Perceptrons, past and present

G. Dreyfus*, L. Personnaz*, G. Toulouse**

**École Supérieure de Physique et de Chimie Industrielles*
Laboratoire d'Électronique
10, ue Vauquelin
F - 75005 Paris
Tel.: (33) 01 40 79 45 41
Fax: (33) 01 40 79 44 25
e-mail: dreyfus@neurones.espci.fr, personna@neurones.espci.fr

***École Normale Supérieure*
Laboratoire de Physique
24 rue Lhomond
F - 75005 Paris
Tel.: (33) 01 44 32 34 87
Fax (33) 01 43 36 76 66
e-mail: toulouse@physique.ens.fr

Running head: Perceptrons, past and present

Abstract

The story of Perceptrons is not unlike that of aeronautics. Although man drew its initial inspiration from the flight of birds, airplanes really came of age with the development of a deep knowledge in hydrodynamics and flight mechanics. Similarly, Perceptrons were invented in the 1960's with a view to mimicking the brain, but their real use in engineering is only at its starting point. In the present survey, we first describe briefly the state-of-the art of neural nets, emphasizing their basic mathematical properties and their fields of applications; the second part of the presentation describes the evolution of the ideas, from the first attempts in the area of pattern recognition, to more recent developments in nonlinear process modeling and control.

Introduction

The development of networks of formal neurons (including Perceptron structures) in the field of engineering started in the 1960's, and was virtually abandoned in 1970; it was revived in the 1980's. One salient feature of the evolution of the field in recent years is the recognition that, from the point of view of engineers, physicists or computer scientists, the biological metaphor is essentially irrelevant: Perceptrons have mathematical properties of their own, which make them useful in a wide variety of areas, irrespective of their biological origin. Of course this observation, helpful for practitioners as it is, is not meant to induce anyone to ignore the inspiration drawn from biology, not the potential for fruitful modeling of biological computation, as described elsewhere in this volume.

The first part of this survey is devoted to a cursory description of the present view of networks of formal neurons: their basic mathematical property is explained, and the main tasks that neural networks can perform are presented in that perspective. The second part of the paper will be an attempt at describing the main steps along the path that led to the present state.

Where Do We Stand ?

Formal neurons

A formal neuron (hereinafter termed simply "neuron") is a simple mathematical operator; in the vast majority of applications, it is expressed as a few lines of software; if stringent real-time operation constraints exist, it can be implemented as an electronic circuit. The inputs of a neuron are numbers which are processed in the following way: the neuron computes its output as a non-linear function of the weighted sum of its inputs (see Figure 1). The inputs may be either the outputs of other neurons, or exogenous signals. The weights (sometimes termed "synaptic weights"), the inputs and the outputs are, in general, real numbers. The non-linear function must be bounded: it may be a step function, a sigmoid (e.g. tanh) function, a gaussian, etc.

Perceptrons

Although the computational power of a single neuron is very low, networks of such neurons have very useful properties. Before describing them, we first introduce an important distinction between two classes of network architectures:

- *Feedforward neural networks* are networks in which information flows from inputs to outputs, without any feedback (Figure 2); they are static systems: the input-output relationships are in the form of non-linear algebraic equations; the outputs at a given time depend only on the inputs at the same time;

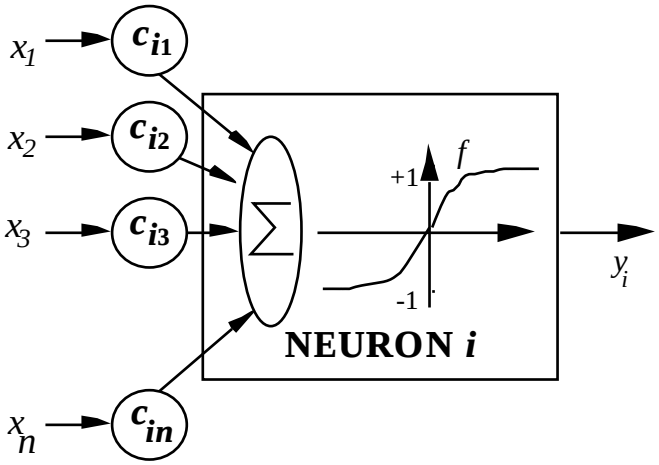

FIGURE 1

The output y_i of neuron i is a nonlinear function f of the sum of its inputs x_j weighted by the synaptic weights c_{ij}:

$$y_i = f\left[\sum_{j=1}^{n} c_{ij}\, x_j\right].$$

- *Feedback neural networks* (also termed recurrent networks) are networks in which information may be fed back from the output of a neuron to its input, possibly via other neurons; such networks are dynamic systems: the input-output relationships are in the form of non-linear differential equations (or difference equations, see Figure 3); the outputs depend on the past values of the inputs of the network.

In feedforward multilayer networks, intermediate ("hidden") neurons process information conveyed by the inputs, and transmit their results to the output neuron(s). Such networks are used as automatic classifiers for pattern recognition, static nonlinear models of processes, or nonlinear transverse filters. Figure 2 shows a typical structure used for classification, in which the hidden neurons are arranged in a layer: connections exist between the inputs and the hidden neurons, and between the hidden neurons and the output neurons.

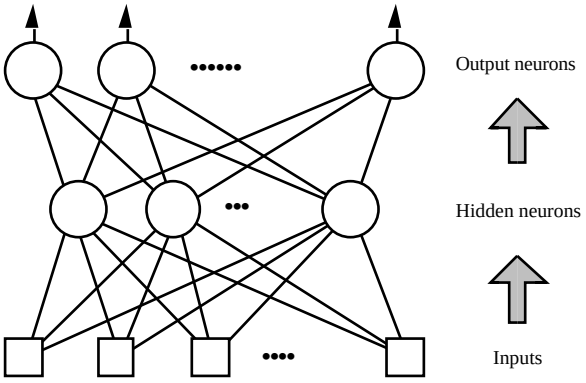

FIGURE 2

A multilayer Perceptron (feedforward network)

Feedback networks can be used for the nonlinear dynamic modeling and control of processes. Figure 3 shows a discrete-time feedback neural network. Since there are loops within the network, delays must be present in order for the system to be causal. It can be shown actually that any feedback neural network, however complex, can be put into a canonical form (Figure 4) which is made of a feedforward network (usually a multilayer perceptron) having some outputs (called state outputs) fed back to the inputs.

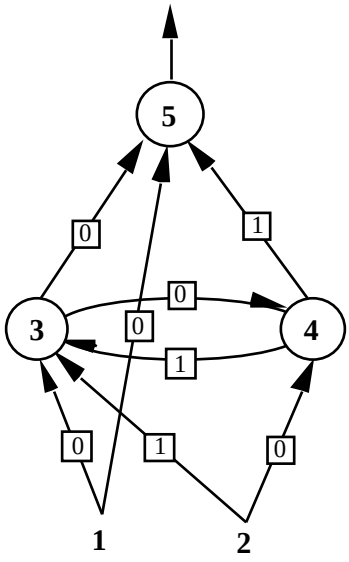

FIGURE 3
A discrete-time feedback network; the numbers in the squares denote time delays.

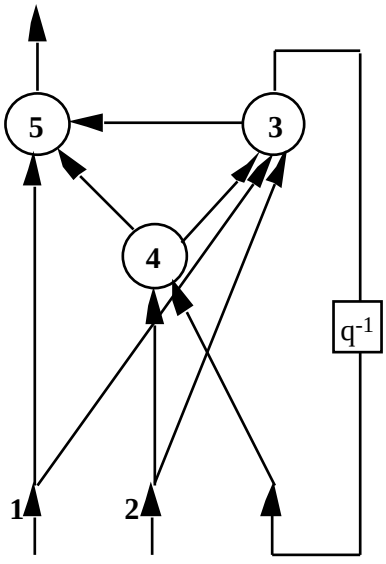

FIGURE 4
The canonical form of the discrete-time recurrent network of Figure 3. The symbol q^{-1} denotes a unit time delay.

Note that along time an evolution of terminology occurred. Originally, the word Perceptron was restricted to an elementary and specific structure; it was subsequently extended to include all feedforward nets. Our use of the term will be at times even more liberal.

Fundamental property of multilayer perceptrons and areas of applications

Multilayer perceptrons have a remarkable property: they are *parsimonious universal approximators* (or *regression estimators*) (Hornik et al., 1994).

The *approximation* property may be stated as follows: *any sufficiently regular nonlinear function can be approximated with arbitrary accuracy by a feedforward network having a single layer with a finite number of hidden neurons with sigmoidal nonlinearity and a "linear" output neuron.* This property is not specific to neural nets: polynomials, for instance, are universal approximators; so are wavelets, radial basis functions, and others.

The salient feature of neural networks is their *parsimony:* for a given accuracy, the number of weights required is generally smaller for a "neural" approximation than for an approximation with most other usual approximators. More specifically, the number of weights grows linearly with the number of variables of the function, whereas it grows exponentially for polynomials for instance. In practice, perceptrons are often advantageous when the number of variables is larger than or equal to three.

In the vast majority of applications, the task of a multilayer perceptron is not the approximation of a *known* function. The problem of interest is actually the following: given a collection of values of the variables (the inputs of the network), and a collection of numbers (the desired values of the outputs) which are the corresponding values of an unknown nonlinear function, find a mathematical expression that "best" fits the data (i.e. minimizes the approximation error) and that "best" generalizes (i.e. interpolates) to new data. Because of their parsimonious universal approximation property, neural networks are excellent candidates to perform such a task. The problem then is to determine the values of the weights that allow the neural network to achieve that best fit: this is done through an algorithmic process called *training* or *supervised learning*. After training, the perceptron should ideally "capture" what is deterministic in the data. Note that this problem is a classical problem in statistics known as *nonlinear regression;* training is just the estimation of the parameters of a nonlinear regression.

In order to find the best nonlinear fit of the output to the data, the following steps must be taken :

(i) Choose the architecture of the network, i.e. the network inputs (the relevant variables), the topology and size of the network: this determines a family of non-linear functions with unknown parameters (the weights of the network) which are candidates for performing the data fitting.

(ii) Train the network, i.e. compute the set of weight values that minimize the approximation error over the data set used for training (training set).

(iii) Assess the performance of the network on a data set (called test set) which is distinct from the training set, but which stems from the same population.

The fundamental property of perceptrons tells us that a neural network with a sufficiently large number of hidden units can fit any nonlinear function; however, a network with too much "flexibility", i.e. too large a number of adjustable weights, will wiggle unnecessarily between the data points, hence will generalize poorly. Conversely, a network which is too "stiff" may be unable to fit even the training data satisfactorily. This is illustrated on Figure 5: the red dots are the measurements of the noisy output of the process which are used for training the network, and the black line is the output of the network after training; it is clear that the output of the network with the smaller number of neurons is a more satisfactory approximation of the deterministic part of the process output than the output of the larger network.

