbol{\vartheta} = [\vartheta_i]$, with ϑ_i being the threshold of the i-th unit. From the $\mathbf{f}_k$ vectors, new "recoded" vectors denoted $\mathbf{h}_k$ are created:

$$\mathbf{h}_k = 2\mathbf{f}_k - \mathbf{1}_{I\times 1} \qquad (\textit{i.e.}, \text{ the 0 values are replaced by } -1.) \tag{3.1}$$

with $\mathbf{1}_{I\times 1}$ being an $I \times 1$ vector of ones.

The weight matrix is obtained as

$$\mathbf{W} = \sum_k \mathbf{h}_k \mathbf{h}_k^T \ . \tag{3.2}$$

A stimulus is "recalled" by presenting a "cue" to the matrix, and letting the memory stabilize to an estimation. More formally, when $\mathbf{f}_k$ is used as a cue, recall from the memory is obtained through the following steps:

0. *Initialization.* Let $t = 0$ (t is the number of the current iteration). Set $\bar{\mathbf{f}}_k^{(t)} = \bar{\mathbf{f}}_k^{(0)} = \mathbf{f}_k$.

1. Let $\hat{\mathbf{f}}_k^{(t)} = \mathbf{W}\bar{\mathbf{f}}_k^{(t-1)}$.

2. Let

$$\bar{\mathbf{f}}_k^{(t)} = [\bar{f}_{i,k}^{(t)}] \quad \{i = 1 \ldots I\}, \quad \text{with} \begin{cases} \bar{f}_{i,k}^{(t)} = 1 & \text{iff } \hat{f}_{i,k}^{(t)} \geq \vartheta_i \\ \bar{f}_{i,k}^{(t)} = 0 & \text{iff } \hat{f}_{i,k}^{(t)} < \vartheta_i \end{cases} . \tag{3.3}$$

This amounts to setting the cells whose activation level is larger than or equal to their threshold to the value 1, and to setting the cells whose activation level is smaller than their threshold to the value of 0.

3. Compare

$$\bar{\mathbf{f}}_k^{(t-1)} \text{ and } \bar{\mathbf{f}}_k^{(t)}.$$

If

$$\bar{\mathbf{f}}_k^{(t-1)} \neq \bar{\mathbf{f}}_k^{(t)},$$

then change t to $t+1$ and re-iterate the procedure from step **1**. If

$$\bar{\mathbf{f}}_k^{(t-1)} = \bar{\mathbf{f}}_k^{(t)}$$

(or $\bar{\mathbf{f}}_k^{(t-1)} \simeq \bar{\mathbf{f}}_k^{(t)}$, if an approximation is judged sufficient) then a stable response has been found and the procedure can stop.

The stable responses constitute the *attractors* of the system ([33, 34], for the *BSB* models cf. [27]). This algorithm is equivalent to searching for the binary vector $\mathbf{s}$ minimizing the energy values $E_{\mathbf{s}}$ computed as:

$$E_{\mathbf{s}} = -\mathbf{s}\mathbf{W}\mathbf{s}^T + \boldsymbol{\vartheta}\mathbf{s}^T \tag{3.4}$$

using a steepest descent algorithm with starting point $\mathbf{f}_k$. It can be shown that $E_{\mathbf{s}}$ is decreasing (or stays constant) at each iteration. This property is often expressed by saying that $E_{\mathbf{s}}$ is a *Lyapounov* function of the dynamical system implemented by the network [13, 47, 63, 73, 77].

In general, the minimum reached by the network is a *local minimum*. In order to improve the chances of reaching a *global minimum* several equivalent techniques can be used: the *Boltzmann machines* [5, 32], *Markov random fields* [4, 26, 54], or simulated annealing [1, 40, 78].

3.2. Boltzmann machines

Boltzmann machines can be seen as a variation on the theme of Hopfield networks. The essential idea is to used the activation of the cell as a *probability*, and to use it to switch the state of a cell to the *on* (*i.e.*, to the value 1) or *off* (*i.e.*, to the value 0) state. The unique aspect of the Boltzmann machine is that it can be seen as analogous to a physical system in which the probabilities of changing the states of cells in the network can be affected by a global "temperature" parameter. When the temperature is high, neurons change states with higher probability than when the temperature is low. During the stabilization process, the temperature of the network is gradually lowered, (a process known as "annealing" due to its similarity to the annealing of metals), resulting in a system that becomes progressively more stable over time.

Formally, the local energy difference at the i-th unit for the configuration $\mathbf{s}$ is defined as

$$\Delta E_i = \mathbf{w}_i^T \mathbf{s} - \vartheta_j \qquad (\text{with } \mathbf{w}_i\text{: } i\text{-th column of } \mathbf{W}) \ . \tag{3.5}$$

This is equivalent to computing for the i-th unit the difference of energy between the configuration with the unit being *on* and the configuration with the unit being *off*.

In the previous algorithm describing the Hopfield network, step **2** will be modified by deciding to set $\bar{f}_{i,k}^{(t)}$ equal to 1 with the probability

$$P_i = \frac{1}{1 + e^{-\Delta E_i / T}} \tag{3.6}$$

where T is a real positive value corresponding to the temperature of the system (this is known as the *logistic*, or *Boltzmann*, or *Fermi* equation). As mentioned at the beginning of this section, at high temperatures, the system tends to be less influenced by the value of ΔE_i. When T tends towards 0, the logistic function degenerates into the step function and the Boltzmann machine becomes a Hopfield net. When T tends towards infinity, P_i tends towards .5 no matter what the value of ΔE_i is, and, as a consequence, the Boltzmann machine tends to behave randomly (*i.e.*, it ignores the information given by the system).

The probability of transiting from one state to the other can be expressed as a ratio of probabilities. For example, to evaluate the probability of reaching the state $\mathbf{s}$ from the state $\mathbf{s}'$ with energy $E_{\mathbf{s}}$ and $E_{\mathbf{s}'}$ respectively, compute the ratio of probabilities which is:

$$\frac{P_{\mathbf{s}}}{P_{\mathbf{s}'}} = \frac{e^{-E_{\mathbf{s}}/T}}{e^{-E_{\mathbf{s}'}/T}} = e^{-(E_{\mathbf{s}} - E_{\mathbf{s}'})/T} \ . \tag{3.7}$$

As the temperature rises, this ratio gets closer to the value 1, indicating that it is easier at high temperature to go from one state to the other. It should be possible, therefore, to escape local minima (but at the price of increasing the risk of reaching a solution far from a minimum). In practice, the strategy is to start the process at a relatively high temperature and then to reduce the temperature progressively very slowly. It is relatively fair to say that finding the proper annealing schedule for a given problem can be considered as an intuitive art more than anything else.

This procedure can also be analyzed with reference to spin glasses and mean field theories, which are domains relatively well studied in physics [30, 31]. This may explain the popularity of these neural networks among physicists and the strong impact Hopfield's paper has had on the multidisciplinary nature of the neural networks field.

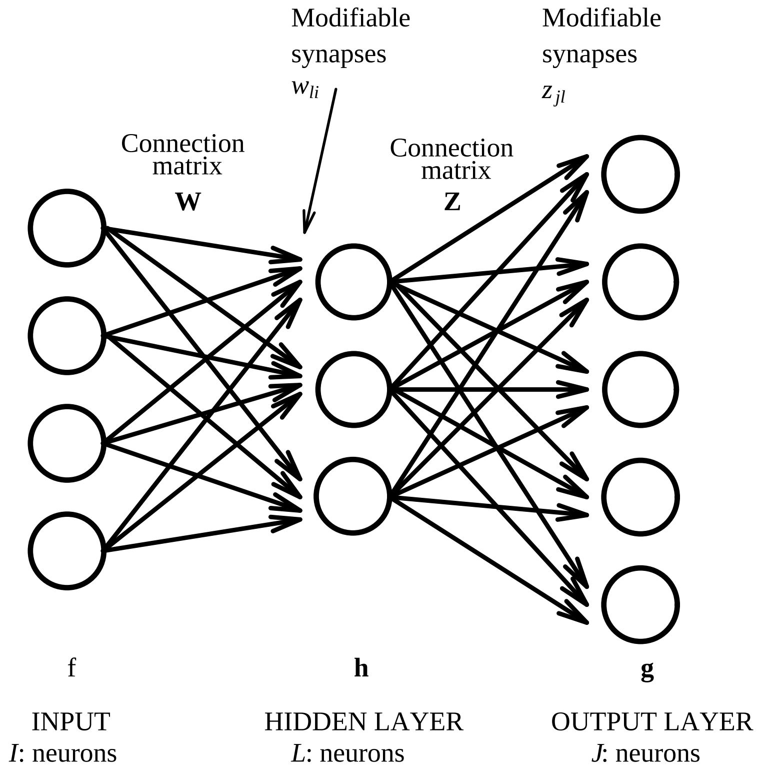

FIG. 6. — The architecture of a typical error back-propagation network. It is composed of (at least) three layers, one input layer, one (or several) hidden layer(s), and one output layer. The input layer is made of I cells or neurons, the hidden layer of L cells or neurons, the output layer of J cells or neurons. The weights $w_{\ell,i}$ of the connections from the input layer to the hidden layer are stored in the $L \times I$ matrix $\mathbf{W}$. The weights $z_{j,\ell}$ of the connections from the hidden layer to the output layer are stored in the $J \times L$ matrix $\mathbf{Z}$. The k-th stimulus is denoted $\mathbf{f}_k$, the corresponding response of the hidden layer is denoted $\mathbf{h}_k$, the response of the output layer is denoted $\hat{\mathbf{g}}_k$, and the corresponding target response is denoted $\mathbf{g}_k$.

