1

Neural Networks: An Overview

G. Dreyfus

How useful is that new technology? This is a natural question to ask whenever an emerging technique, such as neural networks, is transferred from research laboratories to industry. In addition, the biological flavor of the term "neural network" may lead to some confusion. For those reasons, this chapter is devoted to a presentation of the mathematical foundations and algorithms that underlie the use of neural networks, together with the description of typical applications; although the latter are quite varied, they are all based on a small number of simple principles.

Putting neural networks to work is quite simple, and good software development tools are available. However, in order to avoid disappointing results, it is important to have an in-depth understanding of what neural networks really do and of what they are really good at. The purpose of the present chapter is to explain under what circumstances neural networks are preferable to other data processing techniques and for what purposes they may be useful.

Basic definitions will be first presented: (formal) neuron, neural networks, neural network training (both supervised and unsupervised), feedforward and feedback (or recurrent) networks.

The basic property of neural networks with supervised training, parsimonious approximation, will subsequently be explained. Due to that property, neural networks are excellent nonlinear modeling tools. In that context, the concept of supervised training will emerge naturally as a nonlinear version of classical statistical modeling methods. Attention will be drawn to the necessary and sufficient conditions for an application of neural networks with supervised training to be successful.

Automatic classification (or discrimination) is an area of application of neural networks that has specific features. A general presentation of automatic classification, from a probabilistic point of view, will be made. It will be shown that not all classification problems can be solved efficiently by neural networks, and we will characterize the class of problems where neural classification is most appropriate. A general methodology for the design of neural classifiers will be explained.

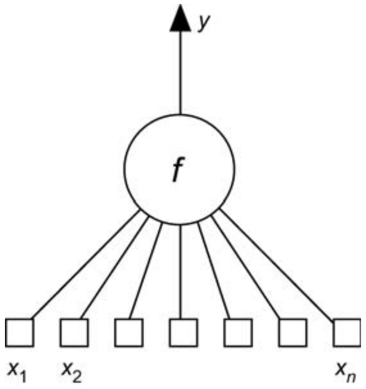


Fig. 1.1. A neuron is a nonlinear bounded function $y = f(x_1, x_2, \ldots x_n; w_1, w_2, \ldots, w_p)$ where the $\{x_i\}$ are the variables and the $\{w_j\}$ are the parameters (or weights) of the neuron

Finally, various applications will be described that illustrate the variety of areas where neural networks can provide efficient and elegant solutions to engineering problems, such that pattern recognition, nondestructive testing, information filtering, bioengineering, material formulation, modeling of industrial processes, environmental control, robotics, etc. Further applications (spectra interpretation, classification of satellite images, classification of sonar signals, process control) will be either mentioned or described in detail in subsequent chapters.

1.1 Neural Networks: Definitions and Properties

A *neuron* is a nonlinear, parameterized, bounded function.

For convenience, a linear parameterized function is often termed a linear neuron.

The variables of the neuron are often called inputs of the neuron and its value is its output. A neuron can be conveniently represented graphically as shown on Fig. 1.1. This representation stems from the biological inspiration that prompted the initial interest in formal neurons, between 1940 and 1970 [McCulloch 1943; Minsky 1969].

Function f can be parameterized in any appropriate fashion. Two types of parameterization are of current use.

- The parameters are assigned to the inputs of the neurons; the output of the neuron is a nonlinear combination of the inputs $\{x_i\}$, weighted by the parameters $\{w_i\}$, which are often termed weights, or, to be reminiscent of the biological inspiration of neural networks, synaptic weights. Following the current terminology, that linear combination will be termed potential in the present book, and, more specifically, linear potential in Chap. 5. The most frequently used potential v is a weighted sum of the inputs, with an additional constant term called "bias",

$$v = w_0 + \sum_{i=1}^{n-1} w_i x_i .$$

Function f is termed activation function. For reasons that will be explained below, it is advisable that function f be a sigmoid function (i.e., an s-shaped function), such as the tanh function or the inverse tangent function. In most applications that will be described in the present chapter, the output y of a neuron with inputs $\{x_i\}$ is given by $y = \tanh[w_0 + \sum_{i=1}^{n-1} w_i x_i]$.

- The parameters are assigned to the neuron nonlinearity, i.e., they belong to the very definition of the activation function such is the case when function f is a radial basis function (RBF) or a wavelet; the former stem from approximation theory [Powell 1987], the latter from signal processing [Mallat 1989].
  For instance, the output of a Gaussian RBF is given by

  $$y = \exp\left[-\sum_{i=1}^{n} (x_i - w_i)^2 \middle/ 2w_{n+1}^2 \right],$$

  where the parameters w_i, $i = 1$ to n, are the position of the center of the Gaussian and w_{n+1} is its standard deviation.

Additional examples of neurons are given in the theoretical and algorithmic supplements, at the end of the chapter.

For practical purposes, the main difference between the above two categories of neurons is that RBFs and wavelets are local nonlinearities, which vanish asymptotically in all directions of input space, whereas neurons that have a potential and a sigmoid nonlinearity have an infinite-range influence along the direction defined by $v = 0$.

1.1.1 Neural Networks

It has just been shown that a neuron is a nonlinear, parameterized function of its input variables. Naturally enough, a network of neurons is the composition of the nonlinear functions of two or more neurons.

Neural networks come in two classes: feedforward networks and recurrent (or feedback) networks.

1.1.1.1 Feedforward Neural Networks

General Form

A *feedforward neural network* is a nonlinear function of its inputs, which is the composition of the functions of its neurons.

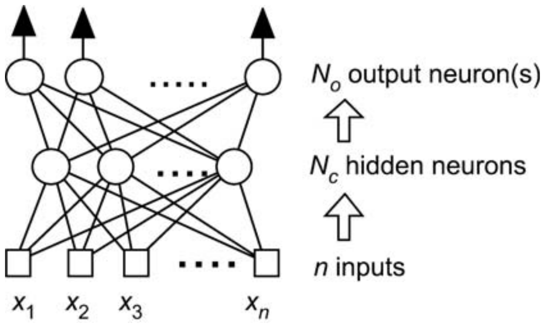


Fig. 1.2. A neural network with n inputs, a layer of N_c hidden neurons, and N_o output neurons

Therefore, a feedforward neural network is represented graphically as a set of neurons connected together, in which the information flows only in the forward direction, from inputs to outputs. In a graph representation, where the vertices are the neurons and the edges are the connections, the graph of a feedforward network is acyclic: no path in the graph, following the connections, can lead back to the starting point. The graph representation of the topology of the network is a useful tool, especially for analyzing recurrent networks, as will be shown in Chap. 2.

The neurons that perform the final computation, i.e., whose outputs are the outputs of the network, are called output neurons; the other neurons, which perform intermediate computations, are termed hidden neurons (see Fig. 1.2).

One should be wary of the term connection, which should be taken metaphorically. In the vast majority of applications, neurons are not physical objects, e.g., implemented electronically in silicon, and connections do not have any actual existence: the computations performed by each neuron are implemented as software programs, written in any convenient language and running on any computer. The term connection stems from the biological origin of neural networks; it is convenient, but it may be definitely misleading. So is the term connectionism.

Multilayer Networks

A great variety of network topologies can be imagined, under the sole constraint that the graph of connections be acyclic. However, for reasons that will be developed in a subsequent section, the vast majority of neural network applications implement multilayer networks, an example of which is shown on Fig. 1.2.

General Form

That network computes N_o functions of the input variables of the network; each output is a nonlinear function (computed by the corresponding output neuron) of the nonlinear functions computed by the hidden neurons.

A *feedforward network* with n inputs, N_c hidden neurons and N_o output neurons computes N_o nonlinear functions of its n input variables as compositions of the N_c functions computed by the hidden neurons.

It should be noted that feedforward networks are static; if the inputs are constant, then so are the outputs. The time necessary for the computation of the function of each neuron is usually negligibly small. Thus, feedforward neural networks are often termed static networks in contrast with recurrent or dynamic networks, which will be described in a specific section below.

Feedforward multilayer networks with sigmoid nonlinearities are often termed *multilayer perceptrons,* or *MLPs*.

In the literature, an input layer and input neurons are frequently mentioned as part of the structure of a multilayer perceptron. That is confusing because the inputs (shown as squares on Fig. 1.2, as opposed to neurons, which are shown as circles) are definitely not neurons: they do not perform any processing on the inputs, which they just pass as variables of the hidden neurons.

Feedforward Neural Networks with a Single Hidden Layer of Sigmoids and a Single Linear Output Neuron

The final part of this presentation of feedforward neural networks will be devoted to a class of feedforward neural networks that is particularly important in practice: networks with a single layer of hidden neurons with a sigmoid activation function, and a linear output neuron (Fig. 1.3).

The output of that network is given by

$$\begin{aligned} g(\boldsymbol{x}, \boldsymbol{w}) &= \sum_{i=1}^{N_c} \left[w_{N_c+1,i} \tanh \left(\sum_{j=1}^{n} w_{ij} x_j + w_{i0} \right) \right] + w_{N_c+1,0} \\ &= \sum_{i=1}^{N_c} \left[w_{N_c+1,i} \tanh \left(\sum_{j=0}^{n} w_{ij} x_j \right) \right] + w_{N_c+1,0}, \end{aligned}$$

where $\boldsymbol{x}$ is the input $(n+1)$-vector, and $\boldsymbol{w}$ is the vector of $(n+1)N_c+(N_c+1)$ parameters. Hidden neurons are numbered from 1 to N_c, and the output neuron is numbered $N_c + 1$. Conventionally, the parameter w_{ij} is assigned to the connection that conveys information from neuron j (or from network input j) to neuron i.

The output $g(\boldsymbol{x}, \boldsymbol{w})$ of the network is a linear function of the parameters of the last connection layer (connections that convey information from the N_c hidden neurons to the output neuron $N_c + 1$), and it is a nonlinear function

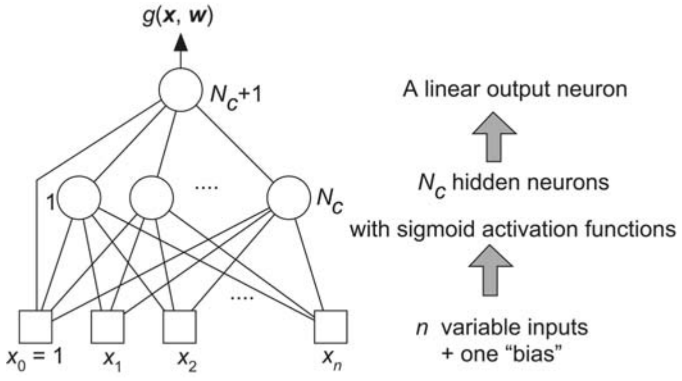


Fig. 1.3. A neural network with $n+1$ inputs, a layer of N_c hidden neurons with sigmoid activation function, and a linear output neuron. Its output $g(\boldsymbol{x}, \boldsymbol{w})$ is a nonlinear function of the input vector $\boldsymbol{x}$, whose components are $1, x_1, x_2, \ldots, x_n$, and of the vector of parameters $\boldsymbol{w}$, whose components are the $(n+1)N_c + N_c + 1$ parameters of the network

of the parameters of the first layer of connections (connections that convey information from the $n+1$ inputs of the network to the N_c hidden neurons). That property has important consequences, which will be described in detail in a subsequent section.

The output of a multilayer perceptron is a nonlinear function of its inputs and of its parameters.

1.1.1.2 What Is a Neural Network with Zero Hidden Neurons?

A feedforward neural network with zero hidden neuron and a linear output neuron is an affine function of its inputs. Hence, any linear system can be regarded as a neural network. That statement, however, does not bring anything new or useful to the well-developed theory of linear systems.

1.1.1.3 Direct Terms

If the function to be computed by the feedforward neural network is thought to have a significant linear component, it may be useful to add linear terms (sometimes called direct terms) to the above structure; they appear as additional connections on the graph representation of the network, which convey information directly from the inputs to the output neuron (Fig. 1.4). For instance, the output of a feedforward neural network with a single layer of activation functions and a linear output function becomes

$$g(\boldsymbol{x}, \boldsymbol{w}) = \sum_{i=1}^{N_c} \left[w_{N_c+1,i} \tanh\left(\sum_{j=0}^{n} w_{ij} x_j \right) \right] + \sum_{j=0}^{n} w_{N_c+1,j} x_j .$$

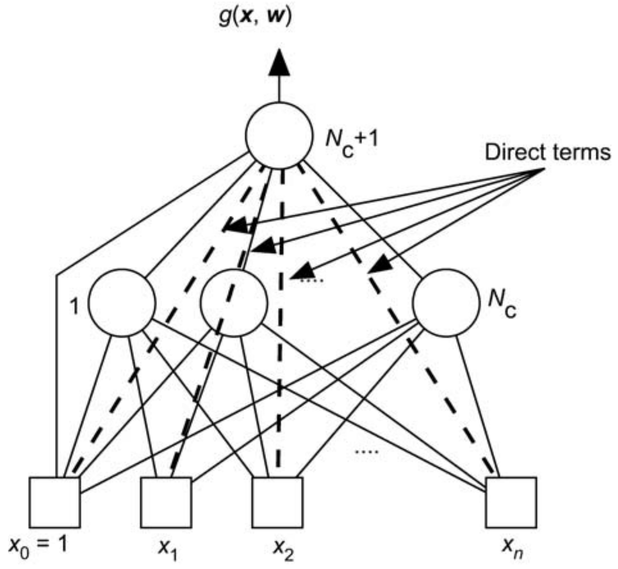


Fig. 1.4. A feedforward neural network with direct terms. Its output $g(\boldsymbol{x}, \boldsymbol{w})$ depends on the input vector $\boldsymbol{x}$, whose components are $1, x_1, x_2, \ldots, x_n$, and on the vector of parameters $\boldsymbol{w}$, whose components are the parameters of the network

RBF (Radial Basis Functions) and Wavelet Networks

The parameters of such networks are assigned to the nonlinear activation function, instead of being assigned to the connections; as in MLP's, the output is a linear combination of the outputs of the hidden RBF's. Therefore, the output of the network (for Gaussian RBF's) is given by

$$g(\boldsymbol{x}, \boldsymbol{w}) = \sum_{i=1}^{N_c} \left[w_{N_c+1,i} \exp \left(-\frac{\sum_{j=1}^{n} (x_j - w_{ij})^2}{2w_i^2} \right) \right],$$

where $\boldsymbol{x}$ is the n-vector of inputs, and $\boldsymbol{w}$ is the vector of $((n+2)N_c)$ parameters [Broomhead 1988; Moody 1989]; hidden neurons are numbered from 1 to N_c, and the output neuron is numbered $N_c + 1$.

The parameters of an RBF network fall into two classes: the parameters of the last layer, which convey information from the N_c RBF (outputs to the output linear neuron), and the parameters of the RBF's (centers and standard deviations for Gaussian RBF's). The connections of the first layer (from inputs to RBF's) are all equal to 1. In such networks, the output is a linear function of the parameters of the last layer and it is a nonlinear function of the parameters of the Gaussians. This has an important consequence that will be examined below.

Wavelet networks have exactly the same structure, except for the fact that the nonlinearities of the neurons are wavelets instead of being Gaussians. The parameters that are relevant to the nonlinearity are the translations and the dilations of the wavelets [Benveniste 1994; Oussar 2000].

1.1.1.4 Recurrent (Feedback) Neural Networks

General Form

The present section is devoted to a presentation of the most general neural network architecture: recurrent neural networks, whose connection graph exhibits *cycles*. In that graph, there exists at least one path that, following the connections, leads back to the starting vertex (neuron); such a path is called a *cycle*. Since the output of a neuron cannot be a function of itself, such an architecture requires that *time* be explicitly taken into account: the output of a neuron cannot be a function of itself *at the same instant of time*, but it can be a function *of its past value(s)*.

At present, the vast majority of neural network applications are implemented as digital systems (either standard computers, or special-purpose digital circuits for signal processing): therefore, *discrete-time systems* are the natural framework for investigating recurrent networks, which are described mathematically by recurrent equations (hence the name of those networks). Discrete-time (or recurrent) equations are discrete-time equivalents of continuous-time differential equations.

Therefore, each connection of a recurrent neural network is assigned a *delay* (possibly equal to zero), in addition to being assigned a parameter as in feedforward neural networks. Each delay is an integer multiple of an elementary time that is considered as a time unit. From causality, a quantity, at a given time, cannot be a function of itself at the same time: therefore, the sum of the delays of the edges of a cycle in the graph of connections must be nonzero.

A *discrete-time recurrent neural network* obeys a set of nonlinear discrete-time recurrent equations, through the composition of the functions of its neurons, and through the time delays associated to its connections.