Therefore, the goal of the model designer is to find the best tradeoff between the accuracy of the fit to the training data and the ability of the network to generalize. Both constructive techniques (starting with a very small network and increasing its complexity step by step) (see for instance Nadal et al. 1989, Knerr et al. 1990) and pruning techniques (decreasing the complexity of an oversize network) (see for instance Hassibi et al. 1993) have been widely used, but rigorous statistical methods provide almost optimal networks (Leontaritis et al. 1987, Urbani et al. 1992). Regularization techniques, imposing constraints on the magnitudes of the weights, have also been investigated in depth (Poggio et al. 1990)

The areas of applications of multilayer perceptrons can be derived in a straightforward fashion:

- **static non-linear modeling of processes:** there is a wealth of engineering problems, in a wide variety of areas, which are amenable to data fitting (see examples below): since perceptrons are parsimonious universal approximators, they are excellent candidates for performing such tasks, provided the available data is appropriate;
- **dynamic non-linear modeling of processes:** as a consequence of the universal approximation property, feedback neural networks can be trained to be models of a very large class of nonlinear dynamic processes;
- **process control**: a process controller computes the control signals that are necessary in order to convey to the process a prescribed dynamics; when the process is nonlinear, a neural network can be trained to implement the necessary nonlinear function;

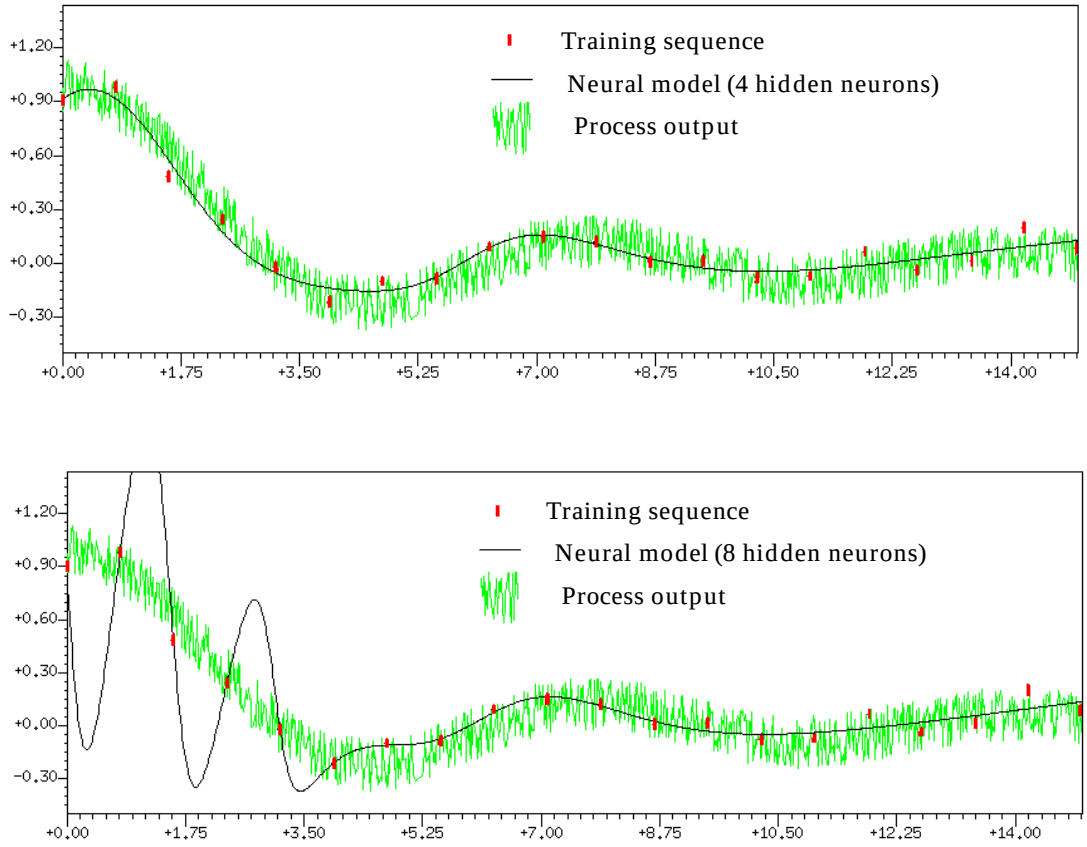

FIGURE 5

The benefits of parsimony : the most parsimonious network, featuring 4 hidden neurons (13 weights), has better generalization properties than the network with 8 hidden neurons (25 weights).

- **classification**: assume that patterns are to be classified into one of two classes, A or B, each pattern being described by a set of variables (e.g. the intensity of the pixels of an image); a (usually human) supervisor sets the values of the desired outputs to +1 for all patterns that he recognizes as belonging to class A, and to 0 for all patterns of class B. Neural networks are good candidates for fitting a function to the set of desired outputs, and one can prove that this function is an estimate of the probability that the unknown pattern belongs to class A. Thus, perceptrons provide a very rich information, much richer than a simple binary response would be.

  It should be noticed that the classification abilities of neural nets are a relatively indirect consequence of the universal approximation property. Nevertheless, neural networks have been extensively studied and used in this context, to the extent that perceptrons are mostly known as classifiers. We shall see below the historical reasons of this.
- **optimization**: optimization is the only area of applications of neural nets that does not actually take advantage of the universal approximation property. Given a

discrete-time recurrent neural network made of binary units (i.e. of neurons whose non-linear activation function is a step), it is usually possible to define a Lyapunov function (or energy function) for this network, i.e. a scalar function of the state of the network which has the following property: when the network is left to evolve under its own dynamics, the Lyapunov function decreases at each time step, until the network reaches a stable state. Thus, the dynamics of the network leads to a minimum of the Liapunov function. This property can be used for optimization purposes in the following way: given a function *J* to be optimized, if it is possible to design a neural network whose Liapunov function is precisely function *J*, then this network has the ability, if left to evolve under its free dynamics, of finding the minimum of the cost function (Hopfield et al. 1984).

The supervised training of neural networks

As mentioned above, the supervised training of neural networks is the task whereby the weights of a network are computed in such a way that, for each input pattern of the training set, the output of the network be "as close as possible" to the corresponding desired output; the latter is the class to which the pattern belongs (in the case of classification), or the measured output of the process to be modeled (in the case of process modeling). Therefore, a "distance" between the network output (which depends on the weights *c)* and the desired output must be defined; usually, it is defined as the sum (termed "cost function"), over all examples *k*, of the square of the difference between the actual output $y^k(c)$ and the desired output d^k

$$J(c) = \frac{1}{2} \sum_{k=1}^{N} \left[d^k - y^k\right]^2,$$

where *N* is the number of patterns of the training set (also termed *examples*).

If the network has more than one output, this is summed over all outputs. Thus, training consists in minimizing the cost function with respect to the weights *c*. This optimization is performed by standard nonlinear optimization methods, which make use, in an iterative fashion, of the gradient of the cost function; this gradient itself is computed by a simple method called "backpropagation" (which is described elsewhere in this book). Before training, the weights are assigned random values; during training, the weights are updated iteratively, so that the cost function *J* decreases, and training is terminated when a satisfactory tradeoff is reached between accuracy on the training set and generalization ability measured on the test set, distinct from the training set.

Some typical applications of neural networks

Static non-linear modeling of processes**.** The application of perceptrons in the area of Quantitative Structure-Activity Relationship (QSAR) is a typical example of the ability of neural networks to fit data. The goal is the prediction of a chemical property (expressed by a real number) of a molecule from a set of descriptors (also real numbers) such as its molecular weight, its dipole moment, its volume, or any other relevant quantity (Figure 6). A variety of chemical properties can be thus estimated: aqueous

solubility, partition coefficients, etc. Some descriptors can be measured, others can be computed *ab initio* by molecular simulation software tools. The results obtained by neural networks in this area are consistently superior to those obtained by other regression techniques on the same databases.

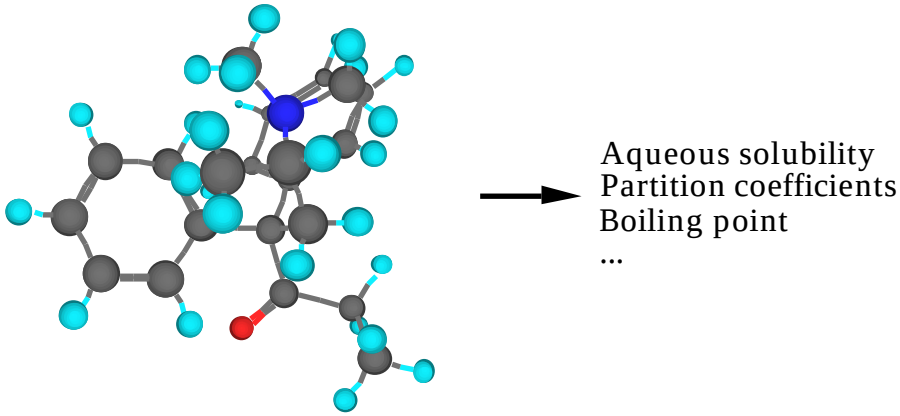

FIGURE 6
The goals of QSAR

This approach can be extended to other areas: prediction of the properties of composite materials from their composition, prediction of pharmacological properties of molecules, etc. Perceptrons can be regarded as an aid to the design of new molecules, new materials, etc.; they can be extremely valuable by saving the labor and cost of synthesizing a new molecule or a new material which can be predicted not to have the desired properties.

Dynamic non-linear modeling of processes. As mentioned above, feedback neural networks, which obey non-linear difference equations, can be used for dynamic non-linear process modeling. These applications are given as examples of the quite general procedure of prediction through interpolation.

The aim of the modeling is to find a mathematical model, such as a neural network, whose response to any input signal (usually termed control signal in the field of process control) is identical to the response that the process would exhibit in the absence of noise (Figure 7).

Very frequently, some knowledge on the process is available in the form of non-linear differential equations, deriving from a physical (or chemical, economical, financial, etc.) analysis of the process. In such a case, it would be wasteful to throw away this valuable information. One of the most remarkable features of neural networks is the fact that they are not necessarily "black boxes": prior knowledge can be used in order to give an appropriate structure to the network, and determine the value of some of its parameters.

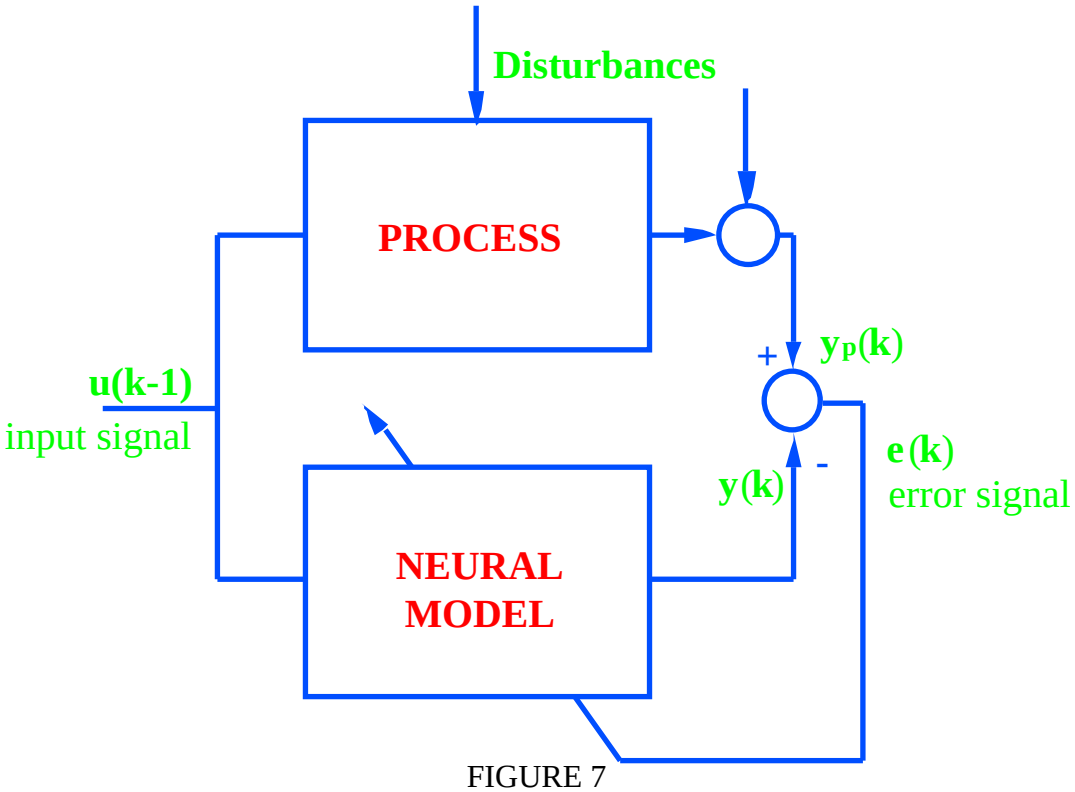

FIGURE 7
Principle of black-box modeling

Such *knowledge-based neural models* are in use in industrial applications (Ploix et al. 1996).