4. Hidden layer networks and back-propagation

As noted previously in the perceptron section, it can be shown that if one or more "hidden layers" are added between the input and output cell layers, several of the limitations of perceptrons can be overcome. The main problem that plagued early neural network researchers was the absence of a learning rule to adjust the weights of the hidden layer(s) based on the set of stimuli to be learned. In other words, while it was easy to note when the activation of an output cell was incorrect, assigning "blame" to individual connections within multi-layered networks was a difficult problem. Since then, several rules have been proposed, the most popular of which is *error back-propagation*, discovered in the late sixties by Bryson and Ho [16], and rediscovered by Werbos in 1974 [83]. It was, once again, independently rediscovered and popularized in the early eighties by several authors (Le Cun, [45]; Parker, [60]; Rumelhart, Hinton and Williams [72], with these latter authors being its most ardent proselytes).

An error back-propagation network is composed of at least three layers of cells: an input layer, one or more hidden layers, and an output layer. In this introduction, I assume for simplicity, but without loss of generality, that the networks have only one hidden layer. A back-propagation network is described in Figure 6.

The essential idea behind back-propagation is actually quite simple and straighforward. Like the perceptron, back-propagation networks use supervised learning: they need to know what response they should give for a given stimulus. The cells of the output layer compute the error as the difference between the actual response of the network and the intended response. The synaptic weights on these cells will be adjusted using the standard Widrow-Hoff learning rule for nonlinear units. In other words, the connection weights will be adjusted in order to decrease the error signal if the same stimulus is presented again. The error is propagated *backward* through the *same* connections and synaptic weights to the cells of the hidden layer. Next, the error at the hidden cells is estimated as the weighted average of the error of the output cells. The values of the connections $z_{j,\ell}$ are used as the weights for computing the weighted average. So, for example, if an output cell has a large positive error, and if the synaptic weight between a given hidden cell and the output cell is large, it implies that the hidden cell has a large part in the output cell error. Thus, the error signal for the hidden cell should show a large component of error coming from this output cell. After the error signal of each hidden cell has been estimated, its synaptic weights are adjusted appropriately using the standard Widrow-Hoff learning rule, exactly as for the output cells.

More precisely, for a network with one input layer, one hidden layer, and one output layer, the following notations are defined:

- $\mathbf{f}_k$: the $I \times 1$ vector representing the k-th stimulus (*i.e.*, the input layer is made of I cells).
- $\mathbf{h}_k$: the $L \times 1$ vector representing the response of the hidden layer for the k-th stimulus (*i.e.*, the hidden layer is made of L cells).
- $\hat{\mathbf{g}}_k$: the $J \times 1$ vector representing the response of the output layer for the k-th stimulus (*i.e.*, the output layer is made of J cells).
- $\mathbf{g}_k$: the $J \times 1$ vector representing the target (or desired) response of the output layer for the k-th stimulus.
- $\mathbf{W}$: the $L \times I$ matrix of synaptic weights of the connections between the I cells of the input layer and the L cells of the hidden layer; $w_{\ell,i}$ gives the weight of the connection between the i-th input cell and the ℓ-th cell of the hidden layer.
- $\mathbf{Z}$: the $J \times L$ matrix storing the weights of the connections between the hidden layer and the output layer; $z_{j,\ell}$ gives the weight of the connection between the ℓ-th cell of the hidden layer and the j-th cell of the output layer.

In order to use error back-propagation, the response of a cell should be a nonlinear function of the cell activation (technically speaking, the nonlinearity is necessary only for the hidden layer cells). Denoting the activation of cell n by a_n, its response will be

$$o_n = f(a_n)\ . \tag{4.1}$$

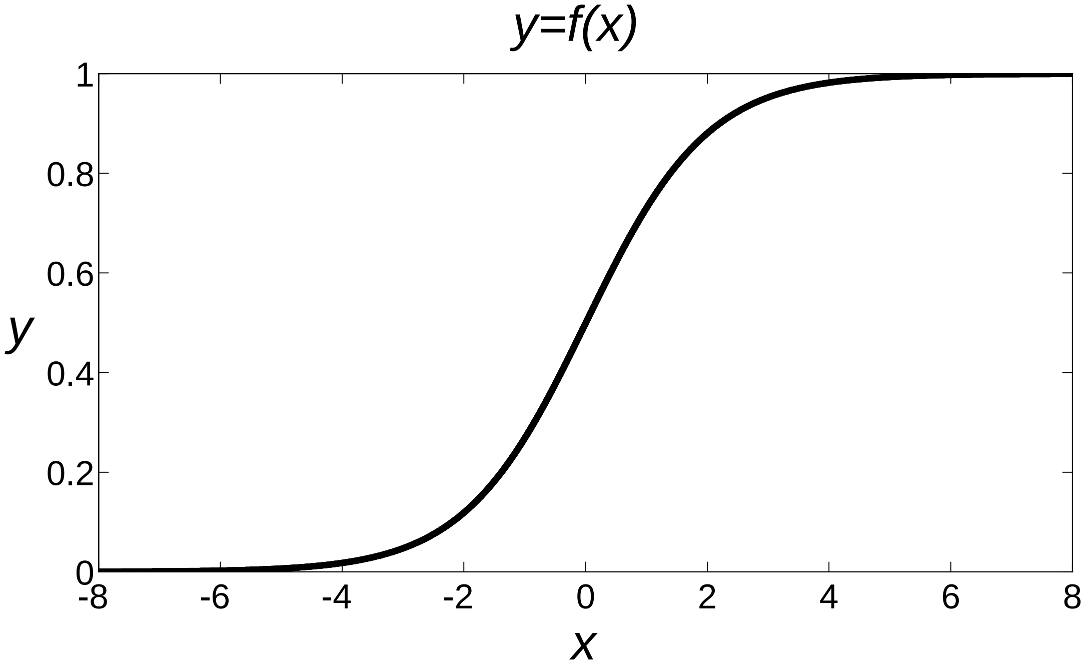

FIG. 7. — Graph of the logistic function.

While there are several acceptable functions, the most widely used is the logistic function drawn in Figure 7. It is given by the following Equation (cf., also Equation 3.6):

$$f(x) = \frac{1}{1+e^{-x}} \ . \tag{4.2}$$

The logistic function maps the activation into the continous interval [0, 1]. One reason for its popularity is the ease of computing its derivative:

$$f'(x) = -\frac{e^{-x}}{(1+e^{-x})^2} = f(x)[1-f(x)] \ . \tag{4.3}$$

The answer of the network to an input stimulus is given by a forward transmission of the stimulus. First, the signal is forwarded from the input layer to the hidden layer. The activation of the hidden layer cells is computed and then tranformed or filtered using a nonlinear function (*e.g.*, the logistic function). So, if the k-th stimulus, noted $\mathbf{f}_k$, is presented, the response of the hidden layer will be:

$$\mathbf{h}_k = f(\mathbf{W}\mathbf{f}_k) \ . \tag{4.4}$$

Next, the hidden layer response is forwarded to the output layer cells. The activation and transformation of these cells is computed in a manner similar to that used for the hidden layer cells. So, the response of the output layer to the k-th stimulus will be (assuming for convenience that the same nonlinear function is used for all the cells):

$$\hat{\mathbf{g}}_k = f(\mathbf{Z}\mathbf{h}_k) \ . \tag{4.5}$$

Since back-propagation is a supervised learning technique, the output layer cells must be provided with the expected answer. Once the error signal at the output cells has been determined, the process of correcting the synaptic weights in order to minimize the error signal (actually, the mean square error signal) begins. The correction procedure is the same for all the cells, the only difference is that the procedure used to evaluate the error signal differs between the hidden layer cells and the output layer cells.

For the cells of the output layer, the error is evaluated by comparing the actual response of the cell with its expected response. So, the error for output cell j is

$$e_j = g_j - \hat{g}_j \ .$$

Using vector notation, the $J \times 1$ error vector of the output cells for the k-th stimulus is:

$$\mathbf{e}_k = (\mathbf{g}_k - \hat{\mathbf{g}}_k) \ . \tag{4.6}$$

The *error signal*, combines the cell error with its activation. Specifically, the error signal is weighted by the derivative of the output activation (*e.g.*, as given in Equation 4.3). Precisely:

$$\boldsymbol{\delta}_{\text{output},k} = f'(\mathbf{Z}\mathbf{h}_k) \circledast (\mathbf{e}_k) = \hat{\mathbf{g}}_k \circledast (\mathbf{1} - \hat{\mathbf{g}}_k) \circledast (\mathbf{g}_k - \hat{\mathbf{g}}_k) \ , \tag{4.7}$$

where $\circledast$ denotes the Hadamar product (*i.e.*, the elementwise product, cf. [74, 35]) and $\mathbf{1}$ denotes a unitary vector of the appropriate order. The function f is assumed to be the logistic function for the second term of the previous Equation to be simplified as it is.

The learning rule has the form of the learning rule described previously for the perceptron or for the linear associator. It corrects the matrix $\mathbf{Z}$ iteratively. At step $(t+1)$, the weights stored in matrix $\mathbf{Z}$ become:

$$\mathbf{Z}_{(t+1)} = \mathbf{Z}_{(t)} + \eta \boldsymbol{\delta}_{\text{output},k} \mathbf{h}_k^T = \mathbf{Z}_{(t)} + \Delta_t \mathbf{Z} \tag{4.8}$$

(with k randomly chosen, and η being a small positive constant).