Property. *For causality to hold, each cycle of the connection graph must have at least one connection with a nonzero delay.*

Figure 1.5 shows an example of a recurrent neural network. The digits in the boxes are the delays attached to the connections, expressed as integer multiples of a time unit (or sampling period) T. The network features a cycle, from neuron 3 back to neuron 3 through neuron 4; since the connection from 4 to 3 has a delay of one time unit, the network is causal.

Further Details

At time kT, the inputs of neuron 3 are $u_1(kT), u_2[(k-1)T], y_4[(k-1)T]$ (where k is a positive integer and $y_4(kT)$ is the output of neuron 4 at time kT), and

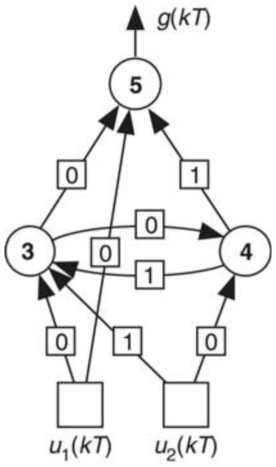


Fig. 1.5. A two-input recurrent neural network. Digits in *square boxes* are the delay assigned to each connection, an integer multiple of the time unit (or sampling period) T. The network features a cycle from 3 to 3 through 4

it computes its output $y_3(kT)$; the inputs of neuron 4 are $u_2(kT)$ and $y_3(kT)$, and it computes its output $y_4(kT)$; the inputs of neuron 5 are $y_3(kT), u_1(kT)$ et $y_4[(k-1)T]$, and it computes its output, which is the output of the network $g(kT)$.

The Canonical Form of Recurrent Neural Networks

Because recurrent neural networks are governed by recurrent discrete-time equations, it is natural to investigate the relations between such nonlinear models and the conventional dynamic linear models, as used in linear modeling and control.

The general mathematical description of a linear system is the state equations,

$$\begin{aligned} \boldsymbol{x}(k) &= A\,\boldsymbol{x}(k-1) + B\,\boldsymbol{u}(k-1) \\ \boldsymbol{g}(k) &= C\,\boldsymbol{x}(k-1) + D\,\boldsymbol{u}(k-1), \end{aligned}$$

where $\boldsymbol{x}(k)$ is the state vector at time $kT, \boldsymbol{u}(k)$ is the input vector at time $kT, \boldsymbol{g}(k)$ is the output vector at time kT and A, B, C, D are matrices. The state variables are the minimal set of variables such that their values at time $(k+1)T$ can be computed if (i) their initial values are known, and if (ii) the values of the inputs are known at all time from 0 to kT. The number of state variables is the *order* of the system.

Similarly the canonical form of a nonlinear system is defined as

$$\begin{aligned} \boldsymbol{x}(k) &= \boldsymbol{\Phi}[\boldsymbol{x}(k-1), \boldsymbol{u}(k-1)] \\ \boldsymbol{g}(k) &= \boldsymbol{\Psi}[\boldsymbol{x}(k-1), \boldsymbol{u}(k-1)], \end{aligned}$$

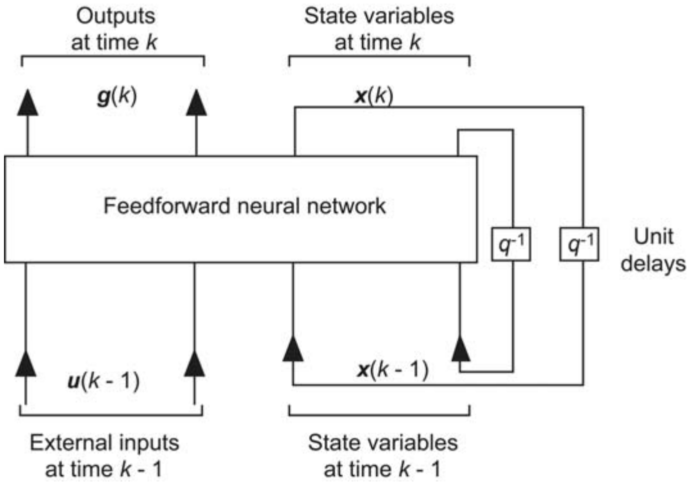


Fig. 1.6. The canonical form of a recurrent neural network. The symbol q^{-1} stands for a unit time delay

where $\boldsymbol{\Phi}$ and $\boldsymbol{\Psi}$ are nonlinear vector functions, e.g., neural networks, and where $\boldsymbol{x}$ is the state vector. As in the linear case, the state variables are the elements of the minimal set of variables such that the model can be described completely at time $k+1$ given the initial values of the state variables, and the inputs from time 0 to time k. It will be shown in Chap. 2 that any recurrent neural network can be cast into a canonical form, as shown on Fig. 1.6, where q^{-1} stands for a unit time delay. This symbol, which is usual in control theory, will be used in throughout this book, especially in Chaps. 2 and 4.

Property. *Any recurrent neural network, however complex, can be cast into a canonical form, made of a feedforward neural network, some outputs of which (termed state outputs) are fed back to the inputs through unit delays [Nerrand 1993].*

For instance, the neural network of Fig. 1.5 can be cast into the canonical form that is shown on Fig. 1.7. That network has a single state variable (hence it is a first-order network): the output of neuron 3. In that example, neuron 3 is a hidden neuron, but it will be shown below that a state neuron can also be an output neuron.

Further Details

At time kT, the inputs of neuron 4 are $u_2[(k-1)T]$ and $x[(k-1)T] = y_3[(k-1)T]$); therefore, its output is $y_4[(k-1)T]$; just as in the original (non-canonical) form, the inputs of neuron 3 are $u_1(kT), u_2[(k-1)T]$, $y_4[(k-1)T]$; therefore, its output is $y_3(kT)$; the inputs of neuron 5 are

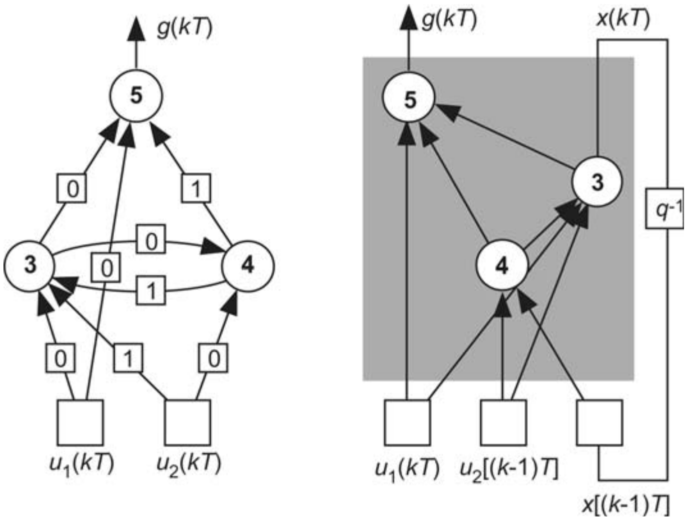


Fig. 1.7. The canonical form (*right-hand side*) of the network shown on Fig. 1.5 (*left-hand side*). That network has a single state variable $x(kT)$ (output of neuron 3): it is a first-order network. The gray part of the canonical form is a feedforward neural network

$y_3(kT), u_1(kT), y_4[(k-1)T]$; therefore, its output is $g(kT)$, which is the output of the network. Hence, both networks are functionally equivalent.

Recurrent neural networks (and their canonical form) will be investigated in detail in Chaps. 2, 4 and 8.

1.1.1.5 Summary

In the present section, we stated the basic definitions that are relevant to the neural networks investigated in the present book. We made specific distinctions between:

- Feedforward (or static) neural networks, which implement nonlinear functions of their inputs,
- Recurrent (or dynamic) neural networks, which are governed by nonlinear discrete-time recurrent equations.

In addition, we showed that any recurrent neural network can be cast into a canonical form, which is made of a feedforward neural network whose outputs are fed back to its inputs with a unit time delay.

Thus, the basic element of any neural network is a feedforward neural network. Therefore, we will first study in detail feedforward neural networks. Before investigating their properties and applications, we will consider the concept of training.

1.1.2 The Training of Neural Networks

Training is the algorithmic procedure whereby the parameters of the neurons of the network are estimated, in order for the neural network to fulfill, as accurately as possible, the task it has been assigned.

Within that framework, two categories of training are considered: supervised training and unsupervised training.

1.1.2.1 Supervised Training

As indicated in the previous section, a feedforward neural network computes a nonlinear function of its inputs. Therefore, such a network can be assigned the task of computing a specific nonlinear function. Two situations may arise:

- The nonlinear function is known analytically: hence the network performs the task of function approximation,
- The nonlinear function is not known analytically, but a finite number of numerical values of the function are known; in most applications, these values are not known exactly because they are obtained through measurements performed on a physical, chemical, financial, economic, biological, etc. process: in such a case, the task that is assigned to the network is that of approximating the regression function of the available data, hence of being a static model of the process.

In the vast majority of their applications, feedforward neural networks with supervised training are used in the second class of situations.

Training can be thought of as "supervised" since the function that the network should implement is known in some or all points: a "teacher" provides "examples" of values of the inputs and of the corresponding values of the output, i.e., of the task that the network should perform. The core of Chap. 2 of the book is devoted to translating the above metaphor into mathematics and algorithms. Chapters 3, 4, 5 and 6 are devoted to the design and applications of neural networks with supervised training for static and dynamic modeling, and for automatic classification (or discrimination).

1.1.2.2 Unsupervised Training

A feedforward neural network can also be assigned a task of data analysis or visualization: a set of data, described by a vector with a large number of components, is available. It may be desired to cluster these data, according to similarity criteria that are not known a priori. Clustering methods are well known in statistics; feedforward neural networks can be assigned a task that is close to clustering: from the high-dimensional data representation, find a representation of much smaller dimension (usually 2-dimensional) that preserves the similarities or neighborhoods. Thus, no teacher is present in that task, since the training of the network should “discover” the similarities between elements of the database, and translate them into vicinities in the new data representation or “map.” The most popular feedforward neural networks with unsupervised training are the “self-organizing maps” or “Kohonen maps”. Chapter 7 is devoted to self-organizing maps and their applications.

1.1.3 The Fundamental Property of Neural Networks with Supervised Training: Parsimonious Approximation

1.1.3.1 Nonlinear in Their Parameters, Neural Networks Are Universal Approximators

Property. *Any bounded, sufficiently regular function can be approximated uniformly with arbitrary accuracy in a finite region of variable space, by a neural network with a single layer of hidden neurons having the same activation function, and a linear output neuron [Hornik 1989, 1990, 1991].*

That property is just a proof of existence and does not provide any method for finding the number of neurons or the values of the parameters; furthermore, it is not specific to neural networks. The following property is indeed specific to neural networks, and it provides a rationale for the applications of neural networks.

1.1.3.2 Some Neural Networks Are Parsimonious

In order to implement real applications, the number of functions that are required to perform an approximation is an important criterion when a choice must be made between different models. It will be shown in the next section that the model designer ought always to choose the model with the smallest number of parameters, i.e., the most *parsimonious* model.

Fundamental Property

It can be shown [Barron 1993] that, if the model is nonlinear with respect to its parameters, it is more parsimonious than if the model is linear with respect to its parameters.

More specifically, it can be shown that the number of parameters necessary to perform an approximation with a given accuracy varies exponentially with the number of variables for models that are linear with respect to their parameters, whereas it increases linearly with the number of variables if the model is not linear with respect to its parameters.

Therefore, that property is valuable for models that have a “large” number of inputs: for a process with one or two variables only, all nonlinear models are roughly equivalent from the viewpoint of parsimony: a model that is nonlinear with respect to its parameters is equivalent, in that respect, to a model that is linear with respect to its parameters.

In the section devoted to the definitions, we showed that the output of a feedforward neural network with a layer of sigmoid activation functions (multilayer Perceptron) is nonlinear with respect to the parameters of the network, whereas the output of a network of radial basis functions with fixed centers and widths, or of wavelets with fixed translations and dilations, is linear with respect to the parameters. Similarly, a polynomial is linear with respect to the coefficients of the monomials. Thus, neurons with sigmoid activation functions provide more parsimonious approximations than polynomials or radial basis functions with fixed centers and widths, or wavelets with fixed translations and dilations. Conversely, if the centers and widths of Gaussian radial basis functions, or the centers and dilations of wavelets, are considered as adjustable parameters, there is no *mathematically proved* advantage to any one of those models over the others. However, some *practical* considerations may lead to favor one of the models over the others: prior knowledge on the type of nonlinearity that is required, local *vs.* nonlocal function, ease and speed of training (see Chap. 2, section "Parameter initialization"), ease of hardware integration into silicon, etc.

The origin of parsimony can be understood qualitatively as follows. Consider a model that is linear with respect to its parameters, such as a polynomial model, e.g.,

$$g(x) = 4 + 2x + 4x^2 - 0.5x^3.$$

The output $g(x)$ of the model is a linear combination of functions $y = 1, y = x, y = x^2, y = x^3$, with parameters (weights) $w_0 = 4, w_1 = 2, w_2 = 4, w_3 = -0.5$. The shapes of those functions are fixed.

Consider a neural model as shown on Fig. 1.8, for which the equation is

$$g(x) = 0.5 - 2\tanh(10x + 5) + 3\tanh(x + 0.25) - 2\tanh(3x - 0.25).$$

This model is also a linear combination of functions ($y = 1, y = \tanh(10x + 5), y = \tanh(x + 0.25), y = \tanh(3x - 0.25)$), but the shapes of these functions depend on the values of the parameters of the connections between the inputs and the hidden neurons. Thus, instead of combining functions whose shapes are fixed, one combines functions whose shapes are adjustable through the parameters of some connections. That provides extra degrees of freedom, which can be taken advantage of for using a smaller number of functions, hence a smaller number of parameters. That is the very essence of parsimony.

1.1.3.3 An Elementary Example

Consider the function

$$y = 16,71\ x^2 - 0,075.$$

We sample 20 equally spaced points that are used for training a multilayer Perceptron with two hidden neurons whose nonlinearity is $\tan^{-1}$, as shown on Fig. 1.9(a). Training is performed with the Levenberg-Marquardt algorithm

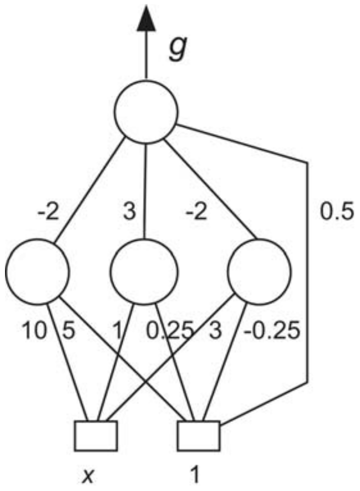


Fig. 1.8. A feedforward neural network with one variable (hence two inputs) and three hidden neurons. The numbers are the values of the parameters

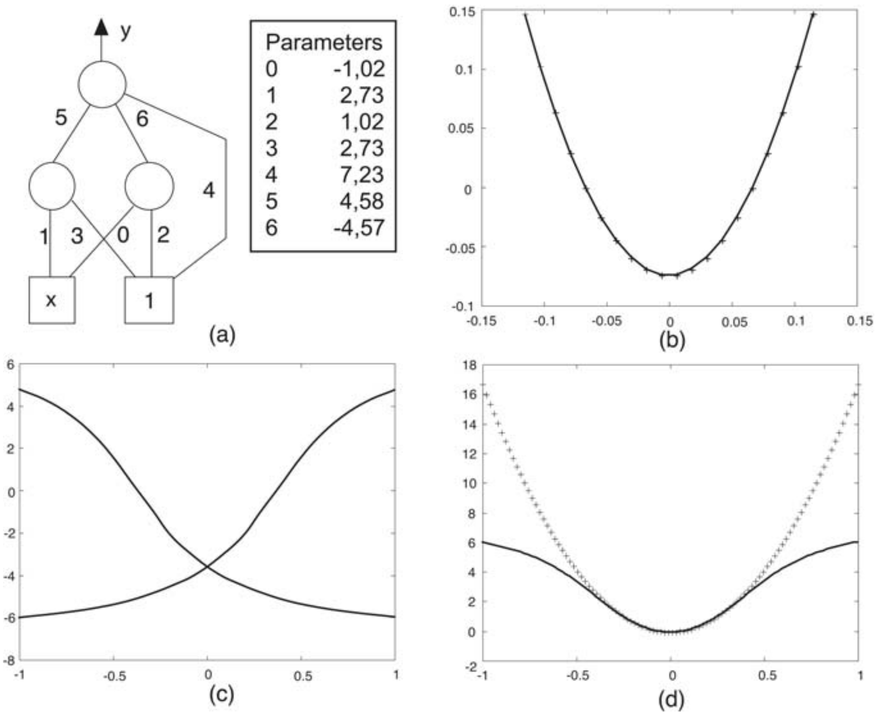
(see Chap. 2), resulting in the parameters shown on Fig. 1.9(a). Figure 1.9(b) shows the points of the training set and the output of the network, which fits the training points with excellent accuracy. Figure 1.9(c) shows the outputs of the hidden neurons, whose linear combination with the bias provides the output of the network. Figure 1.9(d) shows the points of a test set, i.e., a set of points that were not used for training: outside of the domain of variation of the variable x within which training was performed ($[-0.12, +0.12]$), the approximation performed by the network becomes extremely inaccurate, as expected. The striking symmetry in the values of the parameters shows that training has successfully captured the symmetry of the problem (simulation performed with the NeuroOne$^{\text{TM}}$ software suite by NETRAL S.A.).