The prediction of time series from past values is related to process modeling, except for the fact that control inputs may not exist: the inputs of the neural predictor are an appropriate set of past values of the time series itself, and possibly a set of past values of the predictions; in the latter case, the predictor is a feedback network. Financial and economic forecasting are typical areas of applications of time series prediction by neural nets (Weigend et al. 1994).

Process models can be used in a variety of applications: fault detection, personnel training, computer-aided design, etc. The use of models for process control is presented in the next section.

Process control. Control is the action whereby a given dynamic behavior is conveyed to a process. When a nonlinear model is necessary, neural networks are excellent candidates for performing such a task. The design of a neural process controller requires two ingredients:

- a reference model which computes the desired response of the process to the setpoint sequence ;
- a "neural" model, as described above.

The "neural" controller is trained to compute the control signal to be applied to the model in order to obtain the response required by the reference model.
For instance, the process output may be the heading of an autonomous vehicle and the control variable may be the angle of rotation of its driving wheel. Then, if the setpoint is the orientation to be taken, the reference model describes how the vehicle should behave in response to a change in the setpoint signal, depending on the speed of the vehicle for instance; the neural controller will compute the sequence of angles of rotation of the driving wheel which is necessary for the vehicle to reach the desired orientation with the desired dynamic behavior. The French company SAGEM, in a cooperative work with ESPCI, has designed and operated a fully autonomous four-wheel-drive vehicle whose driving wheel, throttle and brakes are controlled by neural networks (Figure 8) (Rivals et al. 1994).

FIGURE 8
An autonomous vehicle, fully piloted by neural networks (courtesy SAGEM)

Classification. In a classification problem, unknown patterns (in a general sense, e.g. handwritten characters, time series, phonemes, etc.) must be assigned to appropriate classes. The patterns are represented by a set of descriptors (real numbers) which are supposed to describe the pattern unambiguously. A classifier either assigns an unknown pattern to a class, or acknowledges that it cannot assign a class, because the pattern is

ambiguous or too dissimilar from the examples. The performances of a classifier depend strongly on the representation of the patterns, which results from preprocessing steps acting upon the raw data; in picture processing for instance, typical preprocessing includes filtering, thinning, contour extraction, size normalization, etc. If the pattern representation is appropriately discriminating, if the network is appropriately designed, and if the training set is a representative sample of the patterns to be processed, neural networks yield recognition performances which are similar to those of the best non-neural classifiers; however, their implementation as special-purpose integrated circuits or on parallel processors is easier than for most classifiers of equivalent quality. There is a wealth of examples of applications of neural networks in this area, which has been investigated in great depth. Industrial applications have been very successful in the field of character recognition: as an example, the French Post Office runs automatic mail sorting machines which use neural networks (together with other methods) for the recognition of handwritten postal codes (Figure 9).

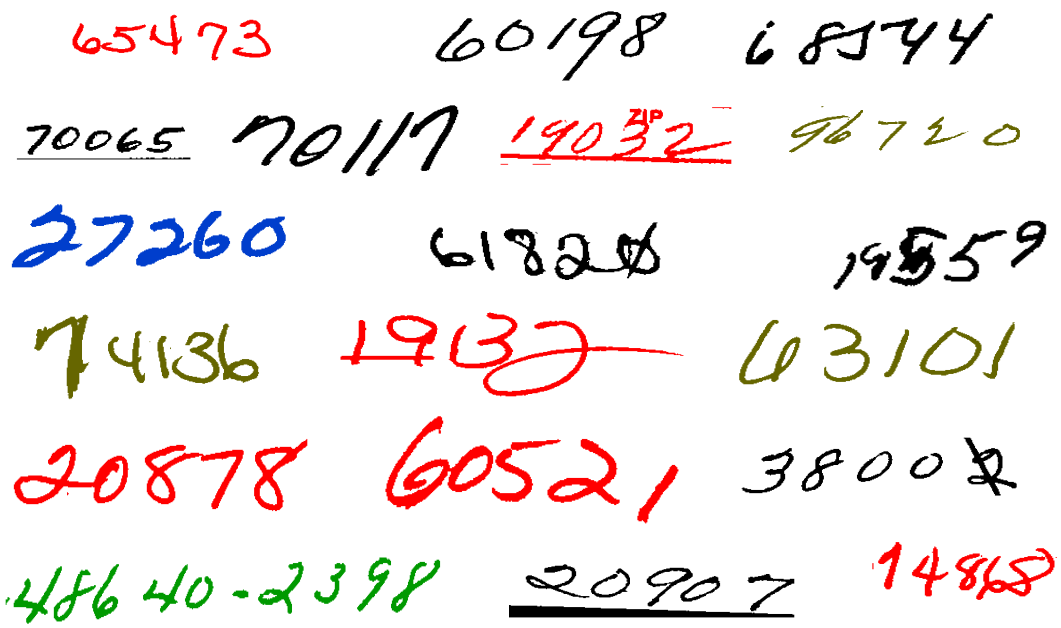

FIGURE 9
Examples of handwritten postal codes drawn from a data base available from the US Postal Service.

Similarly, neural networks are used as components of systems which read automatically the check amounts written in cursive handwriting (Figure 10). On-line recognition of cursive handwriting is also a promising field of applications (Guyon et al. 1996)

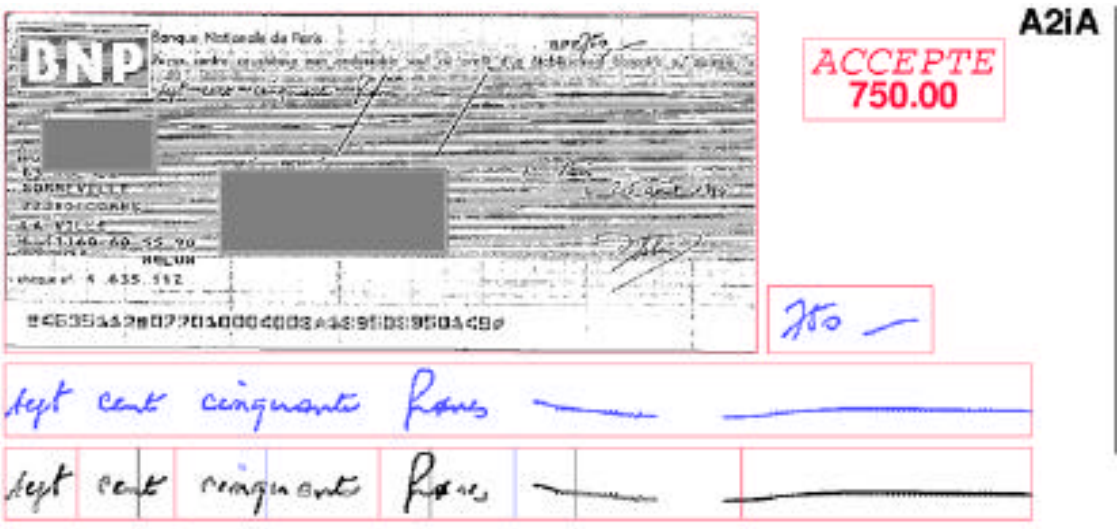

FIGURE 10
The automatic recognition of handwritten amounts on checks (courtesy A2iA).

Of course, classification problems arise also in a variety of applications which are quite different from visual pattern recognition. Perceptrons are thus used in industry for failure diagnosis, in banks for rating companies, etc. As an example, the French Caisse des Dépôts et Consignations uses a neural network for rating the capacity of towns to refund loans; each town is rated from A (best) to E (worse). Figure 11 shows a map with color-coded towns, each town being assigned the color of the most probable class.

How Did We Get There ?

Having summarized the basic principles of neural networks in engineering, we are going to show how the evolution of the ideas led to the present state of understanding of their mathematical properties and of their areas of applications. Historically, the early concepts of formal neurons (with binary or continuous output) appeared as abstractions in studies of the operation modes of the nervous systems (McCulloch and Pitts 1943), in particular their ability to perform binary and logical operations. In the fifties and sixties, these models started to appeal to engineers who wanted to find a response to the challenge of automatic invariant pattern recognition: how is it that very simple nervous systems are able to recognize an object, independently of its size, orientation, and background, a task which even to-day's most powerful computers are unable to perform. Thus, neural networks were first considered for applications in the field of pattern recognition, as evidenced by the name Perceptrons. The interest in neural networks was renewed in 1982 by the introduction of fully connected recurrent networks (known as Hopfield networks) which were described by equations analogous to those which describe magnetic systems (spin glasses) (Hopfield 1982); statistical physicists then went on a long way into the analysis of such systems, solving a rich variety of idealized