For the cells of the hidden layer, the error signal cannot be evaluated by direct comparison with the target since the target is not defined at that level. The error signal is estimated as a function of the output error, of the synaptic weights, and of the (derivative of the) response of the hidden layer cells. Precisely, the error signal vector for the hidden layer cells is computed as

$$\boldsymbol{\delta}_{\text{hidden},k} = f'(\mathbf{W}\mathbf{f}_k) \circledast (\mathbf{Z}^T \boldsymbol{\delta}_{\text{output},\,k}) = \mathbf{h}_k \circledast (\mathbf{1} - \mathbf{h}_k) \circledast (\mathbf{Z}^T \boldsymbol{\delta}_{\text{output},\,k}) \ . \tag{4.9}$$

So the error signal from the output layer is backpropagated to the hidden layer through the weights of the connections between the output layer and the hidden layer (this is the term $\mathbf{Z}^T \boldsymbol{\delta}_{\text{output},\,k}$ in Equation 4.9). This amounts to computing the hidden layer cell's error component as the weighted average of the error signal of the output layer cells. Then, the error component for each cell is weighted by the derivative of its response in order to create the error signal [this is the term $f'(\mathbf{W}\mathbf{f}_k)$ in Equation 4.9].

When the error signal for the cells in the hidden layer has been computed, learning is implemented as for the output layer cells. During the learning process, the matrix $\mathbf{W}$ is corrected iteratively. At step $(t+1)$, the weights stored in matrix $\mathbf{W}$ become:

$$\mathbf{W}_{(t+1)} = \mathbf{W}_{(t)} + \eta \boldsymbol{\delta}_{\text{hidden},k} \mathbf{f}_k^T = \mathbf{W}_{(t)} + \Delta_t \mathbf{W} \ . \tag{4.10}$$

In the next section, I show that back-propagation converges (if η is appropriately chosen) toward a local minimum of the mean square of the error for the output layer. Specifically, back-propagation implements gradient descent, a well-known procedure in numerical analysis.

4.1. Error back-propagation and gradient descent

The gradient descent method is a relatively well-known method of numerical analysis [20, 22, 61, 81] used to locate iteratively a minimum of a nonlinear derivable function.

The gradient of a function is defined as the matrix of the (first) derivative of that function. Suppose that the parameters of a function are stored in a matrix denoted $\mathbf{Z}$ and that the function under consideration is

$$y = g(\mathbf{Z}) \tag{4.11}$$

(*i.e.*, the problem is to find the matrix $\mathbf{Z}$ so that y reaches its minimum value). The algorithm proceeds as follows:

- **1.** Chose arbitrarily the initial values of $\mathbf{Z}_{(t=0)}$. In general, these values will be chosen randomly, but a first good guess can also be used if any can be made.
- **2.** Compute the (local) gradient of g denoted by ∇g as:

$$\nabla g = \frac{\partial g(\mathbf{Z}_{(t)})}{\partial \mathbf{Z}_{(t)}} \ . \tag{4.12}$$

- **3.** Change the values of $\mathbf{Z}_{(t)}$ in the *inverse* direction of its gradient (this is because the gradient indicates the direction in which the function increases; so going in the inverse direction indicates the direction of a possible minimum). If η denotes a small positive constant, the correction to apply to $\mathbf{Z}_{(t)}$ is:

$$\mathbf{Z}_{(t+1)} = \mathbf{Z}_{(t)} + \Delta_t \mathbf{Z} = \mathbf{Z}_{(t)} - \eta \nabla g = \mathbf{Z}_{(t)} - \eta \frac{\partial g(\mathbf{Z}_{(t)})}{\partial \mathbf{Z}_{(t)}} \ . \tag{4.13}$$

- **4.** If

$$\mathbf{Z}_{(t)} = \mathbf{Z}_{(t+1)} \qquad (\text{or } \mathbf{Z}_{(t)} \simeq \mathbf{Z}_{(t+1)} \text{ if an approximation is sufficient}) \tag{4.14}$$

  then stop the procedure, otherwise reiterate steps **2** and **3**.

For an error back-propagation network, the error function is defined as the sum of squares of the differences between the expected values $\mathbf{g}_k$ and the responses of the network $\hat{\mathbf{g}}_k$. Specifically, the error function for the k-th response is defined as:

$$E_k = \tfrac{1}{2}(\mathbf{g}_k - \hat{\mathbf{g}}_k)^T(\mathbf{g}_k - \hat{\mathbf{g}}_k) = \tfrac{1}{2}(\mathbf{g}_k^T\mathbf{g}_k + \hat{\mathbf{g}}_k^T\hat{\mathbf{g}}_k - 2\hat{\mathbf{g}}_k^T\mathbf{g}_k) \ . \tag{4.15}$$

4.1.1. Gradient correction for the output layer

For the output layer, the matrix of parameters is the weight matrix $\mathbf{Z}$. The gradient of E_k relative to $\mathbf{Z}$ is computed using the *chain rule* adapted to matrices (cf. [50, 51]):

$$\nabla_{\mathbf{Z}} E_k = \frac{\partial E_k}{\partial \mathbf{Z}} = \frac{\partial E_k}{\partial \hat{\mathbf{g}}_k} \frac{\partial \hat{\mathbf{g}}_k}{\partial \mathbf{Z}\mathbf{h}_k} \frac{\partial \mathbf{Z}\mathbf{h}_k}{\partial \mathbf{Z}} \ . \tag{4.16}$$

Evaluating each of the terms of Equation 4.16 gives:

$$\frac{\partial E_k}{\partial \hat{\mathbf{g}}_k} = -(\mathbf{g}_k - \hat{\mathbf{g}}_k)^T \ , \tag{4.17}$$

assuming f is the logistic function, and with $\hat{\mathbf{g}}_k = f(\mathbf{Z}\mathbf{h}_k)$:

$$\frac{\partial \hat{\mathbf{g}}_k}{\partial \mathbf{Z}\mathbf{h}_k} = \hat{\mathbf{g}}_k^T \circledast (\mathbf{1} - \hat{\mathbf{g}}_k)^T \ , \tag{4.18}$$

and

$$\frac{\partial \mathbf{Z}\mathbf{h}_k}{\partial \mathbf{Z}} = \mathbf{h}_k \ . \tag{4.19}$$

The correction for $\mathbf{Z}$ at step t is then proportional to

$$-\nabla_{\mathbf{Z}} E_k = (\mathbf{g}_k - \hat{\mathbf{g}}_k)^T \circledast \hat{\mathbf{g}}_k^T \circledast (\mathbf{1} - \hat{\mathbf{g}}_k)^T \mathbf{h}_k = \boldsymbol{\delta}_{\mathrm{output},k}^T \mathbf{h}_k \ . \tag{4.20}$$

Using η as a small constant, this is equivalent to defining the change at step t as:

$$\Delta_t \mathbf{Z} = \eta \boldsymbol{\delta}_{\mathrm{output},k} \mathbf{h}_k^T \tag{4.21}$$

as indicated in Equation 4.8.

4.1.2. Gradient correction for the hidden layer

For the hidden layer, the matrix of parameters is the weight matrix $\mathbf{W}$. The gradient of E_k relative to $\mathbf{W}$ is computed using, once again, the *chain rule* adapted to matrices:

$$\nabla_{\mathbf{W}} E_k = \frac{\partial E_k}{\partial \mathbf{W}} = \frac{\partial E_k}{\partial \hat{\mathbf{g}}_k} \frac{\partial \hat{\mathbf{g}}_k}{\partial \mathbf{Z}\mathbf{h}_k} \frac{\partial \mathbf{Z}\mathbf{h}_k}{\partial \mathbf{h}_k} \frac{\partial \mathbf{h}_k}{\partial \mathbf{W}\mathbf{f}_k} \frac{\partial \mathbf{W}\mathbf{f}_k}{\partial \mathbf{W}} \ . \tag{4.22}$$

The first two terms of Equation 4.22 have been defined previously (cf. Equation 4.16) and correspond to $-\boldsymbol{\delta}_{\mathrm{output},\ k}^T$. Evaluating the other terms gives:

$$\frac{\partial \mathbf{Z}\mathbf{h}_k}{\partial \mathbf{h}_k} = \mathbf{Z}^T \ , \tag{4.23}$$

assuming f is the logistic function, and with $\mathbf{h}_k = f(\mathbf{W}\mathbf{f}_k)$,

$$\frac{\partial \mathbf{h}_k}{\partial \mathbf{W}\mathbf{f}_k} = \mathbf{h}_k^T \circledast (\mathbf{1} - \mathbf{h}_k)^T \ , \qquad (4.24)$$

and, finally:

$$\frac{\partial \mathbf{W}\mathbf{f}_k}{\partial \mathbf{W}} = \mathbf{f}_k \ . \qquad (4.25)$$

Hence, the correction for $\mathbf{W}$ at step t is proportional to:

$$-\nabla_{\mathbf{W}} E_k = \boldsymbol{\delta}_{\text{output},k}\mathbf{Z}^T \circledast \mathbf{h}_k^T \circledast (\mathbf{1} - \mathbf{h}_k)^T \mathbf{f}_k = \boldsymbol{\delta}_{\text{hidden},k}^T \mathbf{f}_k \ . \qquad (4.26)$$

Using η as a small constant and transposing $\mathbf{f}_k$ is equivalent to defining the change at step t as:

$$\Delta_t \mathbf{W} = \eta \boldsymbol{\delta}_{\text{hidden},k} \mathbf{f}_k^T \qquad (4.27)$$

as indicated in Equation 4.10.