It should be clear that using a neural network to approximate a single-variable parabola is overkill, since the parabola has two parameters whereas the neural network has seven parameters! This example has a didactic character insofar as simple one-dimensional graphical representations can be drawn.

1.1.4 Feedforward Neural Networks with Supervised Training for Static Modeling and Discrimination (Classification)

The mathematical properties described in the previous section are the basis of the applications of feedforward neural networks with supervised training. However, for all practical purposes, neural networks are scarcely ever used for uniformly approximating a *known* function.

In most cases, the engineer is faced with the following problem: a set of measured variables $\{\boldsymbol{x}^k,\ k = 1 \text{ to } N\}$, and a set of measurements $\{y_p(\boldsymbol{x}^k),$



Parameters	
0	-1,02
1	2,73
2	1,02
3	2,73
4	7,23
5	4,58
6	-4,57

Fig. 1.9. Interpolation of a parabola by a neural network with two hidden neurons; (**a**) network; (**b**) training set (+) and network output (*line*) after training; (**c**) outputs of the two hidden neurons (sigmoid functions) after training; (**d**) test set (+) and network output (*line*) after training: as expected, the approximation is very inaccurate outside the domain of variation of the inputs during training

$k = 1$ to $N\}$ of a quantity of interest z_p related to a physical, chemical, financial, ..., process, are available. He assumes that there exists a relation between the vector of variables $\{\boldsymbol{x}\}$ and the quantity z_p, and he looks for a mathematical form of that relation, which is valid in the region of variable space where the measurements were performed, given that (1) the number of available measurements is finite, and (2) the measurements are corrupted by noise. Moreover, the variables that actually affect z_p are not necessarily measured. In other words, the engineer tries to build a *model* of the process of interest, from the available measurements only: such a model is called a *black-box* model. In neural network parlance, the observations from which the model is designed are called *examples*. We will consider below the "black-box" modeling of the hydraulic actuator of a robot arm: the set of variables $\{\boldsymbol{x}\}$ has a single element (the angle of the oil valve), and the quantity of interest $\{z_p\}$ is the oil pressure in the actuator. We will also describe an example of prediction of chemical properties of molecules: a relation between a molecular property (e.g., the boiling point) and "descriptors" of the molecules (e.g., the molecular mass, the number of atoms, the dipole moment, etc.); such a model allows predictions of the boiling points of molecules that were not synthesized before. Several similar cases will be described in this book.

Black-box models, as defined above, are in sharp contrast with knowledge-based models, which are made of mathematical equations derived from first principles of physics, chemistry, economics, etc. A knowledge-based model may have a limited number of adjustable parameters, which, in general, have a physical meaning. We will show below that neural networks can be building blocks of gray box or semi-physical models, which take into account both expert knowledge—as in a knowledge-based model—and data—as in a black-box model.

Since neural networks are not really used for function approximation, to what extent is the above-mentioned parsimonious approximation property relevant to neural network applications? In the present chapter, a cursory answer to that question will be provided. A very detailed answer will be provided in Chap. 2, in which a general design methodology will be presented, and in Chap. 3, which provides very useful techniques for the reduction of input dimension, and for the design, and the performance evaluation, of neural networks.

1.1.4.1 Static Modeling

For simplicity, we first consider a model with a single variable x. Assume that an infinite number of measurements of the quantity of interest can be performed for a given value x_0 of the variable x. Their mean value is the quantity of interest z_p, which is called the "expectation" of y_p for the value x_0 of the variable. The expectation value of y_p is a function of x, termed "regression function". Since we know from the previous section that any function can be approximated with arbitrary accuracy by a neural network, it may be expected that the black-box modeling problem, as stated above, can be solved by estimating the parameters of a neural network that approximates the (unknown) regression function.

The approximation will not be uniform, as defined and illustrated in the previous section. For reasons that will be explained in Chap. 2, the model will perform an approximation in the least squares sense: a parameterized function (e.g., a neural network) will be sought, for which the least squares cost function

$$J(\boldsymbol{w}) = \frac{1}{2}\sum_{k=1}^{N}\left[y_p(\boldsymbol{x}^k) - g(\boldsymbol{x}^k, \boldsymbol{w})\right]^2$$

is minimal. In the above relation, $\{\boldsymbol{x}^k, k = 1 \text{ to } N\}$ is a set of measured values of the input variables, and $\{Ly_p(\boldsymbol{x}^k), k = 1 \text{ to } N\}$ as set of corresponding measured values of the quantity to be modeled. Therefore, for a network that has a given architecture (i.e., a given number of inputs and of hidden neurons),

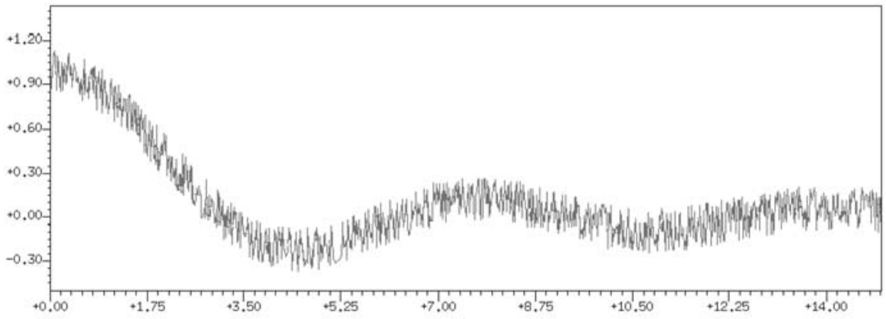


Fig. 1.10. A quantity to be modeled

training is a procedure whereby the least squares cost function is minimized, so as to find an appropriate weight vector $\boldsymbol{w_0}$.

That procedure suggests two questions, which are central in any neural network application, i.e.,

- for a given architecture, how can one find the neural network for which the least squares cost function is minimal?
- if such a neural network has been found, how can its prediction ability be assessed?

Chapter 2 of the present book will provide the reader with a methodology, based on first principles, which will answer the above questions.

These questions are not specific to neural networks: they are standard questions in the field of modeling, that have been asked for many years by all scientists (engineers, economists, biologists, and statisticians) who endeavor to extract relevant information from data [Seber 1989; Antoniadis 1992; Draper 1998]. Actually, the path from function approximation to parameter estimation of a regression function is the traditional path of any statistician in search of a model: therefore, we will take advantage of theoretical advances of statistics, especially in regression.

We will now summarize the steps that were just described.

- When a mathematical model of dependencies between variables is sought, one tries to find the regression function of the variable of interest, i.e., the function that would be obtained by averaging, at each point of variable space, the results of an infinite number of measurements; the regression function is forever unknown. Figure 1.10 shows a quantity $y_p(x)$ that one tries to model: the best approximation of the (unknown) regression function is sought.
- A finite number of measurements are performed, as shown on Fig. 1.11.
- A neural network provides an approximation of the regression function if its parameters are estimated in such a way that the sum of the squared

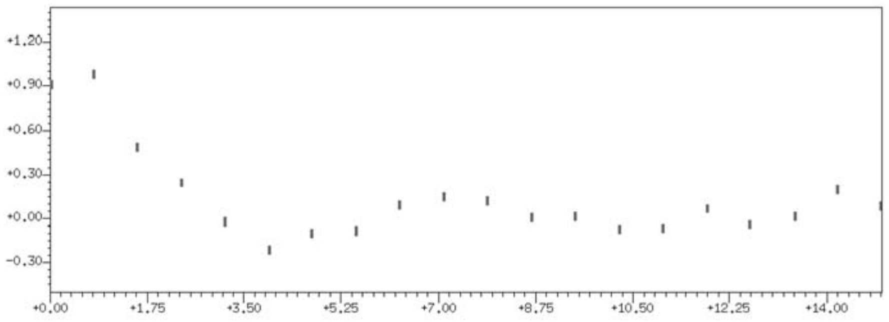


Fig. 1.11. A real-life situation: a finite number of measurements are available. Note that the measurements are equally spaced in the present example, but that is by no means necessary

differences between the values predicted by the network and the measured values is minimum, as shown on Fig. 1.12.

A neural network can thus predict, from examples, the values of a quantity that depends on several variables, for values of the variables that are not present in the database used for estimating the parameters of the model. In the case shown on Fig. 1.12, the neural network can predict values of the quantity of interest for points that lie between the measured points. That ability is termed "statistical inference" in the statistics literature, and is called "generalization" in the neural network literature. It should be absolutely clear that the generalization ability is necessarily limited: it cannot extend beyond the boundaries of the region of input space where training examples are present, as shown on Fig. 1.9. The estimation of the generalization ability is an important question that will be examined in detail in the present book.

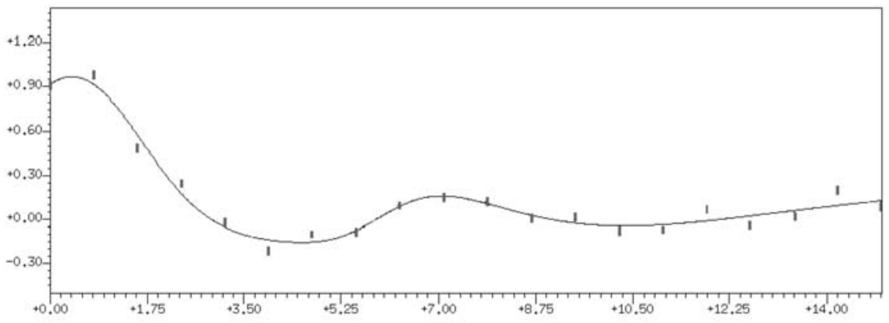


Fig. 1.12. An approximation of the regression function, performed by a neural network, from the experimental points of Fig. 1.11

1.1.4.2 To What Extent Is Parsimony a Valuable Property?

In the context of nonlinear regression and generalization, parsimony is indeed an important asset of neural networks and, more generally, of any model that is nonlinear with respect to its parameters. We mentioned earlier that most applications of neural networks with supervised learning are modeling applications, whereby the parameters of the model are adjusted, *from examples*, so as to fit the nonlinear relationship between the factors (inputs of the model) and the quantity of interest (the output of the model). It is intuitive that the *number of examples requested to estimate the parameters in a significant and robust way is larger than the number of parameters*: the equation of a straight line cannot be fitted from a single point, nor can the equation of a plane be fitted from two points. Therefore, models such as neural networks, which are parsimonious in terms of number of parameters, are also, to some extent, parsimonious in terms of number of examples; that is valuable since measurements can be costly (e.g., measurements performed on an industrial process) or time consuming (e.g., models of economy trained from indicators published monthly), or both.

Therefore, the actual advantage of neural networks over conventional nonlinear modeling techniques is their ability of providing models of equivalent accuracy from a smaller number of examples or, equivalently, of providing more accurate models from the same number of examples. In general, neural networks make the best use of the available data for models with more than 2 inputs.

Figure 1.42 illustrates the parsimony of neural networks in an industrial application: the prediction of a thermodynamic parameter of a glass.

1.1.4.3 Classification (Discrimination)

Classification (or discrimination) is the task whereby items are assigned to a class (or category) among several predefined classes. An algorithm that automatically performs a classification is called a *classifier.*

In the vocabulary of statistics, classification is the task whereby data that exhibit some similarity are grouped into classes that are not predefined; we have mentioned above that neural networks with unsupervised learning can perform such a task. Therefore, the terminology tends to be confusing. In the present book, we will try to make the distinction clear whenever the context may allow confusion. In the present section, we consider only the case of predefined classes.

Classifiers have a very large number of applications for pattern recognition (handwritten digits or characters, image recognition, speech recognition, time sequence recognition, etc.), and in many other areas as well (economy, finance, sociology, language processing, etc.). In general, a pattern may be any item that is described by a set of numerical *descriptors*: an image can be described by the set of the intensities of its pixels, a time sequence by the sequence of its values during a given time interval, a text by the frequency of occurrence of the significant words that it contains, etc. Typically, the questions whose answer a classifier is expected to contribute to are: is this unknown character a a, a b, a c, etc.? is this observed signal normal or anomalous? is this company a safe investment? is this text relevant to a given topic of interest? will there be a pollution alert to-morrow?

The classifier is not necessarily expected to give a full answer to such a question: it may make a contribution to the answer. Actually, it is often the case that the classifier is expected to be a decision aid only, the decision being made by the expert himself. In the first applications of neural networks to classification, the latter were expected to give a definite answer to the classification problem. Since significant advances have been made in the understanding of neural network operation, we know that they are able to provide a much richer information than just a binary decision as to the class of the pattern of interest: neural networks can provide an estimation of the probability of a pattern to belong to a class (also termed *posterior probability* of the class). This is extremely valuable in complex pattern recognition applications that implement several classifiers, each of which providing an estimate of the posterior probability of the class. The final decision is made by a "supervisor" system that assigns the class to the pattern in view of the probability estimates provided by the individual classifiers (committee machines).

Similarly, information filtering is an important problem in the area of data mining: find, in a large text data base, the texts that are relevant to a prescribed topic, and rank these texts by order of decreasing relevance, so that the user of the system can make a choice efficiently among the suggested documents. Again, the classifier does not provide a binary answer, but it estimates the posterior probability of the class "relevant." Feedforward neural networks are more and more frequently used for data mining applications.

Chapter 6 of the present book is fully devoted to feedforward neural networks and support vector machines for discrimination.

1.1.5 Feedforward Neural Networks with Unsupervised Training for Data Analysis and Visualization

Due to the development of powerful data processing and storage systems, very large amounts of information are available, whether in the form of numbers (intensive data processing of experimental results) or in the form of symbols (text corpuses). Therefore, the ability of retrieving information that is known to be present in the data, but that is difficult to extract, becomes crucial. Computer graphics facilitates greatly user-friendly presentation of the data, but the human operator is unable to visualize high-dimensionality data in an efficient way. Therefore, it is often desired to project high-dimensionality data onto a low-dimensionality space (typically dimension 2) in which proximity relations are preserved. Neural networks with unsupervised learning, especially self-organizing maps ("Kohonen maps"), are powerful data visualization techniques.

Chapter 7 of the present book is devoted to unsupervised learning, with emphasis on spectacular applications in satellite observation systems.

1.1.6 Recurrent Neural Networks for Black-Box Modeling, Gray-Box Modeling, and Control

In an earlier section, devoted to recurrent neural networks, we showed that any neural network can be cast into a canonical form, which is made of a feedforward neural network with external recurrent connections. Therefore, the properties of recurrent neural networks with supervised learning are strongly related to those of feedforward neural networks. The latter are used for static modeling from examples; similarly, recurrent neural networks are used for dynamic modeling from examples, i.e., for finding, from measured sequences of inputs and outputs, recurrent (discrete-time) equations that govern a process. A sizeable part of Chap. 2, and Chap. 4, are devoted to dynamic process modeling.

The design of a dynamic model may have several motivations.

- Use the model as a simulator in order to predict the evolution of a process that is described by a model whose equations are inaccurate.
- Use the model as a simulator of a process whose knowledge-based model is known and reliable, but cannot be solved accurately in real time because it contains many coupled differential or partial differential equations that cannot be solved numerically in real time with the desired accuracy: in such circumstances, one can generate a training set from the software code that solves the equations, and design a recurrent neural network that provides accurate solutions within a much shorter computation time; furthermore, it may be advantageous to take advantage of the differential equations of the knowledge-based model, as guidelines to the design of the architecture of the neural model: this is known as "gray-box" or "semi-physical" modeling, described in Sect. 1.1.6.1.
- Use the model as a one-step-ahead predictor, integrated into a control system.