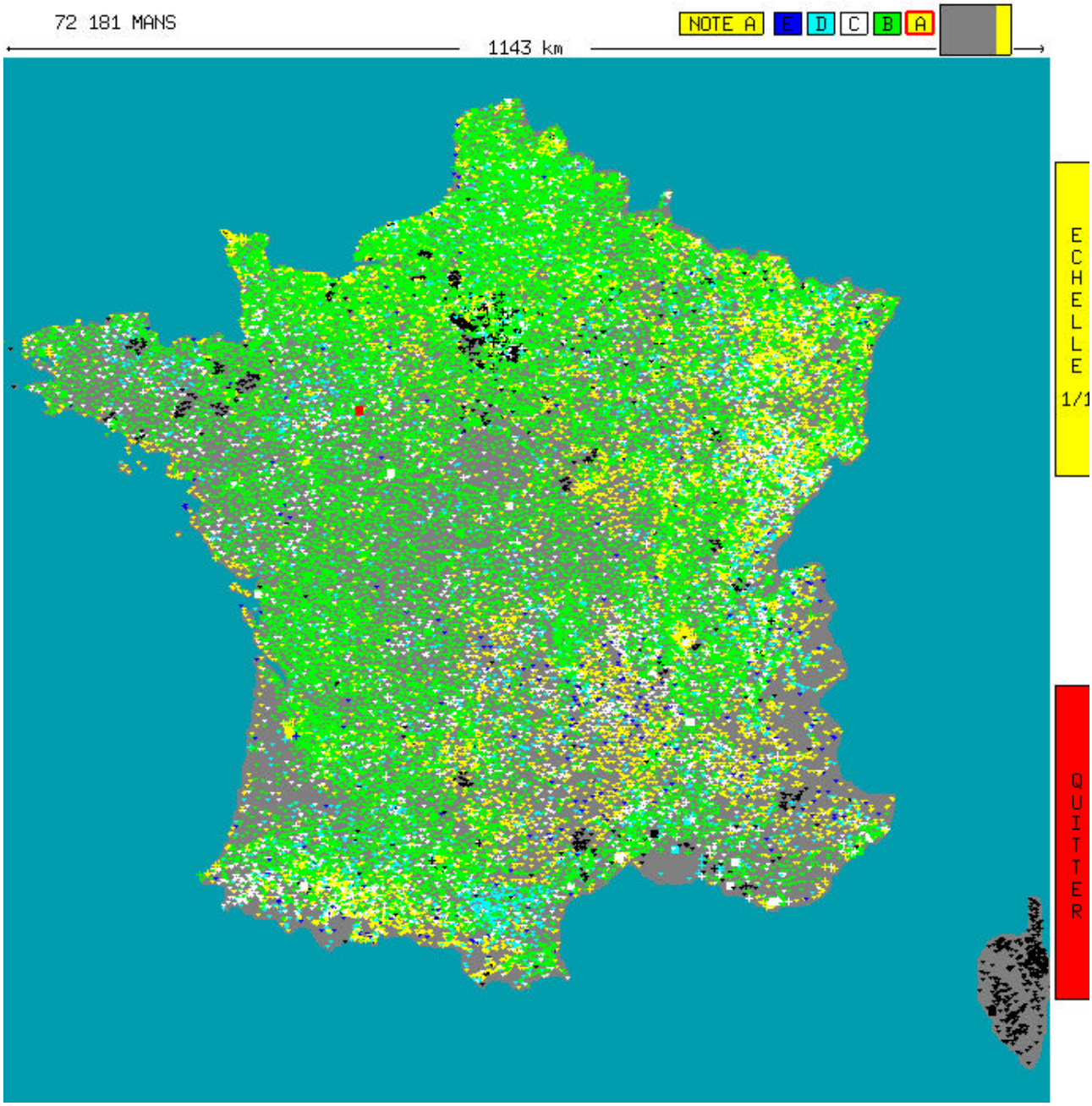

FIGURE 11
Financial analysis of townships

models for memory and learning, and into their use for the modeling of biological networks (see the appropriate chapters in the present Encyclopedia).

Since neural networks were first used for automatic classification, this section will emphasize neural networks viewed as classifiers, although the link between nonlinear regression (using the fundamental property of neural networks, i.e. function

approximation) and classification is by no means obvious, and was made clear only in recent years. After this reminder, more recent developments are described.

Classification

The problem of classification is that of assigning a pattern (described by a set of numbers called descriptors, regressors or features), to one of several classes. A typical example is that of recognizing handwritten numerals from the postal code, a problem which has become a classical benchmark for classifiers. There are two types of classifiers:

- *boundary classifiers* define boundaries between regions which are assigned to classes in the space of representations of the patterns to be classified (Figure 12); these boundaries are derived from a set of patterns whose class is provided by the supervisor; when an unknown pattern is input to the classifier, the latter makes a decision as to the class to which the pattern belongs, based on the position of the point representative of this pattern with respect to the boundaries;

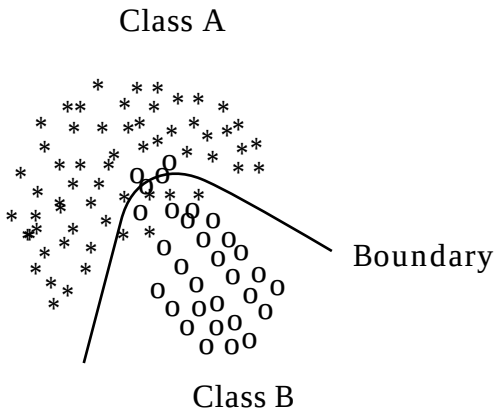

FIGURE 12

Representation of two classes in a 2-D space (each pattern has two descriptors). The crosses and dots are the examples.

- *probability classifiers* estimate the probabilities that a pattern belongs to each class; again, this estimate is based on a set of patterns whose class is known; when an unknown pattern is input to the classifier (Figure 13), the latter estimates the probability that it belongs to each class, but, in contradistinction to the boundary-oriented approach, it does not make a decision as to the class of the pattern; the final decision is produced by another component of the pattern recognition system, based, for instance, on information conveyed by several different classifiers using different representations of the patterns.

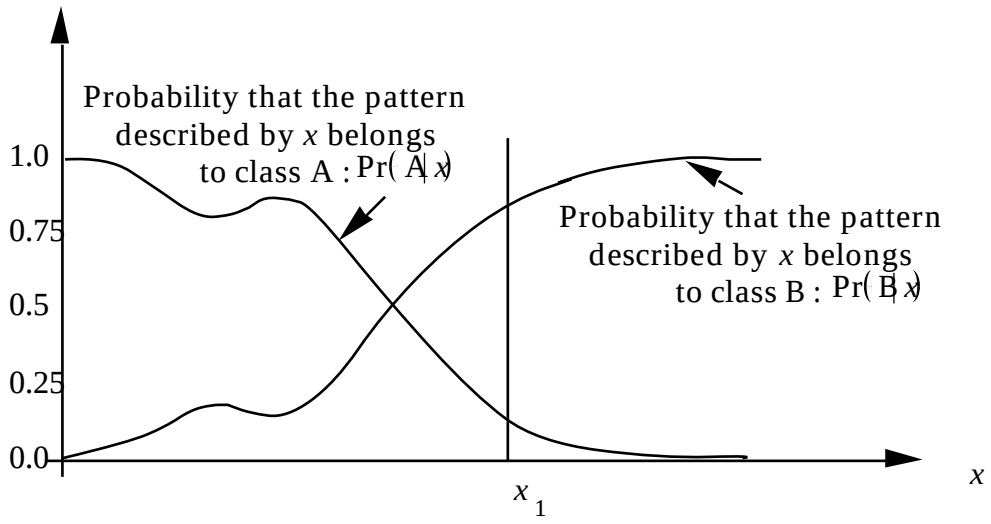

FIGURE 13

Curves showing the probability that a pattern described by descriptor x belongs to either class; the pattern described by $x = x_1$ has a 15% probability of belonging to class A and 85% probability of belonging to class B.

In the following, we first consider the boundary-oriented approach, and then turn to the probability-oriented approach.

The boundary-oriented approach to neural classification. We consider that a set of patterns is available, whose classes are known to an expert (or supervisor), supposedly without error; each pattern is defined by a problem-dependent *representation*, i.e. by a set of numbers (e.g. the set of pixel intensities of an image) which are assumed to be useful for the discrimination of the various classes in the application under consideration. From this set of patterns, it is desired to infer the boundaries between the regions representing each class in the space of representation.

The simplest possible boundary-oriented two-class classifier is a single binary neuron, or Perceptron, also known as *linear separator* (Figure 14). A binary neuron is a neuron whose non-linearity is a step function: its output y is given by $y = H\left[\sum_{i=0}^{n} c_i x_i\right] = H(v)$, where n is the number of features of the chosen representation, $x_0 = 1$, and H is the Heaviside step: the output of the neuron is equal to 1 if v is positive, and 0, if v is negative. The boundary is defined as follows: the weights c_i should be such that the output of the neuron be equal to 1 if the input pattern belongs to one of the classes, and be equal to zero if the input pattern belongs to the other class: the boundary between the classes is a hyperplane whose equation is $v = \sum_{i=0}^{n} c_i x_i = 0$ (Figure 15). The weights c_i are computed by applying a suitable training procedure (as described below) to a subset of the available set of patterns (the training set, made of labeled patterns). If the examples

of the training set can be completely separated, without classification errors, they are said to be *linearly separable*.

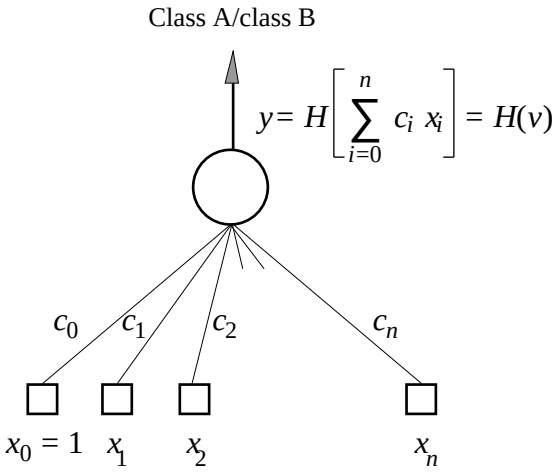

FIGURE 14
A binary neuron (or linear separator)

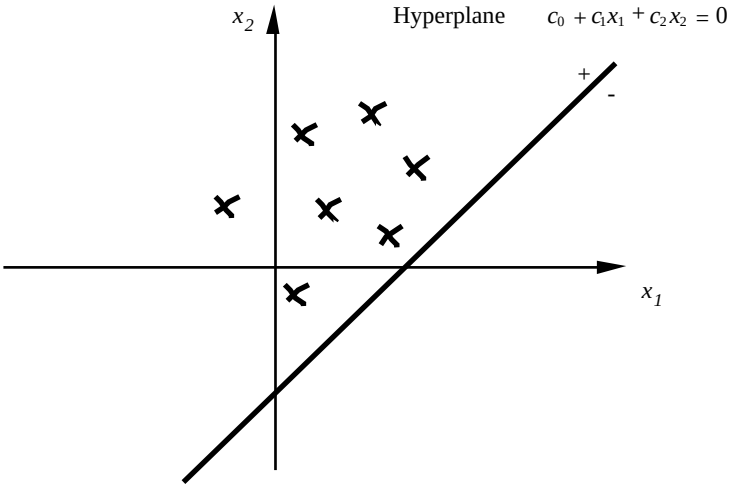

FIGURE 15
Separating hyperplane (in the case $n = 2$).

There are several supervised training algorithms for computing the weights c_i. The oldest one is known as the "Perceptron algorithm". The Perceptron algorithm is an iterative method, using a training set made of patterns labeled by a supervisor in the following way: an example belonging to class A is a pattern labeled with the desired value $d^k = 1$, an example belonging to class B is a pattern labeled with $d^k = 0$. At the k-th iteration, the k-th example is input to the neuron, and the corresponding potential v^k is computed; if the pattern is correctly classified, the weights are left unchanged; if the example is misclassified, then the weights are changed according to the Perceptron rule: $c_i(k) = c_i(k\text{-}1) + \mu\ (2\ d^k - 1)\ x_i^k$, where $c_i(k)$ is the value of weight c_i at iteration k, and the "learning rate" μ is a positive constant, smaller than or equal to 1. All examples are thus

presented in turn, in a random order; it can be proved that, if the training examples are linearly separable, this algorithm is guaranteed to converge to a valid solution within a finite number of iterations; conversely, if the examples are not linearly separable, the algorithm does not converge, and just runs on forever unless a maximum number of iterations is specified.