4.1.3. Gradient and linear associators

The Widrow-Hoff learning rule used for linear associators can be interpreted also as a gradient descent technique. With the notation defined in the previous sections, the function f used by the cell to transform the activation in a response is now very simple (it is a *linear* transformation of the activation, hence the name *linear* associator, cf. Equations 2.4 and 4.1):

$$o_n = f(a_n) = a_n \ . \qquad (4.28)$$

The response of the associator becomes (cf. Equation 2.4):

$$\hat{\mathbf{g}}_k = \mathbf{W}\mathbf{f}_k \ .$$

The error function is as previously defined (cf. Equation 4.15):

$$E_k = \tfrac{1}{2}(\mathbf{g}_k - \hat{\mathbf{g}}_k)^T(\mathbf{g}_k - \hat{\mathbf{g}}_k) = \tfrac{1}{2}(\mathbf{g}_k^T\mathbf{g}_k + \hat{\mathbf{g}}_k^T\hat{\mathbf{g}}_k - 2\hat{\mathbf{g}}_k^T\mathbf{g}_k) \ .$$

Since a linear associator has only one set of weights, the problem is to find the values of $\mathbf{W}$ minimizing E_k for all k. The gradient of E_k relative to $\mathbf{W}$ is evaluated, as usual, using the chain rule rewritten for matrices.

$$\nabla_{\mathbf{W}} E_k = \frac{\partial E_k}{\partial \mathbf{W}} = \frac{\partial E_k}{\partial \hat{\mathbf{g}}_k}\frac{\partial \hat{\mathbf{g}}_k}{\partial \mathbf{W}} \ . \qquad (4.29)$$

As noted previously (cf. Equation 4.16), the first term on the right of Equation 4.29 is:

$$\frac{\partial E_k}{\partial \hat{\mathbf{g}}_k} = -(\mathbf{g}_k - \hat{\mathbf{g}}_k)^T \ ,$$

and with $\hat{\mathbf{g}}_k = f(\mathbf{W}\mathbf{f}_k) = \mathbf{W}\mathbf{f}_k$, the second term of Equation 4.29 becomes

$$\frac{\partial \hat{\mathbf{g}}_k}{\partial \mathbf{W}} = \frac{\partial \mathbf{W}\mathbf{f}_k}{\partial \mathbf{W}} = \mathbf{f}_k \ . \tag{4.30}$$

The correction for $\mathbf{W}$ at step t should be proportional to:

$$-\nabla_{\mathbf{W}} E_k = -\frac{\partial E_k}{\partial \mathbf{W}} = (\mathbf{g}_k - \hat{\mathbf{g}}_k)^T \mathbf{f}_k \ . \tag{4.31}$$

Using η as the proportionality constant, the correction for $\mathbf{W}$ at step t becomes:

$$\Delta_t \mathbf{W} = \eta(\mathbf{g}_k - \hat{\mathbf{g}}_k)\mathbf{f}_k = \eta(\mathbf{g}_k - \mathbf{W}_{(t)}\mathbf{f}_k)\mathbf{f}_k^T \tag{4.32}$$

as indicated in Equations 1.5 and 2.6.

4.2. Evaluation of Back-propagation

It is probably fair to say that the "rediscovery" of back-propagation in the mid-eighties was a pivotal factor in the renaissance of neural networks. With this new technique, neural networks were now able to overcome the primary weaknesses of the perceptron that became clear in the sixties. In particular, neural networks were now able to solve nonlinear mappings such as those required to solve the XOR and parity problems, as well as most of the so-called *hard* problems that Minsky and Papert [53] noted as failures of the perceptron.

Cognitive scientists, in addition, found back-propagation a useful tool for exploring issues concerning the representation of knowledge. In this case, the hidden layer acts as the internal knowledge representation a network uses to make the transformation from input to output. The interest for cognitive scientists, therefore, is to find how the hidden layer develops weights that solve the problem. For example, NETALK was designed by Rosenberg and Sejnowski [67] for the task of converting written English into spoken English. They found that the hidden layers developed a representation somewhat analogous to the phonemes of English (*e.g.*, consonants *versus* vowels, voiced *versus* unvoiced phonemes).

Taking into account its popularity and the number of its applications, it is tempting to trumpet back-propagation as a *panacea* for computational models of cognition and biological signal processing. Some dedicated proselytes have, in fact, had difficulty resisting such a temptation. There are, however, a number of serious drawbacks to the technique. The first one is obvious. In order to implement back-propagation, the network needs to be provided with the correct answer (*i.e.*, back-propagation is a supervised technique). That, by itself, eliminates all applications for which the task is to find stimulus classes *a posteriori* or without supervision. A second problem comes from the number of iterations needed for the network to converge. The empirical evidence of the last five years or so has indicated that even for a simple problem like learning the XOR function, the number of iterations required for convergence can reach into the thousands, when a perceptron will solve it as a ternary relation in less than 10 iterations

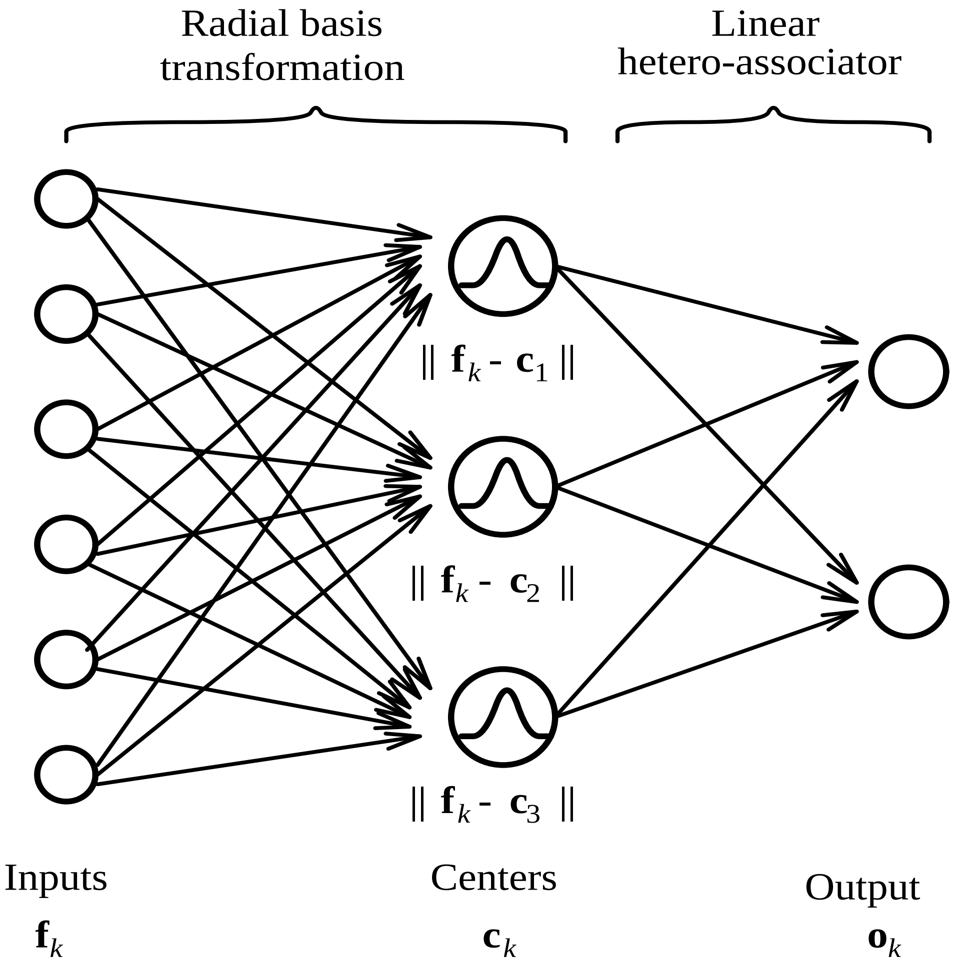

FIG. 8. — The architecture of a typical radial basis function network. The hidden layer computes the distance from the input to each of the centers (each center corresponds to a cell of the hidden layer). The cells of the hidden layer transform their activation (*i.e.*, the distance from the input) in response using a nonlinear transformation (typically a Gaussian function). The cells of the output layer behave like a standard linear hetero-associator.

(cf. [3]). Finally, and perhaps most problematic, it is always difficult to determine, for a given problem, the best architecture in terms of number of hidden layers as well as number of cells *per* layer.

Despite these problems, back-propagation is still widely used. Several alternative models, that attempt to overcome its limitations, are currently under development. One of these is the *Radial basis function* network described in the next section.

5. Radial basis function networks

Radial basis function networks are a recent addition to the neural-modeler toolbox [19, 15, 59, 64, 65]. The architecture of a typical radial basis function network is shown in Figure 8.

These networks are used for finding an approximation of a nonlinear function as well as for finding interpolating values of a function defined only on a finite subset of real numbers. The essential idea is to implement a complex nonlinear mapping from the input pattern space to the output pattern space as a two-step process. The first step is a simple nonlinear mapping from the input layer to the hidden layer. The second step implements a linear transformation from the hidden layer to the output layer. Learning occurs only at the level of the synapses between the hidden layer and the output layer. Because these connections are linear, learning can be very fast.