1.1.6.1 Semiphysical Modeling

In the manufacturing industry, a knowledge-based model of a process of interest is often available, but is not fully satisfactory, and it cannot be improved through further analysis; this may be due to a variety of reasons:

- the model may be too inaccurate for the purpose that it should serve: for instance, if it is desired to perform fault detection by analyzing the difference between the state of the process that is predicted by the model

of normal operation, and the actual state of the process, the model of normal operation must be accurate and run in real time.
- The model may be accurate, but too complex for real-time operation (e.g., for an application in monitoring and control).

If measurements are available, in addition to the equations of the—unsatisfactory—knowledge-based model, it would be unadvisable to forsake altogether the accumulated knowledge on the process and to design a purely black-box model. Semi-physical modeling allows the model designer to have the best of both worlds: the designer can make use of the physical knowledge in order to choose the structure of the recurrent network, and make use of the data in order to estimate the parameters of the model. An industrial application of semi-physical modeling is described below, and the design methodology of a semi-physical model is explained in Chap. 2.

1.1.6.2 Process Control

The purpose of a control system is to convey a prescribed dynamics to the response of a process to a control signal or to a disturbance. In the case of a regulator system the process must stay in a prescribed state in spite of disturbances: the cruise control system of a car must keep the speed constant (equal to the setpoint speed) irrespective of the slope of the road, wind gusts, load variations, etc. A tracking system is designed to follow the variations of the setpoint, irrespective of disturbances: in a fermenting plant, the heating system must be controlled in order for the temperature to follow a prescribed temperature profile, irrespective of the temperature of ingredients that may be added during operation, of heat-producing chemical reactions that may take place, etc. In order to achieve such goals, a model of the process must be available; if necessary, the model must be nonlinear, hence be implemented as a recurrent neural network. Chapter 5 is devoted to nonlinear neural control.

1.1.7 Recurrent Neural Networks Without Training for Combinatorial Optimization

In the previous two sections, we emphasized the applications of recurrent neural networks that take advantage of their *forced dynamics*: the model designer is interested in the response of the system to control signals. By contrast, there is a special class of applications of recurrent neural networks that takes advantage of their spontaneous dynamics, i.e., of their dynamics with zero input.

Recurrent neural networks whose activation function is a step function (McCulloch-Pitts neurons), have a dynamics that features fixed points: if such a network is forced into an initial state, and is subsequently left to evolve under its spontaneous dynamics, it reaches a stable state after a finite transient sequence of states. This stable state depends on the initial state. The final state, i.e., the vector whose components are the (binary) states of the neurons of the network, can be considered as the binary code of a piece of information. Moreover, it can be shown that there exists a function, called the Liapunov function (or energy function), which always decreases during the spontaneous evolution of the state of the network; hence the stable states are the minima of the Liapunov function.

Now consider the inverse problem: in a combinatorial optimization problem, it is desired to find the minimum (or at least a good minimum) of a function (cost function) of binary variables. If there exists a recurrent neural network whose Liapunov function is identical to the cost function of the optimization problem, then the fixed points of the spontaneous dynamics of the recurrent neural network are solutions of the combinatorial optimization problem. If such a network can be constructed, then it will find a solution of the problem by evolving, under its spontaneous dynamics, from an arbitrary initial state.

Therefore, the resolution of a combinatorial optimization problem with a recurrent neural network requires

- finding a recurrent neural network whose energy function is identical to the cost function of the optimization problem,
- finding the parameters of that network,
- controlling the dynamics of the network so as to make sure that it will evolve to reach a good minimum of the cost function, for instance, by taking advantage of stochastic methods such as simulated annealing.

This powerful technique, together with some of its applications, will be described in Chap. 8 of the present book.

1.2 When and How to Use Neural Networks with Supervised Training

In the previous sections, we presented the theoretical arguments that support the use of neural networks in modeling applications. In the present section, we attack the practical questions raised by the design and training of a neural model. First, we will explain when neural networks can advantageously be used—and when they should not be used. In the subsequent section, we will emphasize *how* to use neural networks. An in-depth treatment of these important questions will be given in the next chapters.

1.2.1 When to Use Neural Networks?

We have shown earlier that the fundamental property of neural networks with supervised training is the parsimonious approximation property, i.e., their ability of approximating any sufficiently regular function with arbitrary accuracy.

Therefore, neural networks may be advantageous in any application that requires finding, in a machine learning framework, a nonlinear relation between numerical data.

Under what conditions is such an approach recommended?

- The first condition is necessary but not sufficient: since neural network design is essentially a problem in statistics, a set of examples, that sample the space of inputs appropriately, and that are in appropriate number, must be available.
- After gathering the data, one should make sure that a *nonlinear model* is necessary, since the design of a linear model is much simpler and faster than the design of a neural model. Therefore, if no prior knowledge on the quantity to be modeled is available, one should first try out a linear model; if it turns out that a linear model is too inaccurate, despite the fact that all relevant factors are present in the inputs, then the model designer may rightly resort to nonlinear models such as neural networks.
- If the appropriate examples are available, and if a nonlinear model is necessary, then one should decide whether the use of neural networks, instead of polynomials for instance, is advisable. Parsimony is the relevant choice criterion here: as mentioned above, the number of parameters of the first connection layer (between inputs and hidden neurons) increases linearly with the number of variables, whereas it increases exponentially for polynomial approximation (there exist, however, statistical tests that may, to some extent, limit the combinatorial explosion of parameters in polynomial modeling). Therefore, neural networks are advantageous when the number of variables is large, i.e., empirically, larger than or equal to 3.

To summarize, if appropriate data sets are available, neural networks can be used with advantage in all applications that require the estimation of the parameters of a regression function with three variables or more. If the number of variables is smaller, nonlinear models that are linear with respect to their parameters, such as polynomials, radial basis functions with fixed centers and standard deviation, wavelets with fixed translations and dilations, may be as accurate, and require a simpler implementation.

If the available data are not numerical (e.g., symbolic), they cannot be processed directly by a neural network. Some appropriate preprocessing is required in order to make data numerical (techniques evolved from the theory of fuzzy sets may be appropriate).

1.2.2 How to Design Neural Networks?

Neural networks are nonlinear parameterized functions, which can approximate any nonlinear function. Therefore, approaching a regression function from examples requires finding a neural network for which the sum, over all examples used for training, of the squared modeling errors (the least squares cost function) is minimum. As a consequence, the design of a neural network requires

- finding the relevant inputs, i.e., the factors that have a significant influence on the quantity to be modeled (i.e., an influence that is larger than the measurement noise),
- collecting the data that is necessary for training and testing the neural network,
- finding the appropriate complexity of the model, i.e., the appropriate number of hidden neurons,
- estimating the parameters for which the cost function is minimum, i.e., training the network,
- assessing the generalization ability of the neural network after training.

In view of the results, it may be necessary to iterate the whole procedure, or part of it.

These points will be considered in detail in the next sections.

1.2.2.1 Relevant Inputs

The selection of relevant inputs may have various requirements, depending on the application that is considered.

If the process to be modeled is an industrial problem that has been carefully engineered, the relevant factors and the causal relations between them are usually known. Consider, as an example, the industrial process of spot welding, which will be described in detail in a subsequent section: the metal sheets to be welded are melted together locally by passing a very large current (a few kiloamperes) during a few milliseconds, through two electrodes that are pressed onto the metal surfaces (Fig. 1.13). The quality of the joint is assessed from the diameter of the melted zone; it depends on the current intensity, on the duration of the current flow, on the stress applied to the electrodes while current flows and during cooling, on the surface state of the electrodes, on the nature of the metal sheets, and on a few additional factors. Thus, the desirable model inputs are essentially known from physics: however, it may be important to make a choice between these factors, so that only those factors that have a significant influence on the spot weld diameter, i.e., whose influence is larger than the uncertainty on the measurement of the diameter, are taken into account.

By contrast, if the process of interest is a complex natural process (e.g., in biology or ecology), or if it is an economic, financial or social process, the choice of the relevant inputs may be more difficult. An example of a complex natural process (the solubility of molecules in solvents), where the determination of the relevant factors is not trivial, will be described in a subsequent section. Similarly, great care must be exercised in the choice of relevant inputs for credit rating, an example that will also be described below.

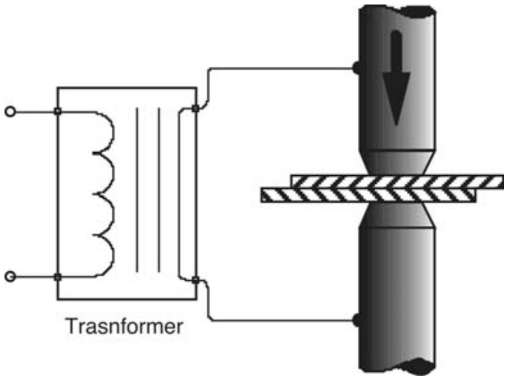


Fig. 1.13. A schematic representation of the spot welding process

The questions of input selection, and of model selection as well, are by no means specific to neural networks: they are of great importance for all modeling techniques, whether linear or nonlinear. It will be shown, in Chap. 2, that model selection techniques that were developed for linear models can be extended to nonlinear models such as neural networks.

1.2.2.2 Data Collection

Before training, observations must be collected in order to build the training set, as well as the validation and test sets, which will be defined below. Those observations must be numerous enough, and they must be typical of the situations that will be encountered by the network when in use. When the number of factors (model inputs) exceeds two or three, sampling the input domain in a regular and systematic way is generally not feasible because combinatorial explosion arises. Therefore, it is usually important to design the experiments as efficiently as possible: experimental design is an important part of model design. This is generally more difficult for nonlinear models than for linear ones. Some elements will be given in the "Experimental design" section of Chap. 2.

1.2.2.3 The Number of Hidden Neurons

The discrepancy between the neural approximation and the function to be approximated is inversely proportional to the number of hidden neurons [Barron 1993]; unfortunately, this result, as well as other theoretical results such as the Vapnik-Cervonenkis dimension (or VC-dimension) [Vapnik 1995] (described in Chap. 6) will only provide loose bounds or estimates of the number of hidden neurons. At present, no result allows the model designer to find the appropriate number of hidden neurons given the available data and the desired performance. Therefore, it is necessary to make use of a specific methodology. In the following, we will first define the problem of designing a nonlinear black-box static model, with emphasis on feedforward neural network design.

Overfitting and the Bias-Variance Dilemma

Since the accuracy of the uniform approximation of a given function by a neural network increases as the number of hidden neurons increases, a naïve design methodology would consist in building the network with as many neurons as possible. However, as mentioned above, in real engineering problems, the network is not required to approximate a known function uniformly, but to approximate an *unknown* function (the regression function) from a finite number of experimental points (the training set); therefore, the network should not only fit the experimental points as closely as possible (in the least squares sense), but it should also *generalize* efficiently, i.e., give a *satisfactory* response to situations that are not present in the training set. The difficulty here is that there is no operational definition of the meaning of *satisfactory*, since the regression function is unknown: the problem of generalization is an *ill-posed problem.* Therefore, the design problem is the following:

- if the neural network has too many parameters (it is said to be overparameterized), it will be too "flexible," so that its output will fit very accurately all points of the training set (including the noise present in these points), but it will provide meaningless responses in situations that are not present in the training set. That is known as *overfitting*.
- by contrast, a neural network with too few parameters will not be complex enough to match the complexity of the (unknown) regression function, so that it will not be able to learn the training data.

This dilemma, known as the *bias-variance dilemma*, is the basic problem that the model designer is faced with.

Figure 1.14 shows the results obtained after training two different networks, with different numbers of hidden neurons (hence of parameters) with sigmoid activation functions, from the same training set: clearly, the most parsimonious model (i.e., the model with the smallest number of parameters) generalizes best. In practice, the number of parameters should be small with respect to the number of elements of the training set. The parsimony of neural networks with sigmoid activation functions is a valuable asset in the design of models that do not exhibit overfitting.

Figure 1.14 shows clearly which candidate neural network is most appropriate. When the model has several inputs, the result cannot be exhibited graphically in such a straightforward fashion: a quantitative performance index must be defined. The most popular way of estimating such an index is the following: in addition to the training set, one should build a *validation set*, made of observations that are distinct from those of the training set, from which a performance index is computed. The most frequently used criterion is the mean square error on the validation set (VMSE), defined as:

$$\mathrm{VMSE} = \sqrt{\frac{1}{N_V}\sum_{k=1}^{N_V}\left[y^k - g(\boldsymbol{x}^k, \boldsymbol{w})\right]^2}$$

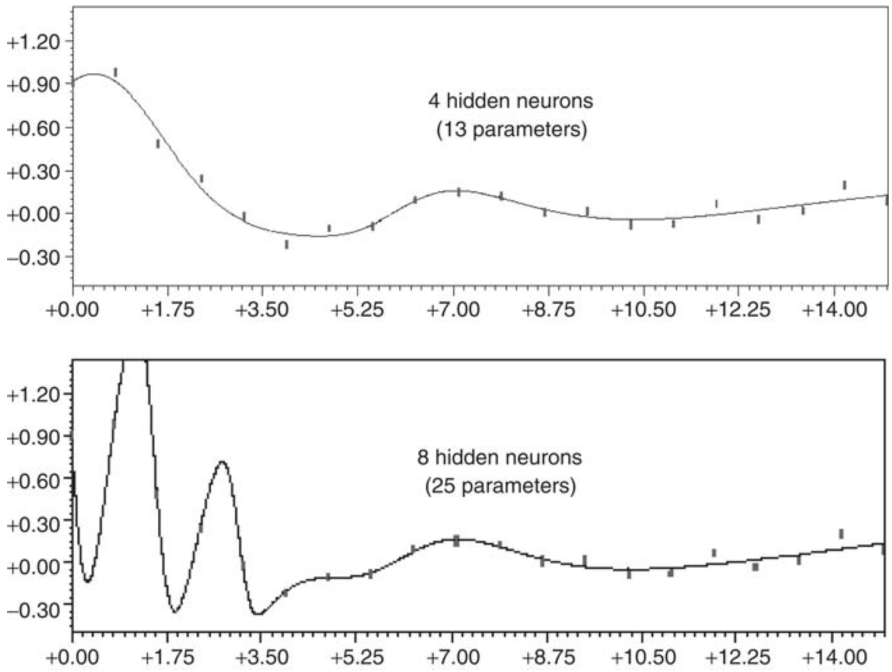


Fig. 1.14. The most parsimonious neural network has the best generalization abilities

where N_V is the number of observations present in the validation set, and where, for simplicity, y^k denote the measurements of the quantity to be modeled: $y^k = y_p(x^k)$. This relation is valid in the usual case of a model with a single output; if the model has several outputs, the VMSE is the sum of the mean square errors on each output.

This quantity should be compared with the mean square error on the training set (TMSE),

$$\mathrm{TMSE} = \sqrt{\frac{1}{N_T}\sum_{k=1}^{N_T}\left[y^k - g(\boldsymbol{x}^k, \boldsymbol{w})\right]^2},$$

where N_T is the number of observations present in the training set.

Consider the example shown on Fig. 1.14, and assume that the observations of the validation set are the midpoints between the observations of the training set. Clearly, the TMSE of the second network is certainly smaller than the TMSE of the first network, whereas the VMSE of the second network is certainly larger than that of the first network. Therefore, if model selection were performed on the basis of the training mean square error, overparameterized networks would systematically be favored, thereby leading to models that exhibit overfitting.

Note that if modeling were perfect, i.e., if the output of the model $g(\boldsymbol{x}, \boldsymbol{w})$ were identical to the regression function, and if the number of observations of the training set and of the validation set were very large, then both the TMSE and VMSE would be equal to the standard deviation of the measurement noise (provided NT and NV $\gg 1$). Therefore, the goal of modeling from examples can be expressed as follows: find the most parsimonious model (e.g., the most parsimonious feedforward neural network) whose TMSE and VMSE are on the same order of magnitude, and are as small as possible, i.e., on the order of magnitude of the standard deviation of the noise.

What to Do in Practice?

The purpose of this book is to provide practical methodologies, founded on sound theoretical bases, for model design through training, whether supervised or unsupervised. A complete methodology for supervised training will be described in Chap. 2 (together with complements in Chap. 3), a methodology for unsupervised training will be described in Chap. 7.