In practice, linear separability of distributions seldom occurs. A boundary between classes, even if the training examples are not linearly separable, can be found by minimizing the following cost function: $J = \frac{1}{2} \sum_{k=1}^{N} \left(2\, d^k - 1 - v^k\right)^2 = \sum_{k=1}^{N} J^k$, where N is the number of examples. This can be achieved by ordinary least-squares methods. The iterative *Widrow*-Hoff method leads to the same unique minimum; at each iteration, irrespective of the fact that the example presented is well classified or misclassified, the weights are changed in the direction opposite to the gradient, with respect to the weights, of J^k (the term of the cost function related to example k); thus $c_i(k) = c_i(k\text{-}1) + \mu\,(k)\,(2\, d^k - 1 - v^k)\, x_i^k$, where $\mu(k)$ is positive. It can be proved that, because J is a quadratic function of the weights, the algorithm finds the unique solution that minimizes J, if $\mu(k)$ vanishes. The boundary surface $v = 0$ thus obtained is a straight line if the pattern representation is two-dimensional ($n = 2$), a plane if $n = 3$, and a hyperplane if $n > 3$; this hyperplane, however, does not necessarily separate the two sets of examples, even if they are linearly separable. Nevertheless, if the distributions of the classes are unimodal, the solution may be satisfactory; if the classes are gaussian distributed, the boundary surface thus obtained is guaranteed to yield the minimal average classification error when the classifier is used (Bayes classifier).

The Widrow-Hoff algorithm (also termed *delta rule* in the neural literature) minimizes a function of the potential v of the neuron, but the quantity that one would really like to minimize is the sum of the squared distances between the desired output d^k and the actual output y^k : $J = \frac{1}{2} \sum_{k=1}^{N} \left(d^k - y^k\right)^2$. Since y^k is not linear with respect to the weights, the least-squares method is not applicable. Thus, gradient methods are generally used in order to minimize J; this implies that the output y of the neuron is differentiable with respect to the weights, which is not the case if the non-linearity of the neuron is a Heaviside step. Thus, the non-linearity of the neuron is no longer $H(v)$, but a sigmoid function $f(v)$ which is a "smooth" version of a step; for instance, one takes $y \equiv f(v) = \frac{1}{1 + e^{-v}}$. The minimization is performed by changing the weights in the direction opposite to the gradient of the term of the cost function related to example k with respect to the weights: $c_i(k) = c_i(k\text{-}1) + \mu\,(k)(d^k - y^k) f'\left(v^k\right)$, where f' is the derivative of the sigmoid function with respect to v. We refer to this rule as the *generalized delta rule*.

Once the neuron has been trained, the presentation of a pattern at its input results in a response which, in contrast to the previous cases, is not binary, but may take continuous values between 0 and +1 (Figure 16). Therefore, the neuron does not make a decision as to the class to which the pattern belongs; it is up to the user to set up a decision mechanism, the simplest one being that the pattern is assigned to class A if $y > 0.5$ and to class B if $y < 0.5$. With such a choice, the hyperplane defined by $y(x) = 0.5$ (or $v(x) = 0$) becomes the boundary between the two classes.

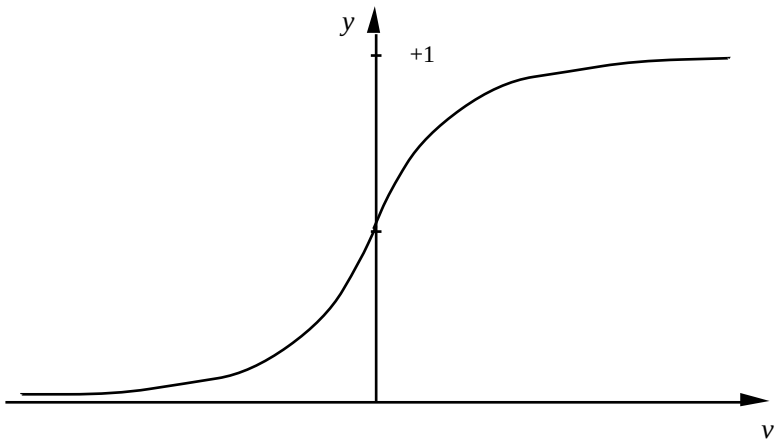

FIGURE 16
A sigmoid function $y = f(v)$

All the above approaches, whereby weights are computed from examples, lead to a crucial issue: how many examples are needed in order to obtain a meaningful boundary ? This is a difficult question, which has not yet received a complete, general answer. An interesting insight into this problem is given by the following result (Cover 1965). Consider a set of N points distributed randomly in an n-dimensional space, and make a random dichotomy in this set (i.e. assign each point at random to one of the two classes with probability 0.5). Cover proved that (i) if the number of examples N is smaller than the number n of descriptors ($N < n$), *any* dichotomy of the set of examples gives two subsets which are linearly separable, (ii) for a large dimensionality of the representation space (say, $n > 100$), if the number of examples N is smaller than or equal to twice the number of descriptors ($N \leq 2n$), then *almost any* dichotomy gives two subsets which are linearly separable. Therefore, if the number of examples is not very large with respect to the number of descriptors, a linear boundary may exist between the classes, but this boundary may be completely meaningless, thus leading to poor classification of patterns which are not in the training set. Quantitatively, it can be proved that the number of examples required for meaningful class separation grows *exponentially* with n; thus, compact representations using a small number of descriptors are highly desirable. This is sometimes called the *curse of dimensionality*.

Interestingly, all the above-mentioned techniques for training single neurons have been applied to Hopfield nets: since all neurons of such nets are connected together, they all have a similar input. As a consequence, it turns out that the behavior of single-layer perceptrons is highly relevant also for understanding the global dynamic behavior of these feedback nets.

We are now in a position to go from single neuron to neural networks. The single neuron is a linear separator. If the examples are not linearly separable, what can one do about it ? The answer is : use a feedforward multilayer network of neurons, which is able to determine boundary surfaces of arbitrary shape. The reason for this is best understood in the framework of the probability-oriented approach to classification.

The probability-oriented approach to neural classification. As mentioned above, the link between the classification properties of neural networks and their approximation property is the following: when using the above-mentioned generalized delta rule, the desired output of the neuron is either +1 (if the example belongs to class A) or 0 (if the example belongs to class B). Let us use the same idea when training a feedforward neural network. Then we know from the fundamental property of neural nets that the network, after training, will provide an approximation of the probability Pr(A½x) that a pattern described by a vector x belongs to A; this is performed by minimizing $J = \frac{1}{2} \sum_{k=1}^{N} \left(d^k - y^k\right)^2$ where $d^k = 1$ if the pattern belongs to A, $d^k = 0$ otherwise, and y^k is the network output; if the examples are well chosen and numerous enough, if the number of hidden neurons is appropriate, and if the training algorithm is efficient, this approximation is very good. This is illustrated on Figure 17 in the case of a one-dimensional pattern representation.

Once the probability has been estimated, a rule for the computation of the class boundary must be chosen; the most natural choice, which minimizes the error risk, is to choose the surface corresponding to equal probabilities of belonging to either class : $\Pr(A|x) = \Pr(B|x) = 0.5$. This relation defines a surface in the representation space. Since the feedforward network can approximate any function, the surface thus defined can have an arbitrary shape. Therefore, a feedforward network with one output neuron can define an arbitrary boundary surface between two classes.

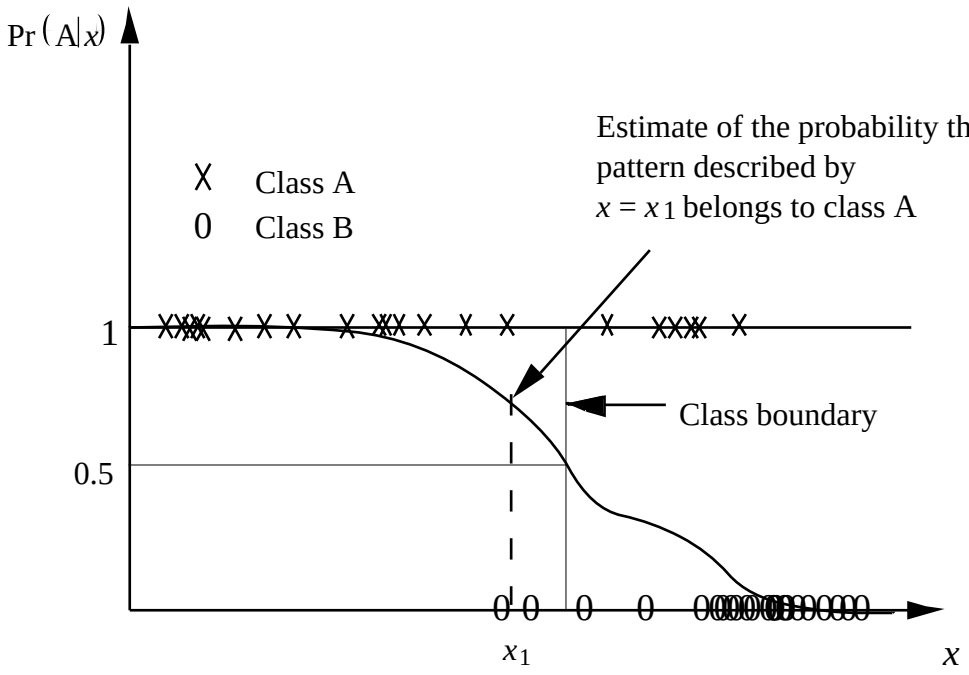

FIGURE 17

An estimate of the probability that a pattern described by *x* belongs to class A. Since the pattern is represented by a single descriptor *x*, the class boundary is a point on the *x*-axis corresponding to an estimated probability of 0.5; the boundary would be a line in two-dimensional space if the patterns were described by 2 descriptors, and a hypersurface in n-dimensional space if the patterns were described by n descriptors

The same principle can be extended to the determination of boundary surfaces between several classes. For a *C*-class problem, one generally uses a feedforward network with *C* outputs; during training, the desired output for output *c* is +1 if the example belongs to class *c*, and 0 otherwise; thus, the network uses a 1-out-of-*C* coding for the classes. In order to interpret the outputs as probabilities, the sum of the outputs must be constrained to be equal to 1 (Bridle 1990).

The above considerations are meant to explain why neural networks are able to compute boundary surfaces of arbitrary shape. However, it should be emphasized that there is much more to probability-oriented classification than just boundary surface determination. It is customary, in real applications, to have several classifiers make decisions from various sources of information (for instance, one may use several representations of the patterns, thus several classifiers using these representations). Therefore, the final decision is made from scores computed by the classifiers: typically, in a character recognition application, each classifier provides a list of hypotheses concerning the unknown character, ranked in order of decreasing probability. In banking applications, and more generally in computer-aided decision making applications, it may be useful to give a graded rating; in the above-mentioned application of neural nets to the

rating of financial capabilities of towns, a typical result is of the form "this town has 78% probability to belong to class B, 19% probability to belong to class C, and 1% probability to be in A, D and E". This is illustrated on Figure 18, which is a zoom on a specific area of France, where a bar graph shows the estimated probabilities of each town to belong to each class.