Specifically, the general problem is to approximate an arbitrary function f from $\mathbb{R}^I$ to $\mathbb{R}^J$ (*i.e.*, f associates a $J \times 1$ dimensional vector $\mathbf{g}_k$ in response to a $I \times 1$ dimensional vector $\mathbf{f}_k$) defined on K observations such that

$$\mathbf{g}_k = f(\mathbf{f}_k) \qquad \text{for } k = \{1, \ldots, K\} \ . \tag{5.1}$$

The general idea is to approximate f by a weighted sum of (in general nonlinear) functions ϕ (named the *basis* functions) such that Equation 5.1 is approximated by

$$\mathbf{g}_k \approx \sum_{\ell} w_\ell \phi(\mathbf{f}_k) \ . \tag{5.2}$$

This technique is called *radial* basis function approximation when instead of using the values $\mathbf{f}_k$ directly, several "centers" are chosen (either arbitrarily or in some specific way) and the *distance* from the vectors $\mathbf{f}_k$ to these centers is used in Equation 5.2 instead of the $\mathbf{f}_k$ values.

A center can be any I-dimensional vector (so that the distance between the centers and the $\mathbf{f}_k$ is always defined). Specifically, if a set of L centers $\mathbf{c}_\ell$ is chosen (with each center being an I dimensional vector), Equation 5.1 is approximated as

$$\mathbf{g}_k \approx \sum_{\ell} w_\ell \phi(\|\mathbf{f}_k - \mathbf{c}_\ell\|) \ . \tag{5.3}$$

When the set of centers (the $\mathbf{c}_\ell$'s) is the same set as the input set (*i.e.*, the $\mathbf{f}_k$'s), the radial basis function network is used for finding an interpolation function valid for new values of $\mathbf{f}_{k'}$ and fitting f perfectly for the K observations $\mathbf{f}_k$. When the set of centers differs from the input set (*i.e.*, the $\mathbf{f}_k$), it contains, in general, a smaller number of elements (*i.e.*, $L < K$). The problem, then, can be seen as a problem of approximating the function f by a set of simpler functions ϕ. In both cases, the objective is close to some rather well-known techniques in numerical analysis (*e.g.*, spline interpolations) with the difference that a distance to the centers is used in the process rather than the raw data.

Equation 5.3 can be rewritten in a more compact form with the notations defined in the section on the hetero-associator. Denote by $\mathbf{C}$ the $L \times I$ matrix of the centers (*i.e.*, $\mathbf{c}_\ell^T$ is the ℓ-th row of $\mathbf{C}$). The distances of the K observations to the L centers are gathered in a $L \times K$ matrix $\mathbf{D}$ with the generic term $d_{\ell,k}$ giving the euclidean distance from observation k to center ℓ:

$$\mathbf{D} = [d_{\ell,k}] = \left[\sqrt{\mathbf{c}_\ell^T \mathbf{f}_k}\right] = \sqrt{(\mathbf{C} \circledast \mathbf{C}) \times \mathbf{1}_{I \times K} + \mathbf{1}_{L \times I}(\mathbf{F}^T \circledast \mathbf{F}^T) - 2 \times \mathbf{C}\mathbf{F}^T} \tag{5.4}$$

with $\mathbf{1}_{I\times K}$ being a $I \times K$ matrix of 1's, the square root function being applied element-wise to the matrix, and $\mathbf{F}$ being the $K \times I$ matrix of the K input patterns applied to the I input cells.

Then, the problem is to find a $L \times J$ matrix $\mathbf{W}$ such that:

$$\mathbf{G} \approx [\phi(\mathbf{D}^T)]\mathbf{W} \tag{5.5}$$

with the function ϕ being applied element-wise to the elements of $\mathbf{D}$, and $\mathbf{G}$ being the $K \times J$ matrix of the K output patterns.

If the matrix $\phi(\mathbf{D})$ is squared, and non-singular, the solution for the matrix $\mathbf{W}$ is obviously:

$$\mathbf{W} = [\phi(\mathbf{D})]^{-1}\mathbf{G} \ . \tag{5.6}$$

If $\phi(\mathbf{D})$ is singular or rectangular, a least-squares approximation is given by

$$\mathbf{W} = [\phi(\mathbf{D})]^{+}\mathbf{G} \qquad \text{with } [\phi(\mathbf{D})]^{+} \text{ being the Moore-Penrose inverse of } [\phi(\mathbf{D})]. \tag{5.7}$$

If the testing set is different from the learning set, then the distance from the elements of the testing set to the centers needs to be computed for each of the elements of the testing set. This distance is then transformed by the ϕ function before being multiplied by the matrix $\mathbf{W}$ to give the estimation of the response to the testing set by the radial basis function network.

Several choices are possible for the ϕ functions. One of the most popular choices is the *Gaussian* function.

$$\phi(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\{-x^2/2\sigma^2\} \tag{5.8}$$

with σ^2 being the *variance* of the Gaussian distribution. The variance can also be approximated for each center separately if necessary.

One reason for the popularity of the Gaussian transformation is that it insures that when $\mathbf{D}$ is squared, and when the centers are not redundant (*i.e.*, no center is present twice), then the matrix $\phi(\mathbf{D})$ is not only non-singular but also positive definite [52] (even though as $\mathbf{D}$ is a distance matrix, it is in general *not* full rank and even not positive semi-definite).

In terms of neural networks, this is equivalent to having a first hidden layer whose purpose is to compute the distance from the input to each of the centers. Each cell of the hidden layer represents a center. Then, the cells of the hidden layer transform their activation (*i.e.*, the distance from the input to the centers) into a response using the ϕ function. The cells of the output layer behave exactly like the cells of a standard linear hetero-associator using the Widrow-Hoff learning rule.

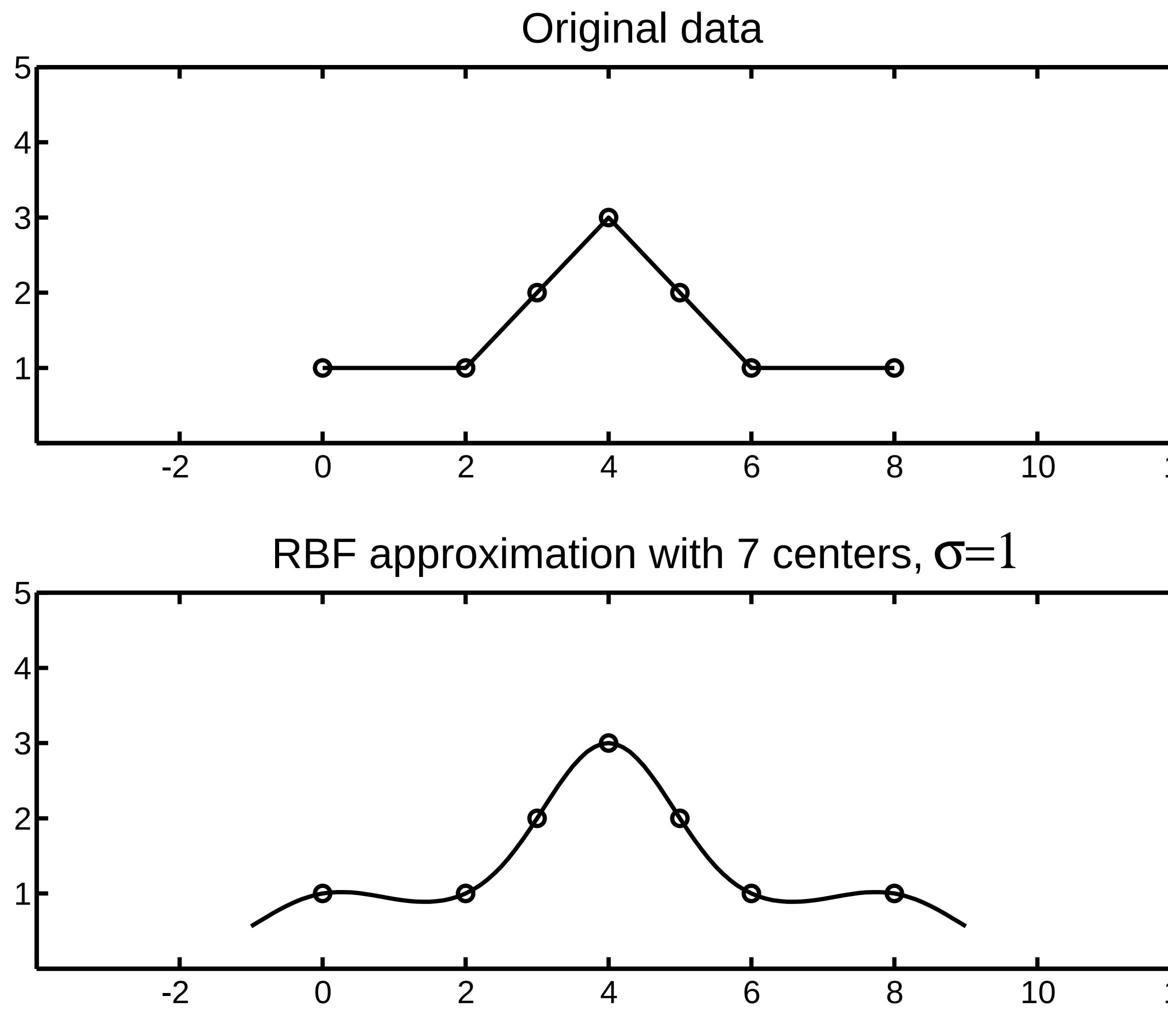

FIG. 9. — Radial basis function approximation example.