1.2.2.4 The Training of Feedforward Neural Networks: An Optimization Problem

Once the complexity of the model, i.e., the number of hidden neurons of a feedforward neural network, is chosen, training can be performed: one has to estimate the parameters of the neural network that, given the number of parameters that are available to him, has a minimum mean square error on the training set. Therefore, training is a *numerical optimization problem.*

For simplicity, we consider a model with a single output $g(\boldsymbol{x}, \boldsymbol{w})$. The training set contains N examples. The least squares cost function was defined above as

$$J(\boldsymbol{w}) = \frac{1}{2}\sum_{k=1}^{N}\left[y_p(\boldsymbol{x}^k) - g(\boldsymbol{x}^k, \boldsymbol{w})\right]^2,$$

where $\boldsymbol{x}^k$ is the vector of the values of the variables for example $k, y_p(\boldsymbol{x}^k)$ is the corresponding measured value of the quantity to be modeled, $\boldsymbol{w}$ is the vector of the parameters (or weights) of the model, and $g(\boldsymbol{x}^k, \boldsymbol{w})$ is the output value of the model with parameters $\boldsymbol{w}$ for the vector of variables $\boldsymbol{x}^k$. Therefore, the cost function is a function of all adjustable parameters $\boldsymbol{w}$ of the model. Training consists in finding the parameter vector $\boldsymbol{w}$ for which $J(\boldsymbol{w})$ is minimum.

- For a model that is linear with respect to its parameters (e.g., radial basis functions with fixed centers and widths, polynomials, etc.), the cost function J is quadratic with respect to the parameters: the ordinary least squares methods can be used. They are simple and efficient. However, the resulting models are not parsimonious.

- For a model that is not linear with respect to its parameters (e.g., a feedforward neural network, or a RBF network with adjustable centers and widths), the optimization problem is multivariable nonlinear, which makes ordinary least squares inapplicable. The techniques that solve such problems are described in detail in Chap. 2; those are iterative techniques that make sequences of estimations of the parameters until a minimum is reached, or a satisfaction criterion is met.

In the latter case, the optimization techniques are gradient methods; they are based on the computation, at each iteration, of the gradient of the cost function with respect to the parameters of the model. The gradient thus computed is subsequently used for updating the values of the parameters found at the previous iteration. *Backpropagation* is a popular, computationally economical way of computing the gradient of the cost function (described in Chap. 2). Therefore, backpropagation is *not* a training algorithm: it is simply a technique for computing the gradient of the cost function, which is very frequently an ingredient of neural network training. It has been often stated that the invention of backpropagation made feedforward neural network training possible; that is definitely not correct: methods for computing the gradient of cost functions were used in signal processing long before the introduction of neural networks. Such methods can be used for feedforward neural network training [Marcos 1992].

Training algorithms have been tremendously improved during the past few years. At the beginning of the 1990's, publications would frequently mention tens or hundreds of thousands of iterations, requiring days of computing on powerful computers. At present, typical trainings require tens or hundreds of iterations. Figure 1.15 displays the training of a model with a single variable. Crosses are the elements of the training set. Parameters are initialized to "small" values (see Chap. 2 for the description of the initialization procedure), so that the output of the network is essentially zero. The result obtained after 13 iterations is "visually" satisfactory; quantitatively, the TMSE and VMSE (the points of the validation set are not shown) are of the same order of magnitude, which is of the order of the standard deviation of the noise, so that the model is appropriate.

1.2.2.5 Conclusion

In this section, we have explained how and why neural networks with supervised training should be used. To summarize, neural networks are useful whenever a nonlinear relation between numerical data is sought. Therefore, neural networks are statistical tools for nonlinear regression. An overview of the tasks implied in nonlinear model design was presented, together with conditions for successful applications. In Chap. 2, the reader will find all necessary details for neural network training, for input selection and for model selection, both for static models (feedforward neural networks) and for dynamic model (recurrent neural networks).

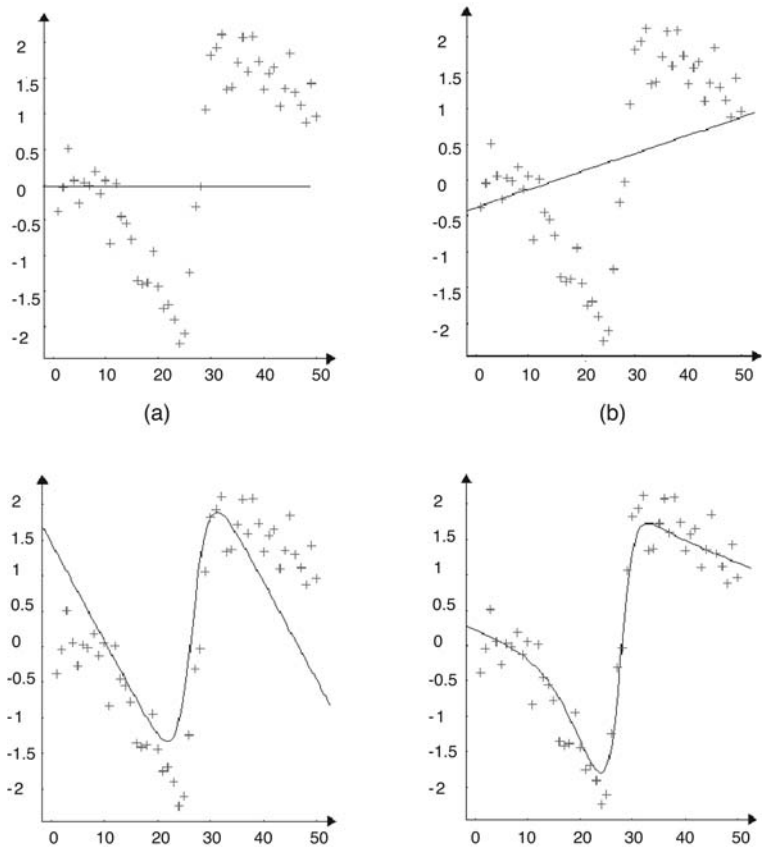


Fig. 1.15. Training of a feedforward neural network with one input variable and three hidden neurons. The line is the output of the model, the crosses are the elements of the training set. (**a**) initial state; (**b**) after one iteration; (**c**) after 6 iterations; (**d**) after 13 iterations (results obtained with the NeuroOne software package by NETRAL S.A.)

1.3 Feedforward Neural Networks and Discrimination (Classification)

In the early stages of the development of neural networks (in the years 1960's), the main incentive was the development of pattern recognition applications, as evidenced by the term perceptron that was used for the ancestor of present-day neural networks. Indeed, the first nontrivial industrial applications of neural networks, at the beginning of the 1980's, were related to pattern or signal recognition. Therefore, the present section is devoted to a general presentation of classification (or, equivalently, discrimination); it will be shown that many classification problems can be viewed as nonlinear regression problems, which

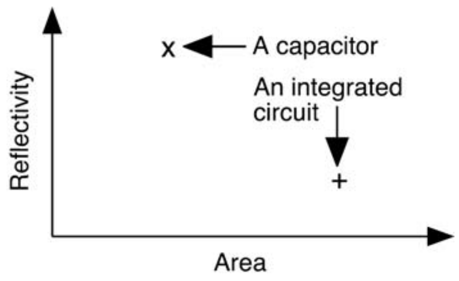


Fig. 1.16. Each sample is represented as a point in the area-reflectivity plane. Capacitors are shown as x's and integrated circuits as +'s

explains why feedforward neural networks are efficient classifiers. The purpose of the present section is to provide a general presentation of classification in its relation to nonlinear regression. Chapter 6 provides a much more detailed view of neural network classification and of techniques that evolved from neural networks.

1.3.1 What Is a Classification Problem?

A classifier is an algorithm that automatically assigns a class (or category) to a given pattern.

Before considering the specific case of "neural" classifiers, it is important to understand the basic characteristics of classification problems. Consider the following illustrative example: in an automatic sorting application, capacitors must be discriminated from integrated circuits, from a black-and-white picture provided by a video camera, so that a robotic arm can grab either a capacitor or an integrated circuit as requested. Roughly, capacitors appear in the picture as bright, small rectangular objects, whereas integrated circuits are large, dark objects. Therefore, the area A and the reflectivity R can be considered as relevant features for discriminating the objects, i.e., for assigning a given object either to the class "integrated circuit" or to the class "capacitor". Assume that samples of capacitors and of integrated circuits have been collected, and that their areas and reflectivities have been measured: then each sample can be represented by a point in a two-dimensional space, whose coordinates are its area and its reflectivity, as shown on Fig. 1.16.

1.3.2 When Is a Statistical Classifier such as a Neural Network Appropriate?

The above example shows that the ingredients of a classification problem are

- a set of N patterns;

- n variables (or features) that describe the patterns and are relevant to the classification task at hand, the set of descriptors of a given pattern building the *representation* of the pattern;
- a set of C classes to which the patterns should be assigned (one of the classes may be a rejection class in which all patterns that cannot be assigned to the other classes will be classified).

Therefore, solving a classification problem requires finding an application of the set of patterns to be classified into the set of classes.

It is important to realize that statistical classifiers such as neural networks are not appropriate for solving all classification problems: many alternative classification methods are available. The following (more or less academic) examples (from [Stoppiglia 1997]) illustrate the area of application of neural networks in classification. For each example, the following questions will be asked:

- Does prior knowledge suggest relevant features?
- Are those features measurable (or can they be computed from measurements)?
- What is the role of the rejection class?

Any vending machine can recognize the coins automatically, and reject fake or foreign coins. The answers to the above questions are

- Relevant features can easily be found: the coin diameter, its weight, its thickness, the alloy composition, etc.; there is a small number of such features, and new coins are actually designed in order to facilitate automatic discrimination.
- The features are measurable quantities.
- In feature space, the classes are small hyper-parallelepipeds defined by the manufacturing tolerances; the rejection class is the rest of feature space.

In such circumstances, a simple decision tree that operates with simple logical rules, derived from the analysis of the problem, can readily solve the classification problem. In such a case, statistical tools such as neural networks are not appropriate.

Vehicle comfort assessment can be viewed as a classification problem. In order to anticipate the reactions of customers to a new vehicle, car manufacturers resort to panels of customers, who are asked to express an opinion. Comfort is an ill-defined concept, which depends on many factors such as noise, seat design, etc. Assessing the comfort, for instance, by classifying it into three classes (very good, fair, poor), is a process that is difficult to formalize because it is based on feelings rather than on measurements.

- The relevant features are not necessarily known and clearly expressed by the customers; even if features could be defined, the assessments might be difficult to relate to the features; two customers, under the same conditions, could give very different assessments.

- The features are not measurable.
- There is no rejection class: all customers have an opinion on the comfort of a vehicle.

The fact that the features are not measurable precludes the use of a statistical method. In such a situation, a fuzzy classification method would be more appropriate.

Handwritten digit recognition, for instance zip code recognition, has been investigated in detail, and many applications are in routine operation. Consider the answers to the three questions that were asked in the previous two examples.

- In sharp contrast with the example of the vending machines, the huge diversity of handwriting styles makes the choice of features nontrivial, but feasible; in contrast to the vehicle comfort assessment problem, different persons who read the same digit will assign it to the same class (except if the digit is almost illegible).
- Features are numbers that can be extracted from the picture: in a typical low-level representation, the features would be the intensities of the pixels; in a high-level description, the features would be the location of horizontal, vertical or diagonal segments, the presence and location of loops, etc.
- The size of the rejection class can be defined, and in some cases, it is a performance criterion: for a given error rate, the rejection rate should be as low as possible. In mail processing, a rejected envelope requires a manual operation, which is less costly than sending a letter to the wrong address. Hence, the performance requirement is expressed as follows: for a given error rate (typically 1%) the rejection rate should be as low as possible. Clearly, it would be easy to design a classifier that never gives a wrong answer, by simply rejecting all patterns: by contrast, given the economic constraints of the problem of zip code reading, a "good" classifier makes a decision as often as possible, while making no more than 1% mistakes. If economic constraints were different, i.e., if a mistake was less costly than a human operation, a classifier should have the smallest possible error rate for a given maximum rejection rate (this is the case for large-scale automated medical diagnoses, where resorting to a medical doctor is more costly than delivering a wrong diagnostic).

In the latter example, statistical classification methods such as neural networks are perfectly appropriate, provided a suitable database is available. As in most nonacademic problems, the central question is that of data representation: a thoughtful representation design, together with data pre-processing techniques such as described in Chap. 3, is often as important as the classification algorithm itself.

1.3.3 Probabilistic Classification and Bayes Formula

Assume that, after analyzing a classification problem, a statistical classification approach has been deemed preferable to, for instance, a decision tree. Probabilistic classification methods are based on the idea that both features and classes may be modeled as random variables (readers unfamiliar with random variables will find more information at the beginning of Chap. 2). In that context, if a pattern is picked randomly from the patterns to be classified, the class to which it belongs is the realization of a discrete random variable. Similarly, the values of the features of a randomly chosen pattern can be viewed as realizations of random variable, which are usually continuous. For instance, in the example of discrimination between capacitors and integrated circuits (Fig. 1.16), the random variable "class" may be equal to 0 for a capacitor and to 1 for an integrated circuit, while the reflectivity R at the area A may be viewed as continuous random variables.

In that context, the classification problem can be simply stated as follows: given a pattern whose class is unknown, whose reflectivity is equal to r and whose area is equal to a (within measurement uncertainties), what is the probability that the random variable "class" be equal to 0 (i.e., that the pattern be a capacitor)? This probability is the *posterior probability* of class "capacitor" given the measured reflectivity and area, denoted by

$$\Pr(\text{class} = 0 \mid \{r, a\}).$$

Consider a set of capacitors and integrated circuits that have been labeled with the labels (0 or 1) of their classes, and whose feature values are also known. That information can be used for deriving two very important quantities,

- the *prior probability* of each class: a pattern picked randomly from the set of patterns has a probability $\Pr(C_i)$ of belonging to class C_i. It we assume that each pattern belongs to one of the classes, then one has $\sum_i \Pr(C_i) = 1$. That information is relevant to classification: assume that the prior probability of the class "capacitor" is known to be 0.9 (hence the probability of the class "integrated circuit" is 0.1); then a dumb classifier that would *always* choose the class "capacitor," irrespective of the pattern features, would exhibit an error rate on the order of 10%.
- the *conditional probability density of each feature*: if an integrated circuit is picked randomly, what is the probability for its area A to lie in an interval $[a - \delta a, a + \delta a]$? Clearly, that probability is proportional to δa. The *probability density of feature A conditioned to class C_i*, or *likelihood of C_i given feature a* is denoted as $p_A(a \mid C_i)$: the probability that feature A be in the interval $[a - \delta a, a + \delta a]$ given that it belongs to class C_i is equal to $p_A(a \mid C_i)\delta a$. Since the pattern whose feature A is measured belongs to class C_i, one has $\int p_A(a \mid C_i)\mathrm{d}a = 1$.

Figure 1.17 shows an estimate of the probability density $p_A(a \mid \text{Class} = \text{integrated circuit})$ as a function of a. Similarly, one could draw the conditional

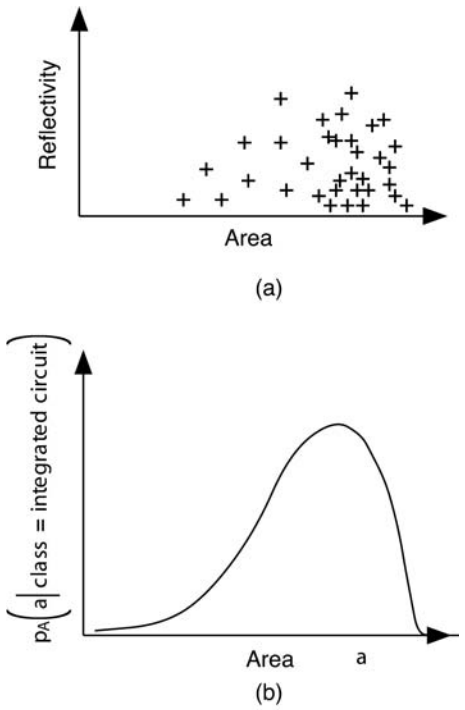


Fig. 1.17. (**a**) Representation of a sample of the class "integrated circuit" in the reflectivity-area plane. (**b**) Estimate of the conditional probability density of the area of the pattern if the latter is an integrated circuit

probability density of the reflectivity R, for the class integrated circuit, as a function of r.

Thus, given a sample of the population of patterns to be classified, estimates of the prior probabilities of the classes $\{\Pr(C_i)\}$, and of the conditional probability densities $p_X(\boldsymbol{x} \mid C_i)$ of their features, are available. Then, by Bayes formula, the solution of the classification problem, i.e., the posterior probability of a class given an unknown pattern, is given by

$$\Pr(C_i \mid \boldsymbol{x}) = \frac{p_X(\boldsymbol{x} \mid C_i)\Pr(C_i)}{\sum\limits_j p_X(\boldsymbol{x} \mid C_j)\Pr(C_j)}.$$

Clearly, that estimate is relevant only if the features of the unknown pattern have the same conditional density probabilities as the patterns that were used to estimate the likelihoods.