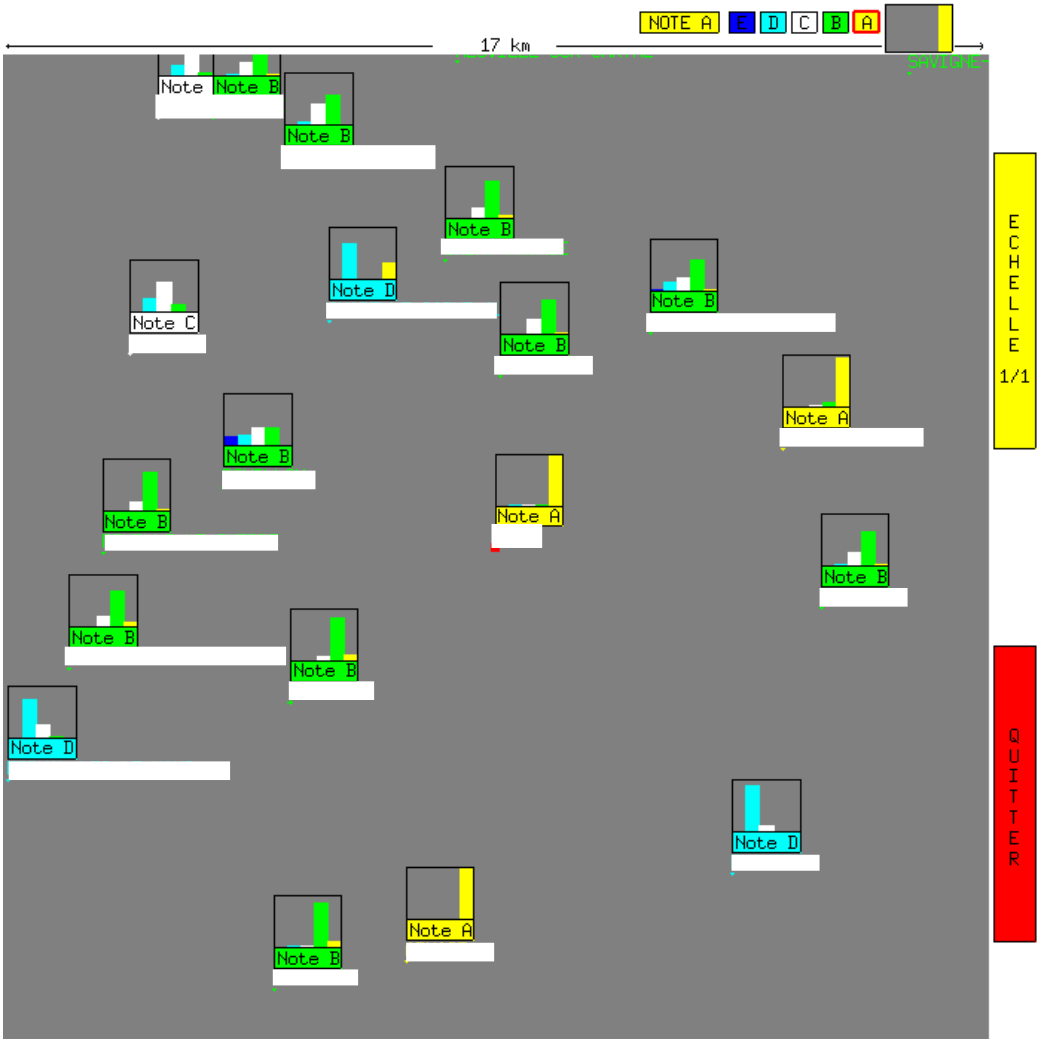

FIGURE 18

Zoom on a specific area of Figure 11, showing the probabilities of each town to belong to each class (the town names have been erased on purpose).

Dynamic modeling of processes

A process, whether physical, chemical, biological, economical, ecological, etc., can be modeled using two different sources of information: *knowledge* on the process, expressed mathematically by differential equations relating the physical, chemical, etc., variables of interest, and *measurements* performed on the process. Models which take advantage essentially of the knowledge are termed *knowledge-based* models; they usually have a small number of parameters which are determined from measurements performed on the process. Models which take essentially advantage of the

measurements performed on the process are termed *black-box* models. Knowledge-based models are usually preferred to black-box models because they are more intelligible, since the mathematical equations and the variables have a well-defined meaning. Very frequently, in practice, little or no usable prior knowledge is available, so that one has to infer a model essentially from the measurements alone; black-box models are also frequently used although a knowledge-based model exists, if the equations of the latter cannot be solved with the available computation resources.

A black-box model is primarily defined by its inputs, which are the variables that act on the process, and its outputs. Two kinds of inputs are to be distinguished: *control* inputs are those inputs on which one can purposefully act in order to control the process, and *disturbances*, which act on the process in an uncontrolled or even unmeasurable way; for instance, the electric current in a heating resistor is a control input for an oven whose temperature we wish to keep at a given value, while the unpredictable temperatures of the items that are loaded into the oven at unpredictable times are disturbances. The *outputs* of the process are the variables of interest which are measured: in the previous example, the temperature in the oven is the output of the process; the outputs may also be subject to disturbances, especially sensor noise.
A black-box model is intended to provide mathematical relations between the measurable input variables and the output variables; these relations involve a number of unknown parameters, which are estimated from the measurements made on the inputs and outputs. Since these measurements are affected by the disturbances and noise, the model should capture the *predictable* part of the variation of the output variables under the effect of the input variables. In other words, the prediction made by the model should be equal to the output that would have been measured on the process if no disturbance had been present. If the effect of the disturbance can be appropriately modeled as additive white noise, the variance of the error in the model's prediction of the process should be equal to the variance of the disturbance.

The black-box modeling of *linear* dynamic systems has been studied extensively for many years. If we make the assumption that there exists an unknown non-linear function which describes the process behavior, neural networks are natural candidates for modeling *non-linear* dynamic systems, since they can approximate any non-linear function. Indeed, most concepts in neural modeling are extensions of ideas developed in the framework of linear system modeling (Narendra et al. 1990).

We mentioned in the first part of the present survey that any dynamic neural network can be put in a canonical form, made of a feedforward network with state outputs fed back to its inputs (Nerrand et al. 1993). Therefore, we first consider feedforward networks as static models.
A static linear (affine) model is of the general form
$y = b_0 + b_1 x_1 + b_2 x_2 ... + b_n x_n$

where the x_i 's are the input variables; a first natural non-linear extension of the model is the following:

$y = c_0 + c_1 \varphi_1(x_1, \ldots, x_n) + c_2 \varphi_2(x_1, \ldots, x_n) + \ldots + c_m \varphi_m(x_1, \ldots, x_n)$

where the φ_i's are appropriately chosen non-linear functions of the variables. Thus, y is non-linear with respect to the inputs, but it is linear with respect to the coefficients c_i : the latter can therefore be estimated by the standard least-squares method from measured values of the output of the process y_p and of the values of φ_1, φ_2, ..., φ_m.

In traditional engineering, the non-linear functions φ_i's are monomials, so that the model is polynomial. The main advantage of the method is the fact that the output is linear with respect to the weights, so that standard least-squares methods can be used; the main drawback is the fact that polynomial models of several inputs and high degree tend to have a very large number of monomials, hence a very large number of parameters. This absence of parsimony makes the use of input selection methods mandatory.

In contradistinction, neural networks may provide a parsimonious model of a process in the region of input space in which measurements, used for training the network, have been performed. The output of a simple feedforward network with n inputs, m hidden neurons with activation function f and a linear output neuron will be typically of the form

$$y = c_0 + \sum_{i=1}^{m} \left[c_i \; f\left(\sum_{j=1}^{n} c_{ij} \, x_j + c_{i0} \right) \right]$$

Thus the output y is no longer linear with respect to the weights, so that the standard least-squares methods are no longer applicable: as mentioned in part 1 of the present survey, one has to resort to non-linear optimization methods such as gradient descent methods using the popular "backpropagation" algorithm for computing the gradient of the cost function.

The fact that the non-linear activation function f is very often a sigmoid stems from several factors. Historically, sigmoids were first used in classification, as "smooth" variants of the step function, as mentioned above; in addition, it turns out that, for function approximation, sigmoids are very easy to handle, especially during training. It must be noted, however, that networks of sigmoids are not always the best solution for a given problem; the optimal solution actually depends on the training set (number and distribution of examples used for training), on the architecture of the network (the unknown function, or a satisfactory approximation thereof, must be a member of the family of functions generated by the network), on the training algorithm, and on the model selection procedure used.

Neural networks with sigmoid activation functions are members of the general family of networks of "ridge" functions; a ridge function is a function of a weighted sum of the inputs (Figure 19).

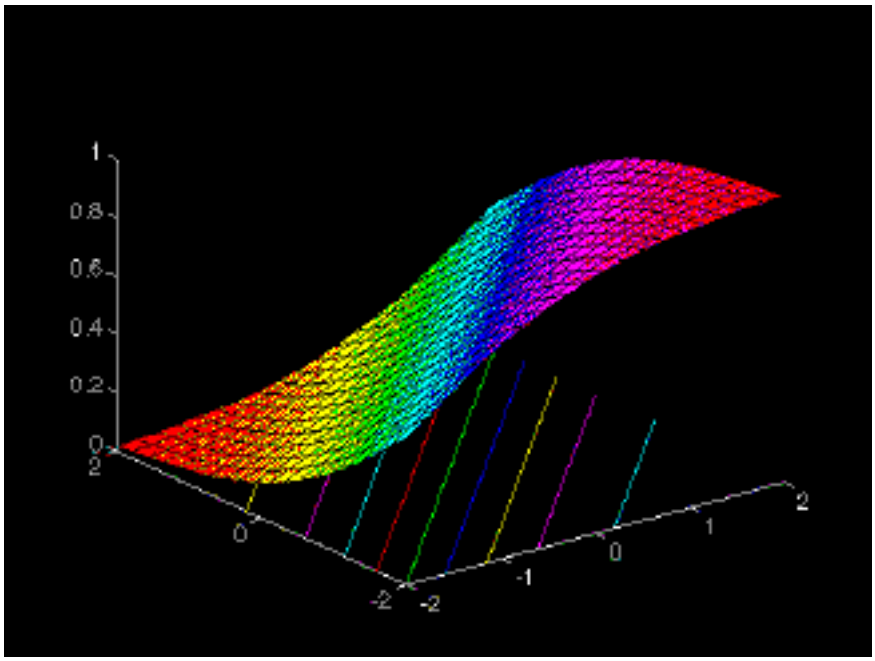

FIGURE 19

A sigmoid ridge function with a two-dimensional input space : $y = \dfrac{1}{1 + \exp\left[-(x_1 - x_2)\right]}$

Gaussian instead of sigmoid ridge functions may be used (Figure 20):

$$y = c_0 + \sum_{i=1}^{m} \left\{ c_i \quad exp\left[-\left(\sum_{j=1}^{n} c_{ij}\, x_j + c_{i0} \right)^2 \right] \right\} \quad .$$

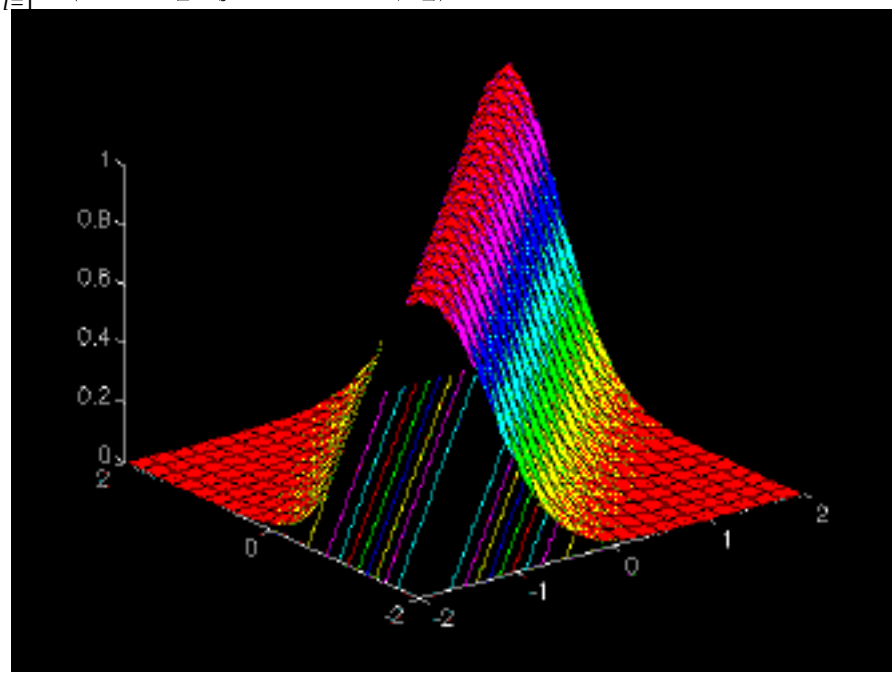

FIGURE 20

A gaussian ridge function $y = \exp\left[-(x_1 - x_2)^2\right]$.