5.1. Radial basis function networks: An example

To illustrate a simple radial basis function network, suppose that the function to be approximated associates the following one-dimensional (*i.e.*, $I = J = 1$) set of $K = 7$ stimuli to their response:

$$\begin{aligned} \mathbf{f}_1 = 0 \quad &\longmapsto \quad \mathbf{g}_1 = 1 \\ \mathbf{f}_2 = 2 \quad &\longmapsto \quad \mathbf{g}_2 = 1 \\ \mathbf{f}_3 = 3 \quad &\longmapsto \quad \mathbf{g}_3 = 2 \\ \mathbf{f}_4 = 4 \quad &\longmapsto \quad \mathbf{g}_4 = 3 \\ \mathbf{f}_5 = 5 \quad &\longmapsto \quad \mathbf{g}_5 = 2 \\ \mathbf{f}_6 = 6 \quad &\longmapsto \quad \mathbf{g}_6 = 1 \\ \mathbf{f}_7 = 8 \quad &\longmapsto \quad \mathbf{g}_3 = 1 \end{aligned} \tag{5.9}$$

Or, using a matrix notation, the set of stimuli is stored in the $K \times I = 7 \times 1$ matrix $\mathbf{F}$, and the

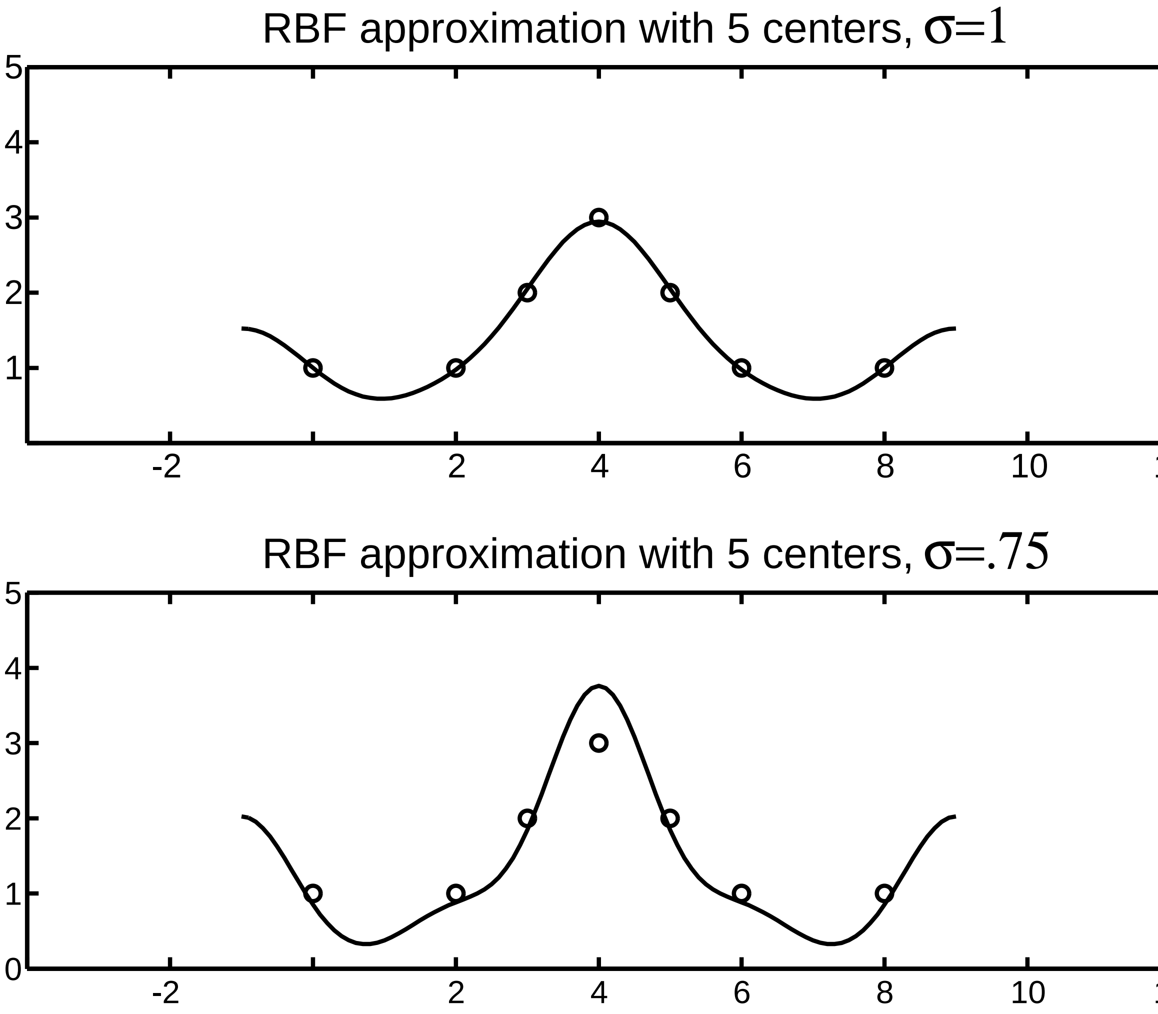

FIG. 10. — Radial basis function approximation example: the effect of different values of sigma. Notice how the first approximation with $\sigma = 1$ gives a much better fit to the data than the approximation with $\sigma = .75$.

set of responses is stored in the $K \times J = 7 \times 1$ matrix $\mathbf{G}$:

$$\mathbf{F} = [0,\ 2,\ 3,\ 4,\ 5,\ 6,\ 8]^T \qquad \text{and } \mathbf{G} = [1,\ 1,\ 2,\ 3,\ 2,\ 1,\ 1]^T\ . \tag{5.10}$$

Suppose that the set of L centers is the same as the set of inputs:

$$\mathbf{C} = \mathbf{F} = [0,\ 2,\ 3,\ 4,\ 5,\ 6,\ 8]^T\ . \tag{5.11}$$

the matrix $\mathbf{D}$ is then

$$\mathbf{D} = \begin{bmatrix} 0 & 2 & 3 & 4 & 5 & 6 & 8 \\ 2 & 0 & 1 & 2 & 3 & 4 & 6 \\ 3 & 1 & 0 & 1 & 2 & 3 & 5 \\ 4 & 2 & 1 & 0 & 1 & 2 & 4 \\ 5 & 3 & 2 & 1 & 0 & 1 & 3 \\ 6 & 4 & 3 & 2 & 1 & 0 & 2 \\ 8 & 6 & 5 & 4 & 3 & 2 & 0 \end{bmatrix}\ . \tag{5.12}$$

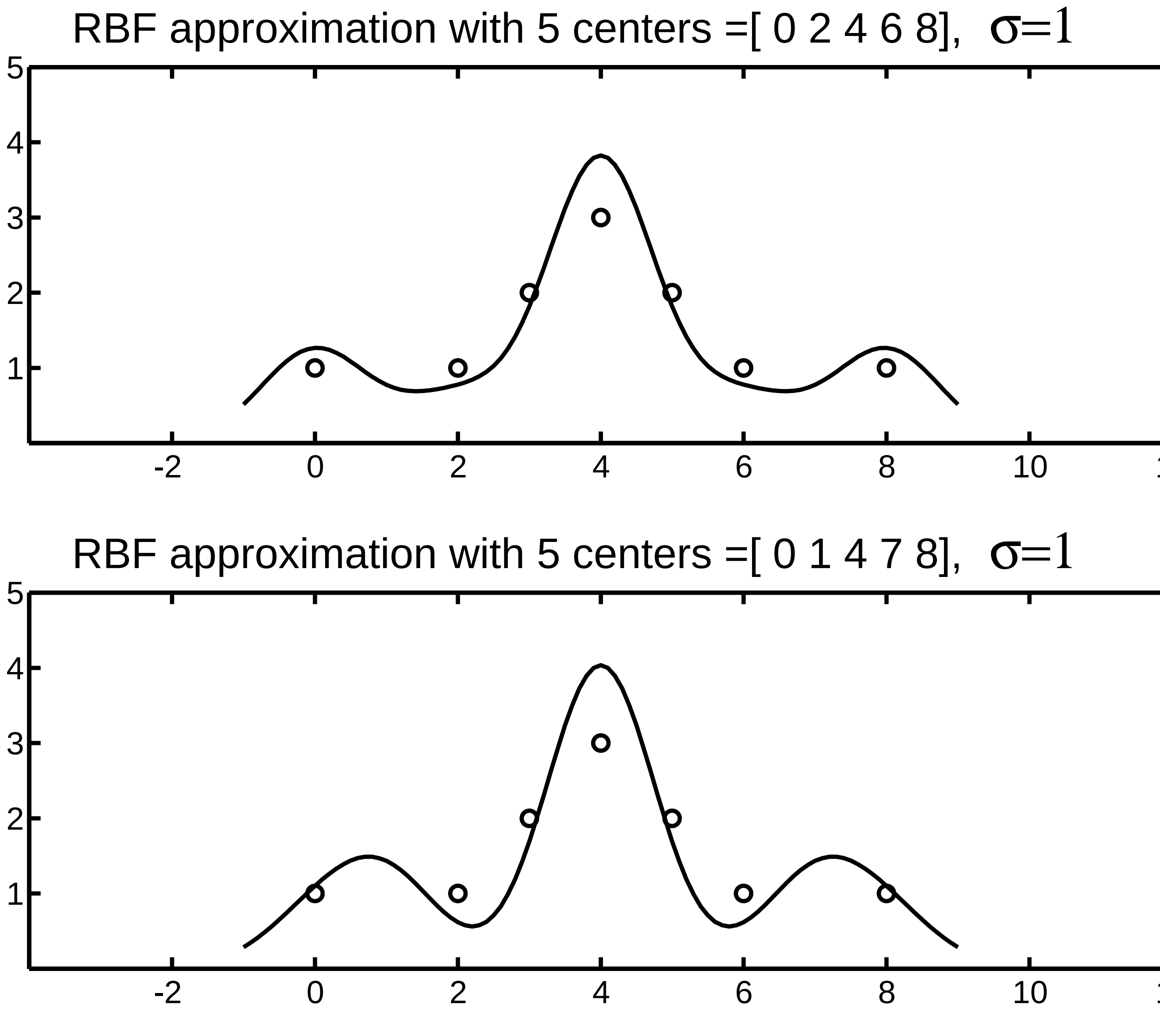

FIG. 11. — The effect of different centers.