Note that

- if the prior probabilities are equal, the posterior probabilities are independent from the prior probabilities, so that the classification relies solely on the likelihoods of the classes;
- if the likelihoods are equal, i.e., if the features have no discriminative power whatsoever, the classification depends on the prior probabilities only.

Elegant though the Bayesian formulation may be, there is a major difficulty in its practical application: the estimation of the quantities in the right-hand side of Bayes formula. Obtaining a good estimate of the prior probabilities of the classes $\Pr(C_i)$ is generally an easy task, through simple frequency counting of each class in the sample. In contrast, the estimation of the likelihoods $p_X(x \mid C_i)$ is subject to a difficulty known as the *curse of dimensionality*: the number of patterns necessary for a reliable estimation of the likelihoods grows exponentially with the dimension of the feature vector. When low-level representations of the patterns are used, the number of features may be very large: if a picture is described by the intensity of its pixels, the dimension of the feature vector is equal to the number of pixels. We will show that neural networks are an interesting alternative to Bayesian classification because they provide a direct estimate of the posterior probabilities without having to estimate the prior class probabilities and the likelihoods.

Consider an application of Bayes formula: Assume that the probability distribution of the heights of women in a given population is Gaussian with mean 1.65 m and standard deviation 0.16 m,

$$p_H(h \mid W) = \frac{1}{0.16\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{h-1.65}{0.16}\right)^2\right),$$

and that the probability distribution of the heights of men in that population is a Gaussian with mean 1.75 m and standard deviation 0.15 m:

$$p_H(h \mid M) = \frac{1}{0.15\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{h-1.75}{0.15}\right)^2\right).$$

The above probability densities are shown on Fig. 1.18. The Gaussians exhibit strong overlapping, which shows that the feature height is not very discriminant. In a real application, such curves would be a strong incentive for the designer to find one or more alternative features.

In addition, assume that there are as many men and women in the population. Given a person whose height is 1.60 m, what is the probability that it is a woman? The answer is provided by Bayes formula

$$\Pr(W \mid 1.60) = \frac{0,5 p_H(1.60 \mid W)}{0.5 p_H(1.60 \mid W) + 0.5 p_H(1.60 \mid M)} \approx 60\%.$$

Clearly, $\Pr(M \mid 1.60) = 40\%$.

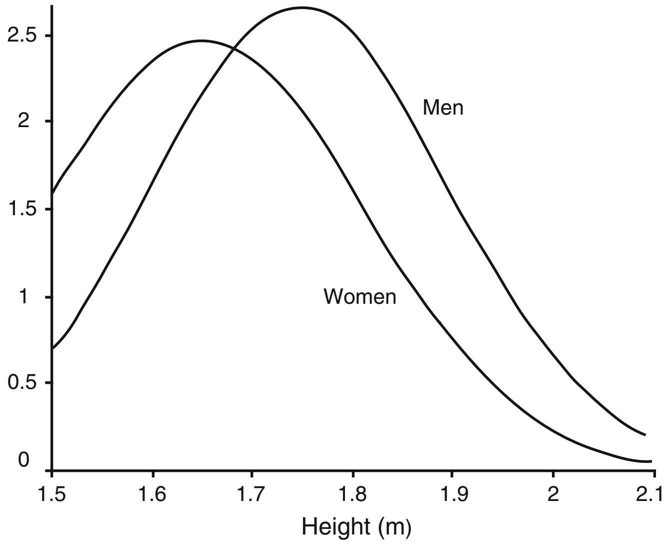


Fig. 1.18. Probability densities of heights of men and women for the population under investigation

In view of the above results, it is natural to assign the person to class W, which has the larger probability. This is an application of Bayes decision rule, which will be explained below. The boundary between the classes thus defined is shown on Fig. 1.19.

Because the prior probabilities are assumed to be equal, the discrimination relies solely on the likelihoods.

Now assume that the person is not a member of the general population, but is picked among the audience of a football match. Then the likelihoods of the classes, given the height, are the same as above, but the prior probabilities are different, since men are generally more numerous than women in the audience of football matches; assume that the proportions are: 30% of women and 70% of men. The posterior probabilities, as computed from Bayes formula, become $\Pr(W \mid 1.60) = 39\%$ and $\Pr(M \mid 1.60) = 61\%$. The results are very different from the previous ones: the observed person is assigned to the class man if Bayes rule is used as above; that important change results from the fact that the likelihoods are not very different because the feature height is not very discriminant, so that the classification relies heavily on the prior probabilities. That result is illustrated by Fig. 1.20.

That simple example shows how to use Bayes formula for estimating posterior probabilities, which are subsequently used for assigning each pattern to a class through Bayes decision rule.

It is important to realize that, in practice, and in contrast with the above examples, prior probabilities and probability densities are not known analytically, but are estimated from a finite set of observations O. Therefore,

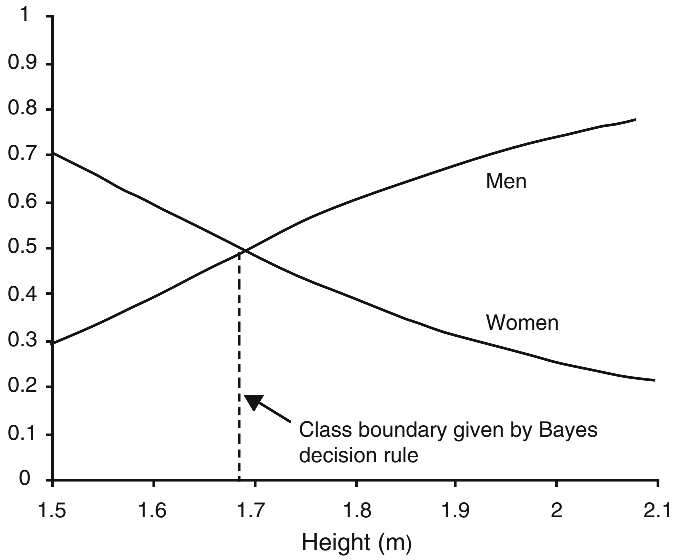


Fig. 1.19. Posterior probabilities of the classes man and woman, as a function of height, and boundary between the classes, when the person under investigation is drawn from the general population

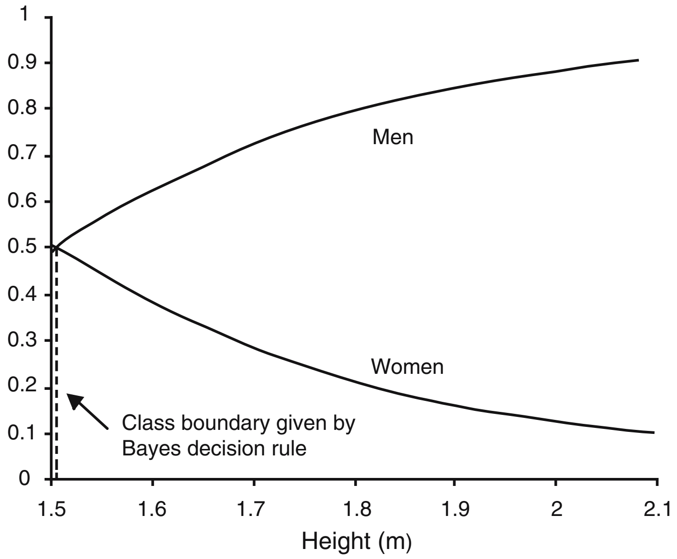


Fig. 1.20. Posterior probabilities of the classes man and woman, as a function of height, and boundary between the classes, when the person under investigation is drawn from the audience of a football match

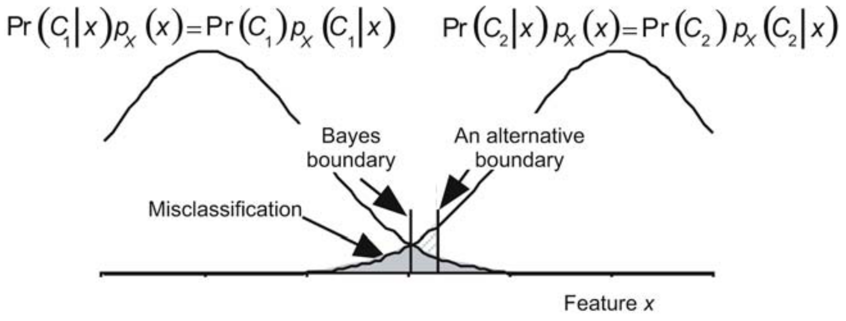


Fig. 1.21. A geometrical interpretation of Bayes decision rule; the *gray area* is the probability of misclassification when Bayes rule is used; the *striped area* is the increase of misclassification probability resulting from a different boundary choice

the likelihood should be denoted as $p_X(x \mid C_i,\, O)$, and the posterior probabilities should be denoted as $\Pr(C_i \mid x, O)$, since their estimates depend on the observation set O. For simplicity, we will not use these notations, but it should be remembered that the estimates of probabilities and of probability distributions are always conditioned to the observations from which they are estimated.

1.3.4 Bayes Decision Rule

When assigning a pattern to a class, the risk of making a classification error is minimum if the pattern is assigned to the class whose posterior probability is highest.

Consider a classification problem with two classes C_1 and C_2, and one feature. Clearly, the probability of misclassification is larger if the pattern lies close to the class boundary. However, during the normal operation of the classifier, it will handle patterns that are described by a large range of values of the feature, so that what one would really like to do is to find the boundary that minimizes the global error probability, i.e., the boundary that minimizes the quantity $\Pr(M) = \int_{-\infty}^{+\infty} \Pr(M \mid x) p_X(x) \mathrm{d}x$, where M denotes the event “misclassification”. Since the probability density $p_X(x)$ is positive, the integral is minimal if $\Pr(M \mid x)$ is minimal for all x. $\Pr(M \mid x)$ is the posterior probability of C_1 if the decision is made of assigning the pattern to C_2, and the posterior probability of C_2 if the decision is made of assigning the pattern to C_1. Therefore, $\Pr(M \mid x)$ is minimized if the decision is to assign the pattern to the class with higher probability.

A geometrical interpretation of that argument is shown on Fig. 1.21: if Bayes rule is used, the misclassification probability is represented by the gray area. Any other boundary choice would increase that area.

The result can be easily extended to the multi-class case and the multi-feature case.

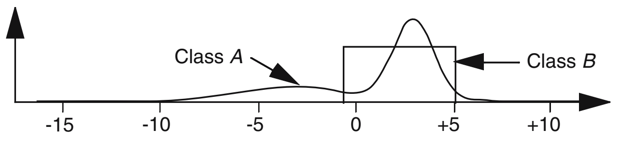


Fig. 1.22. Probability densities for classes A and B

That decision rule is satisfactory if the misclassification costs are the same for the two classes; however, one frequently encounters applications where it may more detrimental, or more costly, to make a false-positive misclassification (the pattern is considered to belong to class A whereas it actually belongs to class B) than a false-negative misclassification (the pattern is considered to belong to class B whereas it actually belongs to class A). In data mining applications for instance, a company that provides information filters may find it more suitable to market filters that reject documents whereas they are relevant to the chosen topic, than to market a filter that does not filter irrelevant documents (the user spots immediately documents that are irrelevant, whereas he may never find out that the filter missed a relevant text). In practice, such considerations are an important part of classifier design, whether in pattern recognition, data mining, credit scoring, etc.). Therefore, it is generally very desirable, in practical applications, to estimate posterior probabilities and subsequently make decisions: classifiers that determine class boundaries directly may lead to serious misconceptions.

The combination of Bayes formula and of Bayes decision rule is called the Bayes classifier, which has the best achievable performance if the prior probabilities and the likelihoods are known exactly. Since the latter condition is not frequently fulfilled in practice, Bayes classifier is essentially of theoretical interest. For instance, it may serve as a reference for assessing the quality of a classifier, by applying it to an academic problem where prior probabilities and likelihoods are known exactly.

As an illustrative example, consider a problem with two classes and one feature; the patterns of class A are generated from a mixture of two Gaussians; the patterns of class B are generated from a uniform distribution in a bounded interval (Fig. 1.22). Therefore, the posterior probabilities can be computed exactly (Fig. 1.23), and so are the boundaries between classes (Fig. 1.24). In

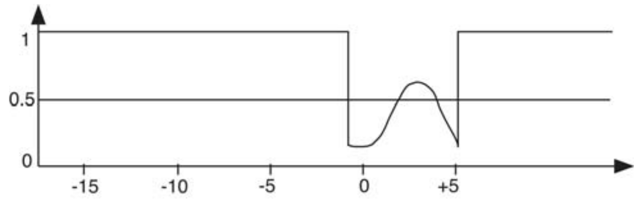


Fig. 1.23. Posterior probability of class A, from Bayes formula

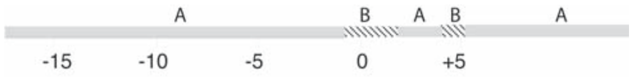


Fig. 1.24. Classification achieved by Bayes classifier

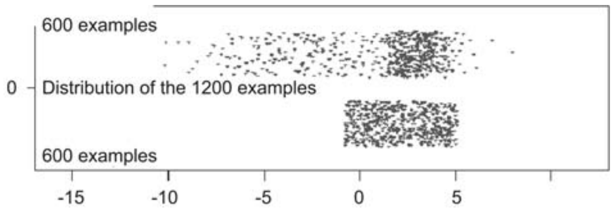


Fig. 1.25. Examples used for estimating the misclassification rate. *Top*: class A; *bottom*: class B

order to estimate the misclassification rate of the resulting Bayes classifier, a large number of realizations of examples of each class are generated, and the proportion of misclassified examples is computed. 600 examples of each class were generated (Fig. 1.25), and, by simple counting, the misclassification rate was estimated to be equal to 30.1%. Therefore, it can be claimed that no classifier, however carefully designed, can achieve a classification performance higher than 69.9%. The best classifiers are the classifiers that come closest to that theoretical limit.

1.3.5 Classification and Regression

The previous section was devoted to the probabilistic foundations of classification. We are going to show why neural networks, which are regression tools, are relevant to classification tasks.

1.3.5.1 Two-Class Problems

We first consider a problem with two classes C_1 and C_2, and an associated random variable Γ, which is a function of the vector of descriptors $\boldsymbol{x}$; that random variable is equal to 1 when the pattern belongs to class C_1, and 0 otherwise. We prove the following result: the regression function of the random variable Γ is the posterior probability of class C_1.

The regression function $y(\boldsymbol{x})$ of variable Γ is the expectation value of Γ given $\boldsymbol{x} : y(\boldsymbol{x}) = E(\Gamma \mid \boldsymbol{x})$. In addition, one has:

$$E(\Gamma \mid \boldsymbol{x}) = \Pr(\Gamma = 1 \mid \boldsymbol{x}) \times 1 + \Pr(\Gamma = 0) \times 0 = \Pr(\Gamma = 1 \mid \boldsymbol{x})$$

which proves the result.

Neural networks are powerful tools for estimating regression functions from examples. Therefore, neural networks are powerful tools for estimating posterior probabilities, as illustrated on Fig. 1.26 this is the rationale or performing

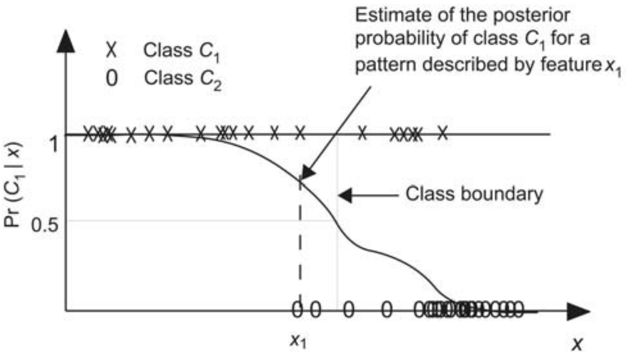


Fig. 1.26. Estimate of the posterior probability of class C_i, and boundary between classes from Bayes decision rule

classification by neural networks. A lucid and detailed description of that approach is given in C. Bishop's excellent book [Bishop 1995].

1.3.5.2 *C*-Class Problems

When the number of classes involved in a classification problem is larger than two, two strategies can be implemented, i.e.,

- find a global solution to the problem by simultaneously estimating the posterior probabilities of all classes;
- split the problem into two-class subproblems, design a set of pairwise classifiers that solve the subproblems, and combine the results of the pairwise classifier into a single posterior probability per class.

We will consider those strategies in the following subsections.