The straight lines are the projections of the equal-value loci onto the (x_1 , x_2) plane

Wavelet ridge functions have also been investigated (Benveniste et al. 1996).

Non-ridge functions such as radial-basis functions (Figure 21) have also been widely used (Moody et al. 1989):

$$y = c_0 + \sum_{i=1}^{m} \left[c_i \prod_{j=1}^{n} exp\left(- \frac{(x_j - \mu_{ij})^2}{2\,\sigma_i^2} \right) \right]$$

where μ_{ij} is the jth coordinate of the center of gaussian i with width σ_i. Two approaches have been taken: choose *a priori* the number, centers and widths of the gaussians and estimate the weights c_i only (note that the output is linear with respect to the weights, so that standard least-squares methods can be used), or choose only the number of gaussians and estimate the weights, the positions of the centers and the widths (which requires a non-linear minimization).

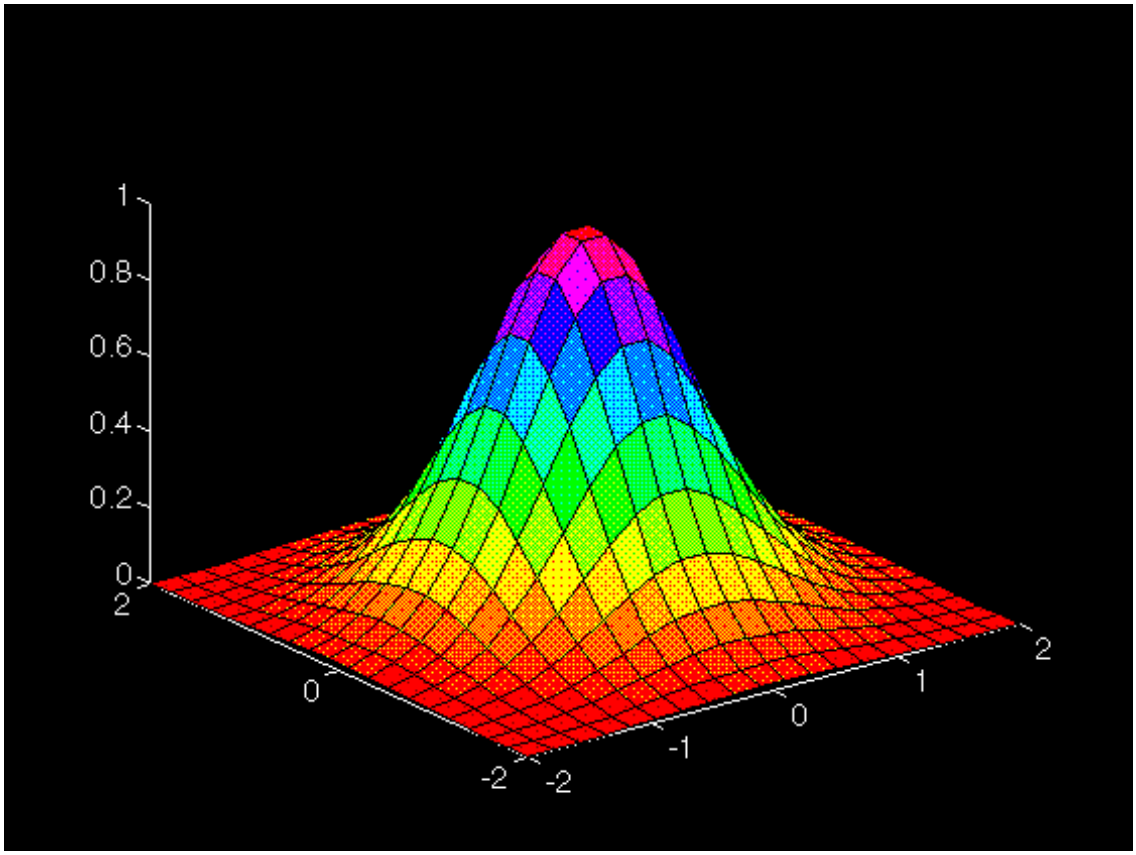

FIGURE 21
A radial basis function $y = \exp\left[-\left(x_1^2 + x_2^2\right)\right]$.

Note that in all cases where non-linear optimization is required, second-order gradient methods turn out to be a great improvement upon the simple gradient methods described above for single neurons. The description of second-order minimization methods falls

beyond the scope of this short survey; they are easily found in standard textbooks on nonlinear optimization or numerical analysis (see for instance Press et al. 1992).

Having discussed the various forms of the feedforward nets, we consider them now as a part of the canonical form of a recurrent network used as dynamic model. In such a model, the inputs and outputs are functions of time, and we restrict our attention here to discrete-time models: the behavior of the model is described by sequences of values sampled at discrete instants of time. Such models are thus described by difference equations of the type

$y(k) = \varphi[y(k\text{-}1), ..., y(k\text{-}n), u(k\text{-}1), ..., u(k\text{-}m)]$

where y is the output of the model and u is the control input (such a model is termed an *input-output* model). Thus, the function φ which determines the output at time k from the outputs and inputs at previous times completely defines the dynamics of the model; this function φ can be implemented by a feedforward neural network of any of the types mentioned above.

However, input-output models are not the most general form of dynamic models. The *state-space* form

$x(k) = \varphi[x(k\text{-}1), u(k\text{-}1)]$

$y(k) = \Psi[x(k)]$

where $x(.)$ is a vector of state variables, has been proved to be commonly more parsimonious than the input-output form (Levin et al. 1995).

In order to illustrate the non-linear modeling capabilities of neural networks and the difference between input-output and state-space models, we consider the modeling of the hydraulic actuator of a robot arm. This is a process with one input (the opening angle of the valve of the hydraulic circuit) and one output (the pressure in the hydraulic circuit). The training and test sequences are shown on Figure 22.

Using an input-output model, the best results were obtained with the second-order network shown on Figure 23, with two hidden neurons (sigmoid activation function). The result obtained after training is shown in Figure 24. The mean-square prediction error on the training sequence is equal to 0.085, whereas the prediction error on the test sequence is 0.30. This is a clear indication that the overfitting phenomenon occurs, due to the relatively small number of examples.

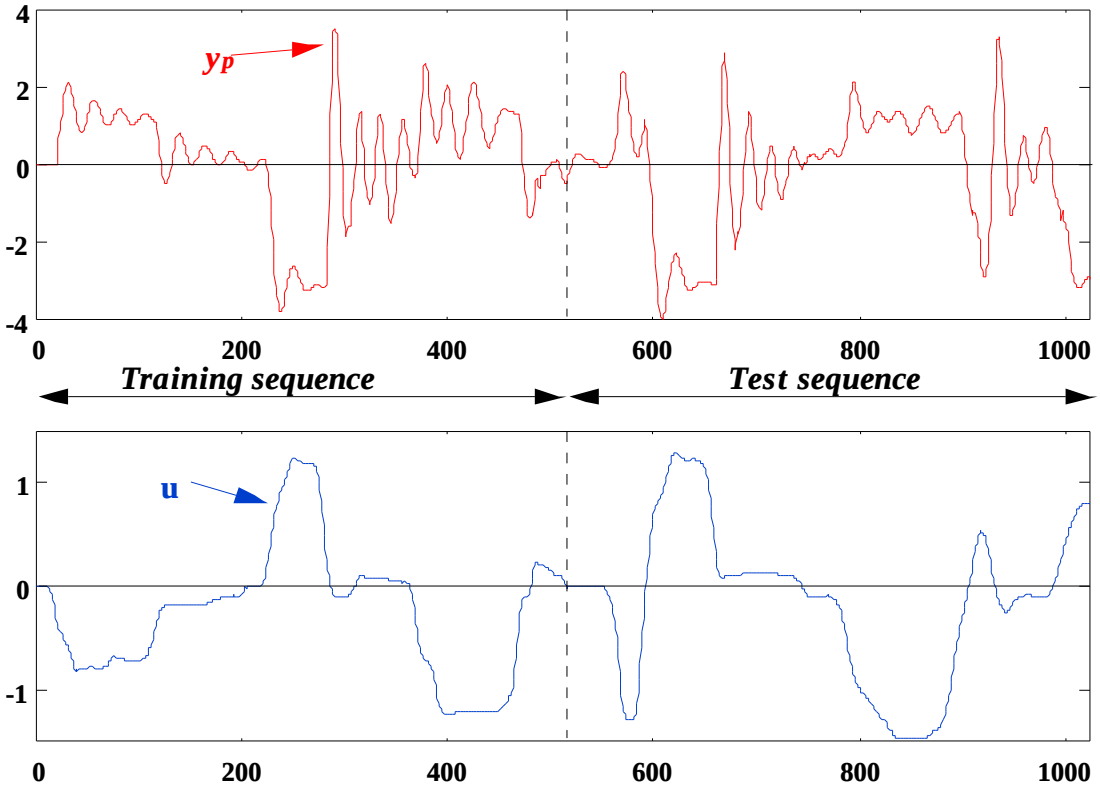

FIGURE 22

The training and test sequences for the modeling of a robot arm actuator; top: output sequence; bottom: control sequence.