Using the Gaussian transformation with $\sigma^2 = 1$ the matrix $\mathbf{D}$ is transformed in:

$$\phi(\mathbf{D}) = [\phi(d_{i,j})] = \left[\frac{1}{\sqrt{2\pi}} \exp\{-d_{i,j}^2/2\}\right] \qquad \mathbf{(5.13)}$$

which gives:

$$\phi(\mathbf{D}) = \begin{bmatrix} 0.3989 & 0.0540 & 0.0044 & 0.0001 & 0 & 0 & 0 \\ 0.0540 & 0.3989 & 0.2420 & 0.0540 & 0.0044 & 0.0001 & 0 \\ 0.0044 & 0.2420 & 0.3989 & 0.2420 & 0.0540 & 0.0044 & 0 \\ 0.0001 & 0.0540 & 0.2420 & 0.3989 & 0.2420 & 0.0540 & 0.0001 \\ 0 & 0.0044 & 0.0540 & 0.2420 & 0.3989 & 0.2420 & 0.0044 \\ 0 & 0.0001 & 0.0044 & 0.0540 & 0.2420 & 0.3989 & 0.0540 \\ 0 & 0 & 0 & 0.0001 & 0.0044 & 0.0540 & 0.3989 \end{bmatrix} . \qquad \mathbf{(5.14)}$$

The optimum matrix of weights which would be found by a hetero-associator can be computed directly by inversion of the matrix $[\phi(\mathbf{D})]$

$$[\phi(\mathbf{D})]^{-1} = \begin{bmatrix} 2.5866 & -0.6348 & 0.5445 & -0.3520 & 0.1932 & -0.0765 & 0.0083 \\ -0.6348 & 5.0744 & -4.7701 & 3.1832 & -1.7667 & 0.7024 & -0.0765 \\ 0.5445 & -4.7701 & 9.3173 & -7.4080 & 4.3547 & -1.7667 & 0.1932 \\ -0.3520 & 3.1832 & -7.4080 & 10.6317 & -7.4080 & 3.1832 & -0.3520 \\ 0.1932 & -1.7667 & 4.3547 & -7.4080 & 9.3173 & -4.7701 & 0.5445 \\ -0.0765 & 0.7024 & -1.7667 & 3.1832 & -4.7701 & 5.0744 & -0.6348 \\ 0.0083 & -0.0765 & 0.1932 & -0.3520 & 0.5445 & -0.6348 & 2.5866 \end{bmatrix} \tag{5.15}$$

and

$$\mathbf{W} = [\phi(\mathbf{D})]^{-1}\mathbf{G} = [2.3029,\ 1.5415,\ -0.6794,\ 7.9252,\ -0.6794,\ 1.5415,\ 2.3029]^T . \tag{5.16}$$

To calculate the answer of the radial basis function network to a stimulus (old or new), it suffices to compute the distance from that stimulus to the 7 centers, to transform the distance matrix with the Gaussian function and to multiply it by the matrix $\mathbf{W}$. For example, the response of the network to the new stimulus $\mathbf{f} = 1$ will be

$$\mathbf{o} = \phi([d_{\mathbf{c}_\ell,\mathbf{f}}])\mathbf{W} = \phi\left\{\begin{bmatrix} 1 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 7 \end{bmatrix}\right\} \times \mathbf{W} = \begin{bmatrix} 0.2420 \\ 0.2420 \\ 0.0540 \\ 0.0044 \\ 0.0001 \\ 0.0000 \\ 0.0000 \end{bmatrix} \times \mathbf{W} = .9286$$

where $d_{\mathbf{c}_\ell,\mathbf{f}}$ denotes the distance from the input $\mathbf{f}$ to the ℓ-th center.

The approximation given by the radial basis function network is illustrated in Figure 9.b for the set of input patterns belonging to the interval $[-1\ 9]$. When compared with a straight line approximation (in Figure 9.a) the network approximation appears quite smooth. Note, also, that, as required, the approximation is perfect for the elements of the training set.

5.1.1. The choice of centers and σ's for an approximation radial basis function network

When a radial basis function network is used for approximating a function (*i.e.*, when the set of centers is smaller than the set of training stimuli), the choice of values for σ becomes very important. This is illustrated in Figure 10.a and 10.b, in which the approximation of the previous data set are displayed. The set of centers is now composed of the 5 values:

$$\mathbf{C} = [-1,\ 2,\ 4,\ 6,\ 9]^T \tag{5.17}$$

Figure 10.a displays the results of the approximation for $\sigma = 1$, and Figure 9.b displays the results of the approximation for $\sigma = .75$. As the comparison makes clear, the choice of σ strongly influences the quality of the approximation.

The choice of the centers (as well as their number) is also important as illustrated by Figure 11.a and 11.b in which the approximation with the set of centers $\mathbf{C} = [0,\ 2,\ 4,\ 6,\ 8]^T$ is compared with the set of centers $\mathbf{C} = [0,\ 1,\ 4,\ 7,\ 8]^T$.

The set of centers is sometimes learned using an unsupervised learning technique like, for example, k-means. The variance of the ϕ function can, similarly be approximated from the sample. However, in both cases choosing these two sets of parameters can be a very delicate operation.

6. Conclusion

In this introductory paper, I have presented some basic tools from the connectionist modeling toolbox. Obviously, this is only an overview of the field. Some examples of practical applications and theoretical developments are presented in the following papers of this volume.

The reader seeking a more detailed introduction to connectionist modeling can consult, among other references, the following recent sources: Aleksander & Morton [6]; Hecht-Nielsen [30]; Kampf & Hasler [37]; Khanna [39]; Müller & Reinhardt [55]; Perez [62]; Simpson [79]; Zeidenberg [86]; Anderson *et al.*[9]; Bechtel & Abrahmson [14]; Freeman & Skapura [24]; Hertz, Krogh & Palmer [31]; Levine [46]; Quilian [66]; and Abdi [3]. More specialized source of reference can be found in fields as varied as physics of complex systems (*e.g.*, Fogelman-Souliè [23]; Serra & Zanarini [76]; Goles & Martinez [28]; Weisbuch [82]), engineering sciences or signal processing (*e.g.*, Widrow & Stearns [85]; Catlin [18]; Souček [80]; Garner [25]; Kosko [43,44]), and neurosciences (*e.g.*, Mac Gregor [49]; Amit [7]).

7. Acknowledgement

Thanks are due to Jay Dowling, Betty Edelman, Alice O'Toole, and Dominique Valentin for comments on earlier drafts of this paper.

References

[1] Aarts E., and Korst J., *Simulated Annealing and Boltzmann Machines.* (Wiley, New York, 1989).

[2] Abdi H., Generalized approaches for connectionist auto-associative memories: Interpretation, implication and illustration for face processing. In *Artificial Intelligence and Cognitive Sciences*, ed. by Demongeot J. (Manchester University Press, Manchester, 1988) pp. 151–164.

[3] Abdi H., *Les Réseaux de Neurones.* (Presses Universitaires de Grenoble, Grenoble, 1994).

[4] Ackley D.H., *A Connectionist Machine for Genetic Hillclimbing.* (Kluwer, Norwell (MA), 1987).

[5] Ackley D.H., Hinton G.E., and Sejnowski, T.J., A learning algorithm for Boltzmann machines. *Cognitive Sci.* **9** (1985) 147–169.

[6] Aleksander I. and Morton, H., *An Introduction to Neural Computing.* (Chapman and Hall, London, 1990).

[7] Amit D.J., *Modelling Brain Function.* (C.U.P., Cambridge, 1989).

[8] Anderson J.A., and Mozer M.C., Categorization and selective neurons. In *Parallel Models of Associative Memory*, ed. by Hinton G.E. and Anderson J.A. (Erlbaum, Hillsade, 1981) pp. 213–236.

[9] Anderson J.A., Pellionisz A. and Rosenfeld E. *Neurocomputing II* (MIT press, Cambridge, 1991).

[10] Anderson J.A., and Rosenfeld E., *Neurocomputing.* (MIT press, Cambridge, 1987).

[11] Anderson J.A., Silverstein J.W., Ritz S.A. and Jones R.S. Distinctive features, categorical perception, and probability learning: some applications of a neural model. *Psychol. Rev.* **84** (1977) 413–451.

[12] Arbib M.A., *Brains, Machines, and Mathematics (2nd Edition).* (Springer Verlag, New York, 1987).

[13] Beltrami E., *Mathematics for Dynamic Modelling.* (Academic press, New York, 1987).

[14] Bechtel W. and Abrahamsen A., *Connectionim and the Mind.* (Blackwell, Oxford, 1991).

[15] Broomhead D.D. and Lowe D., Mutivariable functional interpolation and adaptive networks. *Complex Systems* **2** 321–355.

[16] Bryson A.E. and Ho Y.C., *Applied Optimal Control.* (Blaisdell, New-York, 1969).