Global Strategy

That is the most popular approach, although it is not always the most efficient, especially for difficult classification tasks. For a C-class problem, a feedforward neural network with C outputs is designed (Fig. 1.27), so that the result is encoded in a 1-out-of-C code: the event "the pattern belongs to class C_i" is signaled by the fact that the output vector $\boldsymbol{g}$ has a single nonzero component, which is component number i. Similarly to the two-class case, it can be proved that the expectation value of the components of vector $\boldsymbol{g}$ are the posterior probabilities of the classes.

In neural network parlance, a *one-out-of-C* encoding is known as a *grandmother code*. That refers to a much-debated theory of data representation in nervous systems, whereby some neurons are specialized for the recognition of usual shapes, such as our grandmother's face.

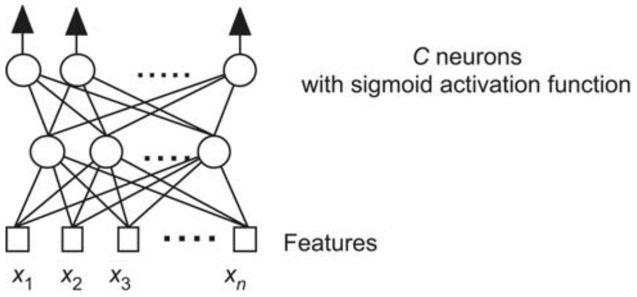


Fig. 1.27. A multilayer Perceptron with C outputs for classification. The activation functions of the output neurons are sigmoids

There are several important differences between a multilayer perceptron for classification and a multilayer perceptron for regression.

- The activation functions of the output neurons of neural networks for modeling is usually linear; by contrast, the output neurons of neural networks for classification have nonlinear activation functions such as sigmoids: since the outputs of the neural network are probabilities, they must lie between 0 and 1 (readily amenable to $[-1, +1]$); in Chap. 6, a theoretical justification for the use of the tanh function as an activation function of output neurons will be given,
- For classification, minimizing the cross-entropy cost function is more natural than minimizing the least squares cost function [Hopfield 1987; Baum 1988; Hampshire 1990]; the training algorithms that will be described in Chap. 2 can readily be applied to this cost function,

$$J = -\sum_k \sum_{i=1}^{C} \gamma_i^k \mathrm{Log}\left[\frac{g_i(\boldsymbol{x}^k)}{\gamma_i^k}\right] + (1-\gamma_i^k)\mathrm{Log}\left[\frac{1-g_i(\boldsymbol{x}^k)}{1-\gamma_i^k}\right].$$

  where γ_i^k is the desired value (0 or 1) for output i when the classifier's input is example k, described by feature vector $\boldsymbol{x}^k$, and $g_i(\boldsymbol{x}^k)$ is the value of output i of the classifier. That function is minimum when all examples are correctly classified.

After training, it is safe to check that the sum of the outputs is equal to 1 for all examples. The Softmax technique [Bridle 1990] guarantees that the above condition is fulfilled automatically. Of course, that is not a problem for pairwise classifiers, which have a single output.

The question of overfitting, which we have encountered in nonlinear regression, is also valid for discrimination. If the classifier is overparameterized, it separates very accurately the patterns of the training set and has a poor generalization ability. Model selection techniques, such as those described in Chap. 2, must be used in order to select the best model. Essentially, one must

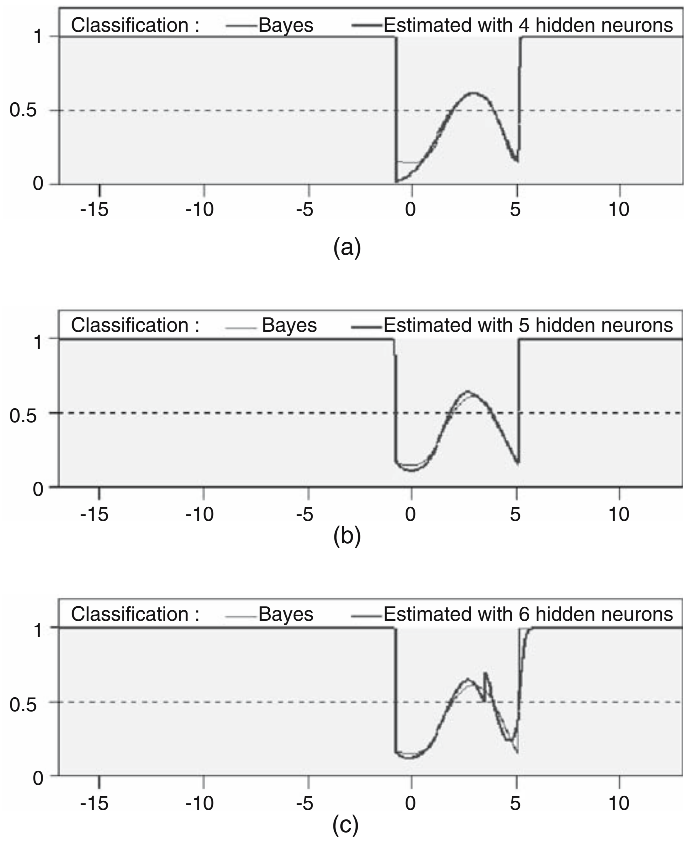


Fig. 1.28. Estimation of posterior probabilities of class A with three classifiers: (**a**) 4 hidden neurons (too low complexity), (**b**) 5 hidden neurons (performance very close to the best achievable correct classification rate, (**c**) 6 hidden neurons (strong overfitting)

find a classifier whose classification error rates are of the same order of magnitude, and as small as possible, on the training set and on an independent validation set. Figure 1.28 shows an example of overfitting in the estimation of the posterior probability of class A in the example shown on Fig. 1.23; clearly, the network with 4 hidden neurons is not complex enough for representing the posterior probability, whereas a neural network with 6 hidden neurons fits the fluctuations of the densities of points of the training set. The neural network with 5 hidden neurons has a misclassification rate of 30.3% (estimated on a test set of several million points), while the minimum achievable error rate, from Bayes classifier, is 30.1%. Therefore, neural networks are among the best classifiers.

Pairwise Classification

For difficult problems, it is often much safer to split a C-class classification problem into $C(C-1)/2$ pairwise classification problems, for the following reasons:

- When performing pairwise classification, the designer can take advantage of many theoretical results and algorithms, pertaining to linear class separation; they are fully developed in Chap. 6; we give a cursory introduction to that material in the next section, entitled linear separability.
- The resulting networks are much more compact, with fast training and simple analysis; since each network has a single output, its probabilistic interpretation is trivial.
- The features that are relevant for separating class A from class B are not necessarily identical to the features that are relevant for separating class A from class C; therefore, each classifier has only the inputs that are relevant to its own task, whereas a multilayer Perceptron for global separation must have all input features that are relevant for the discrimination of all classes; the feature selection techniques that are described in Chap. 2 can be used in a very straightforward fashion.

Once the $C(C-1)/2$ posterior probabilities are estimated, possibly with simple linear separators (neural networks with no hidden neuron), the posterior probability of class C_i for a feature vector $\boldsymbol{x}$ is computed as

$$\Pr(C_i \mid \boldsymbol{x}) = \frac{1}{\displaystyle\sum_{j=1,\, j\neq i}^{C} \frac{1}{\Pr_{ij}} - (C-2)},$$

where C is the number of classes and $\Pr_{ij}$ is the posterior probability of class i or class j, as estimated by the neural network that separates class C_i from class C_j.

Linear Separability

Two sets of patterns, described in an n-dimensional feature space, belonging to two different classes, are said to be "linearly separable" if they lie on different sides of a hyperplane in feature space.

If two sets of examples are linearly separable, a neural network made of a single neuron (also termed perceptron can separate them. Consider a neuron with a sigmoid activation function with n inputs; its output is given by $y = \mathrm{th}\left[\sum_{i=1}^{n} w_i x_i\right]$. The simple relation $P = (y+1)/2$ provides an interpretation of the output of the classifier as a posterior probability. From Bayes decision rule, the equation of the boundary between the classes is given by $P = 0.5$, or equivalently $y = 0$. Therefore, the separating surface is a hyperplane in

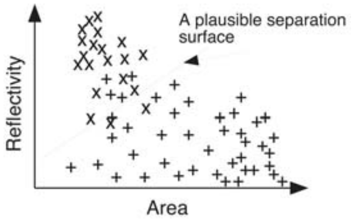


Fig. 1.29. Linear separation by a Perceptron (neural network with a single output, without hidden neurons: 10% misclassification rate

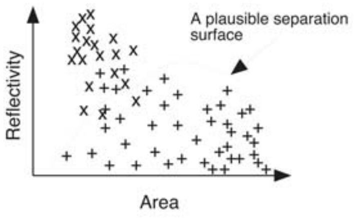


Fig. 1.30. Separation by a network with a small number of hidden neurons. Three examples per class are misclassified

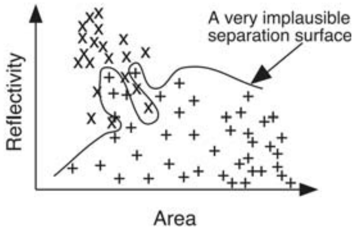


Fig. 1.31. Separation by an overparameterized neural network. All examples are correctly classified, but the generalization capacity is low

n-dimensional space,

$$v = \sum_{i=1}^{n} w_i x_i = 0.$$

Hence, $v > 0$ for all examples of one of the classes, and $v < 0$ for all examples of the other class. Figure 1.29 shows a separation surface that can be defined by a Perceptron, for the example of capacitors and integrated circuits.

Hidden neurons allow multilayer Perceptrons to define more complex separation surfaces, as shown on Fig. 1.30.

As usual, one can obtain zero misclassifications if enough hidden neurons are added, but that is detrimental to generalization because of overfitting. Fig. 1.31 illustrates a case of blatant overfitting.

When a multi-class problem is split into pairwise separation problems, linear separation between two classes is often complex enough; very frequently, in multi-class problems that are claimed to be difficult, the examples turn out to be pairwise linearly separable. In such cases, powerful, elegant algorithms give excellent solutions, as explained in Chap. 6: therefore, the first step in the design of a classifier consists, in checking whether the classes are pairwise linearly separable. Ho and Kashyap's algorithm [Ho 1965], which was discovered long before neural networks came into existence, provides an answer to that problem in finite time:

- If the examples are linearly separable, the algorithm finds a separating hyperplane in finite time.
- If the examples are not linearly separable, the algorithm signals it infinite time (see algorithmic complements at the end of this chapter).

The postal code database provided by the National Institute of Standards and Technology has served as a basis for many classifier designs. It turns out that, even with low-level representations such as a pixel representation, the classes of examples are pairwise linearly separable [Knerr 1992]! Similarly, a famous sonar signal data base has been investigated in great detail by many authors, and many complicated classifiers were designed to solve that two-class problem; actually, in less than ten minutes on a PC, the Ho and Kashyap algorithm, implemented as an uncompiled Matlab program, proves that the examples of the two classes are linearly separable. Therefore, a simple Perceptron can solve the problem, without resorting to any hidden layer. That application is investigated in more detail in Chap. 6.

1.3.5.3 Classifier Design Methodology

From the previous section, the following strategy for the design of a neural classifier can be derived (as discussed above, one should first ascertain that statistical classification is relevant to the problem at hand):

- Find an appropriate representation of the patterns to be classified, especially for pattern recognition (the techniques described in Chap. 3 can be especially useful in that respect); this is a crucial step, since a representation that has a high discriminative power is likely to make the classification problem trivial; this is illustrated in applications described below.
  If the number of examples available for training the classifier is not larger than the dimension of the feature vector, there is no point in pursuing the design any further, according to Cover's theorem [Cover 1965], which is explained in Chap. 6: before proceeding to the next steps, either a more "compact" representation must be found, or additional examples must be collected, or a very stringent regularization method such as weight decay (described in Chap. 2), must be implemented.
- For each pair of classes, select the relevant features with the feature selection methods described in Chap. 2; obviously, the discrimination of class A from class B may not require the same features as the discrimination of class A from class C.

- For each pair of classes, test the pairwise linear separability of the examples with the Ho and Kashyap algorithm.
- For all pairwise linearly separable classes, make use of linear separation methods (described in Chap. 6) and derive an estimate of the posterior probabilities.
- For nonlinearly separable classes, design small multilayer Perceptrons, or spherical perceptrons as described in Chap. 6, with probability estimations; use leave-one-out or cross-validation techniques for model selection (see Chap. 2).
- Estimate the global posterior probabilities of each class from the pairwise probabilities estimated at the previous steps, using the relation indicated in Sect. 1.3.5.2, subsection "pairwise classification" above.
- Determine the decision thresholds in order to define rejection classes.

That strategy is a variant of the STEPNET procedure [Knerr 1990, 1991], which allowed the design of several industrial applications.

In the planning of a classification project, the time required by the first and the last steps of the above strategy should definitely not be underestimated; for nontrivial applications, those are frequently the lengthiest and most painful steps.

The applicability of that strategy is limited by the fact that the number of pairwise classifiers grows as the square of the number of classes. However, each classifier is usually very simple, so that the procedure can be applied with up to a few tens of classes. For larger number of classes, hierarchical strategies must be resorted to.

1.4 Some Applications of Neural Networks to Various Areas of Engineering

1.4.1 Introduction

The present book is intended to assist the engineer or researcher in answering the following question: can neural networks solve my problem, and can they do it *more efficiently* (in terms of accuracy, computation time, etc.) than other techniques?

Contributions to a rational answer were provided at the beginning of the present chapter, where we explained the mathematical foundations and principles underlying the operation of neural networks. Although some elements may look somewhat technical, they are mandatory for getting an in-depth understanding of what one can and cannot do with neural networks. Since the software implementation of neural networks is straightforward with present-time tools, one might be tempted to apply neural networks without prior thinking, which may lead to disappointing results.

In addition to mathematical arguments, it may be helpful, in order to illustrate the use and limitations of neural networks, to describe some typical industrial applications. This is by no means intended to be an exhaustive presentation of neural network applications, which would require several books. The point here is to show some typical examples, and to stress the reason why neural networks made important, possibly decisive, contributions.

1.4.2 An Application in Pattern Recognition: The Automatic Reading of Zip Codes

Character recognition is definitely the application area where neural networks made their first significant contributions to engineering, proving to be reliable alternatives to classical pattern recognition methods. In the present section, some examples and results will be described, relying on the elements of theory and methodology provided in the previous section.

The automatic reading of postal codes is probably one of the most widely investigated problems in picture recognition. The automatic reading of printed envelopes and parcels is a relatively simple problem; by contrast, the huge variety of handwritings made the recognition of handwritten addresses a truly challenging problem. For each item handled by the postal service, a machine must either recognize the code, or resort to human inspection when it fails to identify the code. As indicated above, correcting a sorting mistake made by a machine is more costly than resorting to human inspection for reading and typing in the correct code; therefore, the most frequently used performance criterion for such machines is the following: given a maximum misclassification rate (say 1%), what fraction of the mail must be read by a human operator? At present, typical performances are 5% rejection rate for 1% misclassification.

The development of automatic zip code reading was primarily spurred by the industrial importance of the problem, but also by the fact that, as early as 1990, large-scale data bases were made available to the general public by the United States Postal Service (USPS), and later by the National Institute of Science and Technology (NIST). That policy allowed many laboratories, both in industry and in universities, to improve the state of the art, and to validate, in a statistically significant way, the methods and procedures that they developed; it had a general positive impact on the development of powerful classification methods.

Figure 1.32 displays some examples from the USPS database, which features 9,000 digits (which is not a very large number, considering the variety of handwriting styles). The difficulty of the problem is immediately apparent. Consider the postal code in the upper right corner of the picture; one reads 68544 effortlessly, but one notes

- that the digit 6 is split into two parts,
- that digits 8 and 5 are linked together,
- that digit 5 is split into two parts, the right part being linked to digit 4!

Thus, if one decides to base the recognition of the code on the recognition of its individual digits, the problem of segmentation must be solved first: how does

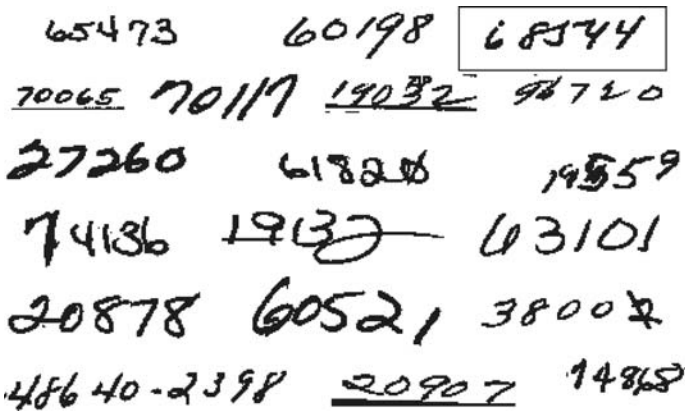


Fig. 1.32. Some excerpts from the NIST handwritten digit database

one decompose a zip code into five separate digits? Having solved that difficult problem, one must cope with the variety of styles, sizes, and orientations, of the isolated digits. To that end, an appropriate representation must be found; the design of an appropriate representation is completely problem-dependent, and requires new efforts for each new application. Clearly, one cannot use the same kind of representations for pictures such as handwritten or printed digits, satellite pictures, or X-ray medical images.