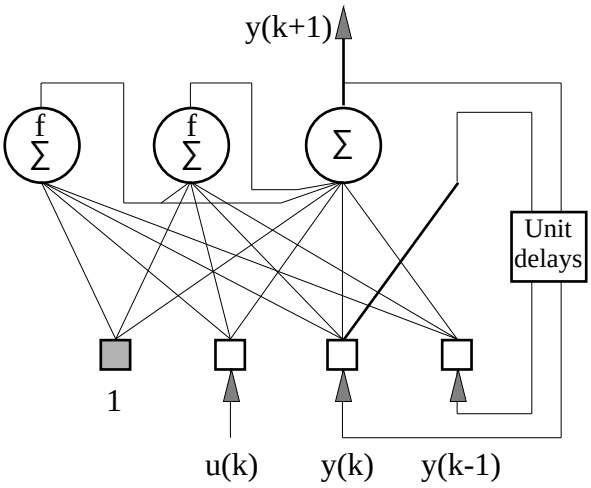

FIGURE 23

Input-output model used for the modeling of the hydraulic actuator of the robot arm

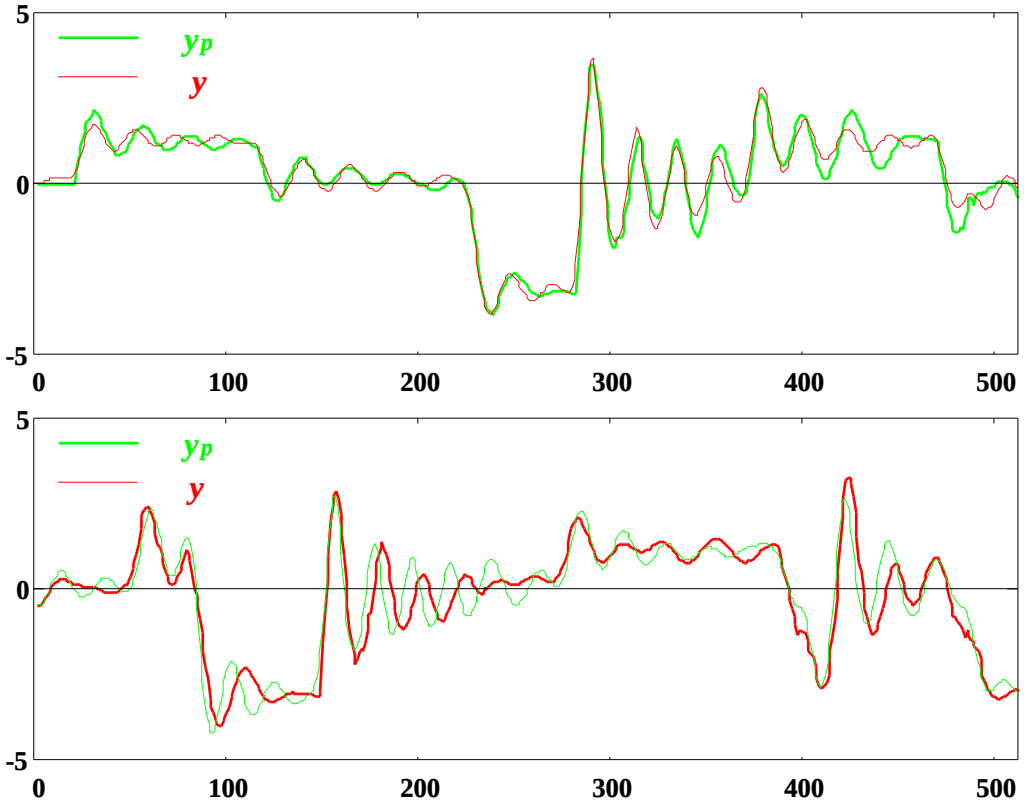

FIGURE 24
Result obtained on the training (top) and test (bottom) sequences after training the input-output model

With the state-space model shown on Figure 25, the results on the test set are as shown on Figure 26.

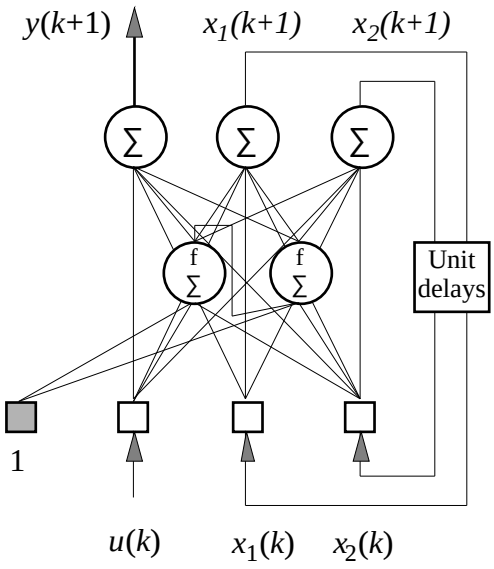

FIGURE 25
State-space model used for the modeling of the actuator of the robot arm

The mean-square error is 0.10 on the training sequence, and 0.11 on the test sequence, which is much better than the input-output model. This clearly illustrates the advantage of using state-space predictors when small training sets only are available: since they require a smaller number of inputs, they are more parsimonious, hence less prone to overfitting.

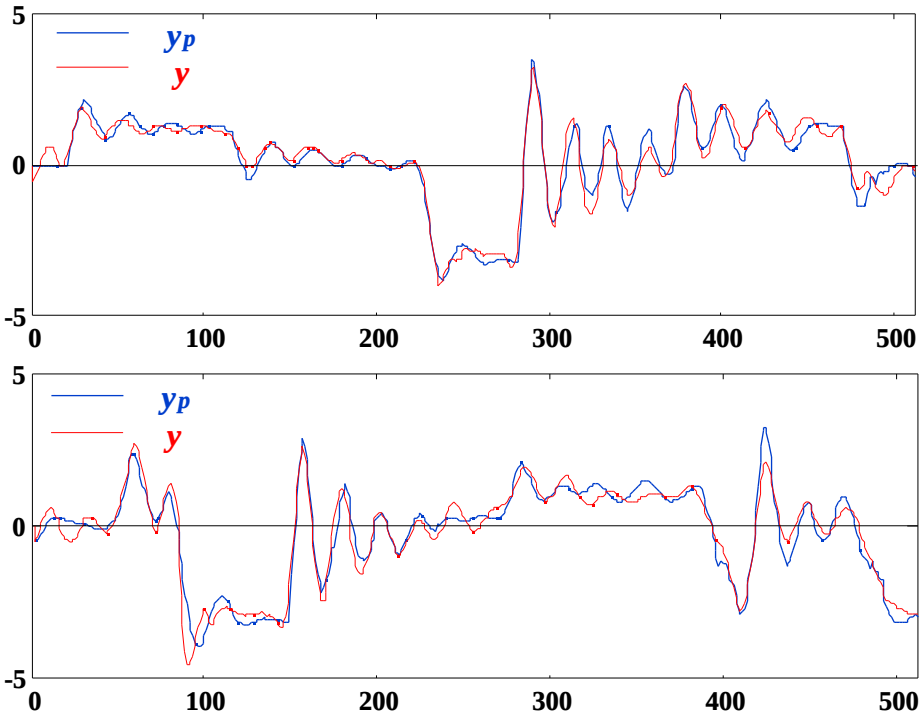

FIGURE 26
Result obtained on the training (top) and test (bottom) sequences
after training the state-space model

Context and Conclusion

The science of neural networks is an open and fascinating domain, at the crossroads of biology, physics and engineering. Now, at the end of this brief survey, devoted mainly to perspectives on the use of networks of formal neurons to engineering applications, a few words on the broader context are in order.

In their strict definition, as purely feedforward architectures, the Perceptrons are the essential building blocks for all statistical physics studies of neural nets. Within this framework, a limit of infinite number of inputs is taken (similar to the limit of infinite number of particles in thermodynamics); this apparent complication turns out to be a very fertile one, because it allows for the solution of many relevant models (memory storage, rule learning (Watkin et al. 1993), generalization, etc.). Furthermore, the exact

results derived from such perceptron structures serve as foundations for the study of more general nets, incorporating feedback connections.

Indeed the presence of reciprocal (feedback) connections is a pervasive feature of biological nervous systems. Despite this conspicuous anatomical fact, neurobiological modeling based on feedforward nets is an active field of research, and one can submit at least four good reasons for that:
i) such models provide often a workable first-order approximation,
ii) they may serve as steps towards higher-order improvements,
iii) in some physiological functions, neural processing is indeed so fast and directed (reflex arcs) that feedforward processing contains the essence,
iv) finally, there is a potential for future inspiring analogies with the computational and algorithmic properties of the formal nets, developed for engineering applications, as described and illustrated in this survey article.

References cited

BENVENISTE, A., JUDITSKY, A., DELYON, B., ZHANG, Q., GLORENNEC, P.-Y. (1994) *10th IFAC Symposium on Identification* (Copenhagen, 1994).

BRIDLE, J.S. (1990) *Neurocomputing: Algorithms, Architectures and Applications* 227-236, F. Fogelman-Soulie and J. Herault (eds.). NATO ASI Series, Springer.

COVER, T.M. (1965) *IEEE Transactions on Electronic Computers* 14, 326-334.

GUYON, I., WANG, P.S.P. (1994) *Advances in Pattern Recognition Systems Using Neural Network Technologies.* World Scientific.

HASSIBI, B., AND STORK, D.G. (1993) *Advances in Neural Information Processing Systems* 5, 164-171.

HOPFIELD, J.J.(1982) *Proc. Natl. Acad. Sci. USA* 79, 2554-2558.

HOPFIELD, J.J., AND TANK, D.V. (1985) *Proc. Natl. Acad. Sci. USA* 81, 3088-3092

HORNIK, K., STINCHCOMBE, M., WHITE, H. AND AUER, P. (1994) *Neural Computation* 6, 1262-1275.

KNERR, S., PERSONNAZ, L., DREYFUS, G. (1990) *Neurocomputing: Algorithms, Architectures and Applications* 41-50, F. Fogelman-Soulie and J. Herault (eds.). NATO ASI Series, Springer.

LEONTARITIS, I. J., AND BILLINGS, S.A. (1987) *Int. J. Control* 1, 311-341.

LEVIN, A. U., NARENDRA, K. S. (1995) *Neural Computation* 7, 349-357.

MC CULLOCH, W.S. AND PITTS, W.H. (1943) *Bull. Math. Bioph.* 3, 115-133.

MOODY, J.E., AND DARKEN, C.J. (1989) *Neural Computation* 1, 281-294.

NADAL, J.P. (1989) *International Journal of Neural Systems* 1, 55-59.

NARENDRA K. S., PARTHASARATHY K. (1990) *IEEE Trans. on Neural Networks* 1, 4-27.

NERRAND, O., ROUSSEL-RAGOT, P., PERSONNAZ, L. AND DREYFUS, G. (1993) *Neural Computation* 5, 165-199.

PLOIX, J.L., AND DREYFUS, G (1996) in *Industrial Applications of Neural Networks,* F. Fogelman-Soulie, P. Gallinari, eds., World Scientific.

POGGIO, T., AND GIROSI, F. (1990) *Science* 247, 1481-1497.

PRESS, W.H., TEUKOLSKY, S.A., VETTERLING, W.T., FLANNERY, B.P.(1992) *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press.

RIVALS, I., CANAS, D., PERSONNAZ, L., AND DREYFUS, G. (1994) *Proceedings of the IEEE Conference on Intelligent Vehicles* 137-142.

URBANI, D., ROUSSEL-RAGOT, P., PERSONNAZ, L., DREYFUS, G. (1994) *Neural Networks for Signal Processing* 4, 229-237.

WATKIN, T.L.H., RAU, M., BIEHL, M. (1993) *Rev. Mod. Phys.* 65, 499-556.

WEIGEND, A.S., AND GERSHENFELD, N.A. (1994) *Time Series Prediction: Forecasting the Future and Understanding the Past.* Addison-Wesley

General references

ARBIB, M. (1995) *The Handbook of Brain Theory and Neural Networks.* MIT Press

BISHOP, C.M.(1995) *Neural networks for Pattern Recognition.* Oxford.

DUDA, R.O., HART, P.E. (1973) *Pattern Classification and Scene Analysis*. Wiley.

GRASSBERGER, P., AND NADAL, J.P. (1994) *From Statistical Physics to Statistical Inference*. Kluwer.

GUTFREUND, H., TOULOUSE, G. (1994) *Biology and Computation: a Physicist's Choice.* World Scientific.

HERTZ, J., KROGH, A., AND PALMER, R.G. (1991) *Introduction to the Theory of Neural Computation.* Addison-Wesley.

LJUNG, L. (1987) *System Identification; Theory for the User*. Prentice Hall.

MINSKY, M., AND PAPERT, S. (1969) *Perceptrons*. MIT Press.

RUMELHART, D.E., HINTON, G.E., WILLIAMS, R.J. (1986) *Parallel Distributed Processing: Explorations into the Microstructure of Cognition*, MIT Press.

STEIN, D.(1992) *Spin Glasses and Biology*. World Scientific.