[17] Carpenter G. and Grossberg S., *Pattern Recognition by Self-Organizing Neural Networks.* (MIT Press, Cambridge, 1991).

[18] Catlin D.E., *Estimation, Control, and the Discrete Kalman Filter.* (Springer-Verlag, Berlin, 1989).

[19] Chen S., Cowan C.F. and Grant P.M., Orthogonal least square learning algorithms for radial basis function networks. *IEEE Transactions on Neural Networks* **2** (1991) 302–309.

[20] Ciarlet P.G., *Introduction to Numerical Linear Algebra and Optimisation.* (C.U.P., Cambridge, 1989).

[21] Duda R.O. and Hart P.E., *Pattern Classification and Scene Analysis.* (Wiley, New York, 1973).

[22] Fletcher R., *Practical Methods of Optimization.* (Wiley, New York, 1987).

[23] Fogelman-Soulié F., *Contribution à une Théorie du Calcul sur Réseaux.* Thèse d'état, IMAG Grenoble, 1985.

[24] Freeman J.A. and Skapura D.M., *Neural Networks.* (Addison-Wesley, Reading MA, 1991).

[25] Gardner W.A., *Introduction to Random Processes.* (McGraw-Hill, New York, 1990).

[26] Geman S. and Geman D., Stochastic relaxation, Gibbs distributions, and the bayesian restoration of images. *IEEE Proc. on Artificial and Machine Intelligence* **8** (1984) 721–741.

[27] Golden R., The "brain-state-in-a-box" neural model is a gradient descent algorithm. *Journal of Mathematical Psychology* **30** (1986) 73–80.

[28] Goles E. and Martinez S., *Neural and Automata Networks.* (Kluwer, Norwell MA, 1990).

[29] Hebb D.O., *The Organization of Behavior.* (Wiley, New York, 1949).

[30] Hecht-Nielsen R., *Neurocomputing.* (Addison-Wesley, Reading, 1990).

[31] Hertz J., Krogh A. and Palmer R.G., *Introduction to the Theory of Neural Computing.* (Addison-Wesley, Reading, 1991).

[32] Hinton G.E. and Sejnowski T.J., Learning and Relearning in Boltzman machine. In *Parallel Distributed Processing*, ed. by Rumelhart D.E. and McClelland J.L. (MIT Press, Cambridge, 1986)

[33] Hopfield J.J., Neural networks and physical system with emergent collective computational abilities. *Proceedings of the National Academy of Science, USA* **79** (1982) 6871–6874.

[34] Hopfield J.J., Neurons with graded responses have collective computational abilities. *Proceeding of the national academy of Sciences, USA* **81** (1984) 3088–2558.

[35] Horn R.A. and Johnson, C.R., *Matrix Analysis.* (C.U.P., Cambridge, 1985).

[36] Jordan M.I., An introduction to linear algebra in parallel distributed processing In *Parallel Distributed Processing*, ed. by Rumelhart D.E. and McClelland J.L. (MIT Press, Cambridge, 1986).

[37] Kamp Y. and Hasler M., *Recursive Neural Networks for Associative Memory.* (Wiley, New York, 1990).

[38] Kerlinger F.N., *Foundation of Behavioral Research.* (Holt, Rinehart and Winston, New-York, 1986).

[39] Khanna T., *Foundations of Neural Networks.* (Addison-Wesley, Reading, 1990).

[40] Kirkpatrick S., Gelatt C.D. and Vecchi M.P., Optimization by simulating annealing. *Sci.* **220** (1983) 671–680.

[41] Kohonen T., *Associative Memory: A System Theoretical Approach.* (Springer Verlag, Berlin, 1977).

[42] Kohonen T., *Self Organization and Associative Memory.* (Springer Verlag, Berlin, 1984).

[43] Kosko B., *Neural Network and Fuzzy Systems.* (Prentice-Hall, Englewwod Cliffs, 1991).

[44] Kosko B., *Neural Network for Signal Processing.* (Prentice-Hall, Englewwod Cliffs, 1992).

[45] Le Cun Y., Learning process in an asymmetric threshold network. In *Disordered Systems and Biological Organization*, ed. by Bienenstock E., Fogelman Soulié F. and Weisbuch G. (Springer Verlag, Berlin, 1986).

[46] Levine D.S., *Introduction to Neural and Cognitive Modelling.* (Erlbaum, Hillsdale, 1991).

[47] Luenberger D.G. *Introduction to Dynamic Systems.* (Wiley, New York, 1979).

[48] McCulloch, W.S., Pitts, W., A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Phys.* **5** (1943) 115–133.

[49] MacGregor R.J., *Neural and Brain Modelling.* (Academic Press, New York, 1987).

[50] Magnus J.R. and Neudecker H., *Matrix Differential Calculus with Application in Statistics and Econometrics.* (Wiley, New York, 1988).

[51] Mardsen J.E. and Tromba A.J., *Vector Calculus.* (Freeman, San Francisco, 1988).

[52] Micchelli C.A., Interpolation of scattered data: Distances matrices and conditionally positive definite functions. *Constructive Approximations* (1986) **2** 11–32.

[53] Minsky M.L. and Papert S.A., *Perceptrons.* (MIT Press, Cambridge, 1969).

[54] Moussouris J., Gibbs and Markov random systems with constraints. *Journal of Statistical Physics* **10** (1974) 11–13.

[55] Muller B. and Reinhardt J., *Neural Networks.* (Springer Verlag, Berlin, 1990).

[56] Nilsson N.J., *Learning Machines.* (MacGraw-Hill, New York, 1965).

[57] O'Toole A.J. and Abdi, H., Connectionist approaches to visually based feature extraction. In *Advances in Cognitive Psychology* **2**, ed. by Tiberghien G. (Wiley, London, 1989) pp. 124–140.

[58] O'Toole A.J., Deffenbacher K., Abdi H. and Bartlett J.C., Simulating the 'other-race' effect as a problem in perceptual learning. *Connection Sci.* **3** (1991) 163–178.

[59] Park I. and Sandberg I.W., Universal approximation using radial-basis function networks. *Neural Computations.* **3** (1991) 246–257.

[60] Parker D.B., Learning logic. Technical report TR-47, Center for Computational Research in Economics and Management Science, (Massachusetts Institute of Technology, Cambridge MA, 1985).

[61] Pierre D.A., *Optimization Theory with Applications.* (Wiley, New York, 1969).

[62] Perez J.C., *La Révolution des Ordinateurs Neuronaux.* (Hermès, Paris, 1990).

[63] Perko L., *Differential Equations and Dynamical Systems.* (Springer-Verlag, Berlin, 1991).

[64] Poggio T. and Girosi F., Networks for approximation and learning. *Proceedings of the IEEE* **78** (1990) 1481–1497.

[65] Powell M.J., Radial basis functions for multivariable interpolation: A review. *IMA conference on algorithms for the approximation of functions and data* (RMCS shrivenham 1985).

[66] Quinlan P.T., *Connectionism and Psychology.* (University of Chicago Press, Chicago, 1991).

[67] Rosenberg C.R., Sejnowski T.J., Parallel networks that learn to pronounce English text. *Complex Systems* (1987) **1** 145–168.

[68] Rosenblatt F., The perceptron: a perceiving and recognizing automation (projet PARA), Cornell Aeronautical Laboratory Report, 85-460-1 1957.

[69] Rosenblatt F., The perceptron: a probabilistic model for information storage and organisation in the brain. *Psychological Review* **65** (1958) 386-408.

[70] Rosenblatt F., *Principles of Neurodynamics.* (Spartan Books, Washington, 1961).

[71] Rumelhart D.E. and McClelland J.L., *Parallel Distributed Processing.* (MIT Press, Cambridge, 1986).

[72] Rumelhart D.E., Hinton G.E. and Williams R.J., Learning internal representations by error propagation. In *Parallel Distributed Processing*, ed. by Rumelhart D.E. and McClelland J.L. (MIT Press, Cambridge, 1986).

[73] Sánchez D.A., *Ordinary Differential Equations and Stability Theory: An Introduction.* (Freeman, San-Francisco, 1968).

[74] Searle S.R., *Matrix Algebra Useful for Statistics.* (Wiley, New York, 1982).

[75] Sebestyen G.S., *Decision Making Processes in Pattern Recognition.* (Macmillan, New York, 1962).

[76] Serra R. and Zanarini G., *Complex Systems and Cognitive Processes.* (Springer-Verlag, Berlin, 1990).

[77] Seydel R., *From Equilibrium to Chaos.* (Elsevier, New York, 1988).

[78] Siarry P. and Dreyfus G., *La Méthode du Recuit Simulé.* (I.D.S.E.T., Paris, 1988).

[79] Simpson P.K., *Artificial Neural Systems.* (Pergamon press, New York, 1990).

[80] Souček B., *Neural and Concurrent Real-Time Systems.* (Wiley, New York, 1989).

[81] Strang G., *Introduction to Applied Mathematics.* (Wellesley-press, Cambridge, 1986).

[82] Weisburg G., *Complex Systems Dynamics.* (Addison-Wesley, Reading, 1991).

[83] Werbos P.J., *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* (Doctoral Dissertation Thesis, Havard University, 1974).

[84] Widrow B. and Hoff M.E., Adaptative switching circuits. *1960 IRE WESCON Convention Records* (1960) 96–104.

[85] Widrow B. and Stearns S., *Adaptive Signal Processing.* (Prentice-Hall, New-York, 1985).

[86] Zeidenberg M., *Neural Network in Artificial Intelligence.* (Ellis Horwood, Chichester, 1990).