Despite the diversity of image processing techniques, some basic operations are found in essentially all applications, as well as in the human visual system: edge detection, contrast enhancement, etc. In the field of handwritten character recognition, normalization is mandatory, in order to apply the recognition algorithm to characters of similar sizes. We have already mentioned that the design of a real application requires a tradeoff between the complexity of the preprocessing that is necessary to yield the chosen representation, and the complexity of classification: a carefully designed preprocessing, which leads to very discriminant features, may allow the use of a very simple classifier, but the preprocessing must not be too demanding in terms of computation time. By contrast, a simple preprocessing, such as normalization alone, may be very fast but will not alleviate the task of the classifier. Thus, one must engineer the best tradeoff that allows meeting the requirements of the application. Two very different approaches to the same application will be described below.

The first approach was developed at the former AT&T Bell Labs. It consists of a neural network, known as LeNet [Le Cun 1991], which makes use of a pixel representation (after normalization). The first layers of the network perform local preprocessings that aim at automatically extracting features that are relevant to classification, while the final layers perform classification. The network is shown on Fig. 1.33.

The network is input with a 16×16 pixel intensity matrix. A first layer of hidden neurons is made of 12 sets of 64 neurons each, where each set of

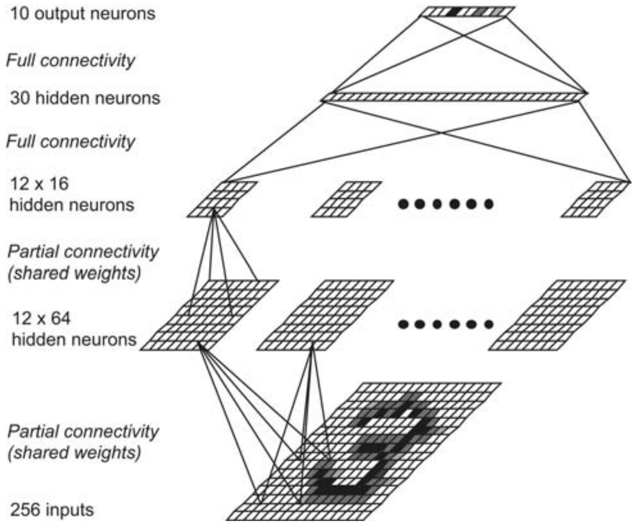


Fig. 1.33. LeNet, a neural network that performs feature extraction and classification

64 hidden neurons receives information from a "receptive field" of 5×5 pixels. Those sets of 64 neurons are called *feature maps*, for the inputs to a given map have the same weights (this is known as the "shared weights" technique, described in Chap. 2): thus, the same operator acts locally on a 25-pixel area of the picture, so that the outputs of a group of 64 neurons are the results of the application of the same operator to the receptive fields. The local operator technique is classical in picture processing, but the present approach is original in that these operators are not engineered, but are "discovered" through training by examples. The same technique is iterated by a second layer of operators that act on the results of the first layer. Thus, 12 maps of 16 hidden neurons are produced by 192 neurons that provide the representation of the digit. Classification is performed by a final layer of 30 hidden neurons, followed by 10 output neurons using a 1-out-of-N code: the number of outputs is equal to the number of classes, output neuron number i must be active if the input digit belongs to class i, and inactive otherwise.

Thus, the network performs, automatically and simultaneously, feature extraction and classification, whereas those operations are usually performed in a sequential fashion. The flexibility of the method has a price: given the size of the network, training is demanding, and, because of the large number of parameters, the network will be prone to overfitting.

In order to solve the same problem, a very different approach was implemented [Knerr 1992], which consists in performing a more elaborate preprocessing of the picture, in order to extract discriminant characteristics that lead to a relatively simple classifier. Preprocessing consists of edge extraction, followed by normalization, which produce 4 feature maps of 64 elements, hence

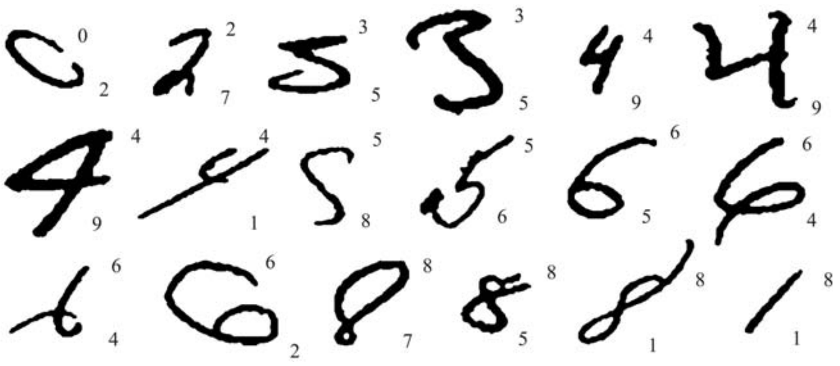


Fig. 1.34. The 18 misclassifications made by pairwise linear separation of the classes. For each digit, the superscript is the label of the digit in the base, and the subscript is the response of the classifier

a 256-component feature vector. Following the classifier design methodology described in the previous section, pairwise classification was performed by 45 different classifiers. Since the training sets were pairwise linearly separable, each classifier consisted of a single neuron; the classifiers were trained separately.

Figure 1.34 shows the 18 misclassifications made by the classifier on the 9,000 digits of the USPS database. Note that the bottom right digit is (correctly) recognized as a 1, whereas it was mistakenly labeled as an 8 in the database!

We have emphasized the impact of the choice of the representation on the efficiency of classification. This point can be nicely illustrated with this application. For the two representations that were mentioned above (representation by the intensities of 256 pixels, and representation by 4 feature maps of 64 elements each after edge detection), the euclidean distances between the centers of gravity of the classes were computed, and reported on Fig. 1.35. The inter-class distances are systematically larger, with the feature-based representation, than with the pixel-based representation. Thus, the feature-based representation increases the inter-class distances in input space, thereby making classification easier.

Table 1.1 illustrates the performance improvement resulting from the use of a more appropriate representation: after adjusting the decision thresholds so as to get, for both representations, a 1% misclassification rate, the rejection rate is much higher for the pixel representation than for the feature-map based representation. Note that since, in both cases, the dimension of input space is 256, the improvement does not result from the fact that the representation is more compact, but from the fact that it is more appropriate to the task. As usual, better engineering provides better results.

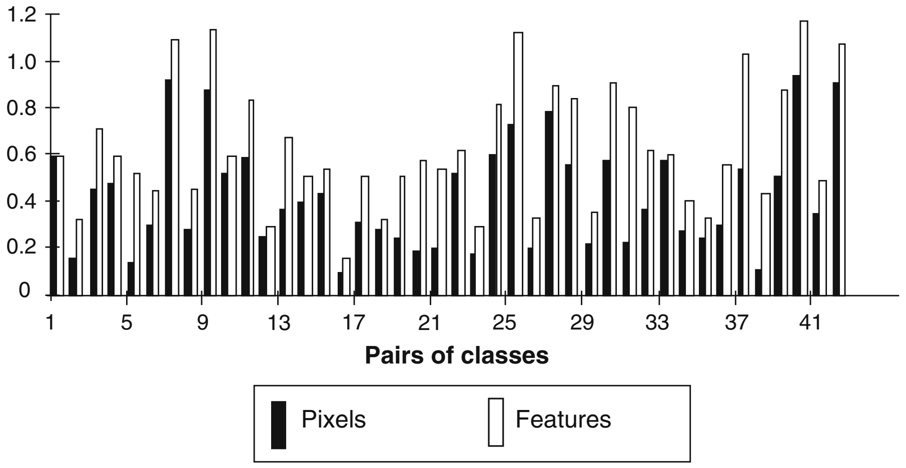


Fig. 1.35. Distances between classes for two representations: the feature-map based representation makes the classes more widely separated in input space, thereby making the classification task easier

1.4.3 An Application in Nondestructive Testing: Defect Detection by Eddy Currents

The example that was presented in the previous section used classification for picture recognition. Of course, patterns that can be recognized automatically vary widely in nature. The application that we describe in the present section pertains to nondestructive testing, where the patterns to be classified are signals. The objective is the automatic detection of defects in the rails of the Paris subway. It was developed by the National Research Institute on the Safety of Transportation Systems for RATP, the company that operates the Paris underground system [Oukhellou 1997].

Defect detection in metal parts by eddy currents is a standard nondestructive testing technique. An electromagnetic coil creates an alternating magnetic field, which generates eddy currents in the metal part to be tested. Those currents are detected by a second coil, and the presence of defects in the metal alters the amplitude and the phase of the resulting signal. Thus, that signal

Table 1.1.

	Correct classification rate (%)	Rejection rate (%)	Misclassification rate (%)
Pixel representation	70.9	28.1	1
Feature-map-based representation	90.3	8.7	1

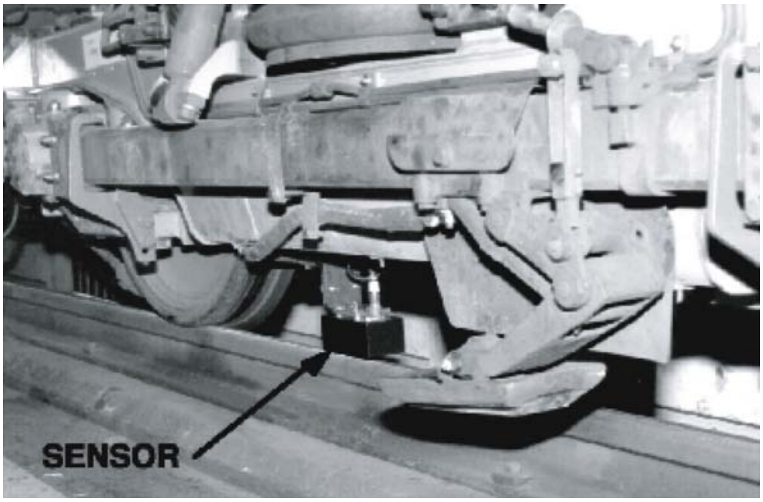


Fig. 1.36. The eddy current generation and detection system

contains a "signature" of the defects. Since there are different categories of defects, which may be more or less detrimental to the operation of the system, classifying the defects is generally desirable. In the present case, it is also important to be able to discriminate between real defects and normal phenomena that are also detected by the eddy current technique, such as the presence of a weld joint or of a switch (the position of the latter is known, which makes discrimination easier).

In the present application, the system that generates and detects eddy currents is mounted below the carriage, a few tens of millimeters above the rail, as shown on Fig. 1.36.

As usual, the choice of the descriptors of the signal is crucial for the efficiency of discrimination. In the present case, a relatively small number of features, derived from Fourier components of the signal, are usually sufficient, provided they are chosen appropriately. Feature selection was performed by the "probe feature method," which is described in Chap. 2 [Oukhellou 1998].

1.4.4 An Application in Forecasting: The Estimation of the Probability of Election to the French Parliament

After the elections to the French parliament, all candidates must make an official statement of the amount of the expenses incurred during the campaign, and of the breakdown of those expenses. Making use of the data pertaining to the 1993 elections, it was possible to assess the probability of being elected as a function of the expenses and of their breakdown. This is a two-class classification problem, and neural networks provide an estimate of the probability

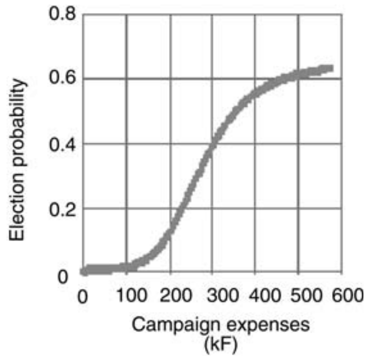


Fig. 1.37. Neural estimate of the election probability as a function of the total campaign expenses (data for the 1993 elections)

of being elected. Figure 1.37 shows the probability of election as a function of the total expense.

That application, although in the area of classification, is somewhat different from the previous two applications: in the latter, the classifier was intended to assign an existing pattern to a class, while, with high probability, the actual class of the unknown pattern would never be known with absolute certainty. In the present application, the situation is different, since the class of each pattern (candidate running for election) will be known unambiguously immediately after the election. This application falls in the class of forecasting by simulation: in order to optimize the probability of success, the candidate can estimate his success probability as a function of the strategy of expenses that he uses, and derive from those results the strategy that is most suitable to his situation.

In the next chapters, some sections will be devoted to forecasting by simulation: it will be shown that it is an area of excellence of neural networks.

1.4.5 An Application in Data Mining: Information Filtering

The rapid increase of the volume of available information, especially by electronic means, makes it mandatory to design and implement efficient information filtering tools, which allow the user to access relevant information only. Since such tools will address the needs of professionals, they must be reliable and user-friendly. The user can access relevant information either by being provided with full texts by the machine (text search), or by being provided text excerpts or answers to questions (information extraction).

Text categorization, also known as filtering, consists in finding, in a text corpus (e.g., of press releases, or of Web pages), the texts that are relevant to a predefined topic. The user can thus be provided with information that is important for his professional duties. In a machine-learning based system, the user does not express his topic of interest through a query, but by providing a set of relevant documents that define a topic or category. For a given topic, text categorization therefore consists in solving a two-class discrimination problem, which can be solved by neural networks, support vector machines (Chap. 6) or hidden Markov models (Chap. 4).

Text categorization is a very difficult problem, which goes much beyond text search by keywords, because a text may be relevant to a topic even though it contains none of the keywords that define the topic, or, conversely, a text may be irrelevant although it contains some or even all keywords.

The present application (from [Stricker 2000]) was developed by the French bank Caisse des dépôts et consignations, which provides an Intranet service for filtering press releases of Agence France Presse (AFP) in real time. The objective of the application is twofold:

- to develop an application that allows the user to create automatically an information filter on any topic of interest to him, under the condition that he provides examples of texts that are relevant to his topic of interest;
- to develop a machine-learning based tool that monitors the obsolescence of classical, rule-based information filters.

In the latter development, a neural-based filter is designed on the same topic as the rule-based filter. Since the neural network does not generate a binary response, but estimates a relevance probability, the largest discrepancies between the two filters can be analyzed and possibly be traced to vocabulary obsolescence: documents that are rated as relevant by the rule-based method, but whose relevance probability, estimated by the neural network, is very low, and documents that are rated as irrelevant by the rule-based filter and having an estimated relevance probability close to one as estimated by the neural filter [Wolinski 2000].

The former development consists in designing and implementing an automatic filter production system, whose major feature is the fact that it does not require any assistance from an expert, as opposed to rule-based filters. Therefore, a two-class discrimination system must be designed, from a database of texts that are labeled as relevant or irrelevant, that requires

- finding a representation of texts by real numbers, which should be as compact as possible,
- designing and implementing a classifier that uses that representation.

Thus, the problem of text representation, hence of input selection, is crucial for that application.

1.4.5.1 Input Selection

The most popular approach to text representation is the bag-of-words representation, whereby a text is represented by a vector, each component of which is a number that is a function of the presence or absence of the word in the text, or of its frequency in the text. Clearly, the main difficulty is the dimension of that vector, which is, in principle, equal to the number of words in the vocabulary. Nevertheless, all words are not equally discriminant: most frequent words (of, the, and) are not useful for discrimination, nor are very rare words. Therefore, the first step of the design of a filter is the determination of the vocabulary that is specific to the topic.

Word Encoding

The words are encoded in the following way: we denote by $FT(m,t)$ the frequency of occurrence of word m in text t, and by $FT(t)$ the average frequency of the terms in text t. Then the word m is described by [Singhal 1996]

$$x(m) = \frac{1 + \log(FT(m,t))}{1 + \log(FT(t))}.$$

Zipf's Law

Zipf's law [Zipf 1949] is helpful for finding discriminant words: given a corpus of T texts, we denote by $FC(m)$ the frequency of occurrence of word m in corpus T. A list of words, ranked in order of decreasing values of $FC(m)$, is built; we denote the rank of word m in that list by $r(m)$. Zipf's law states that $FC(m)r(m) = K$, where K is a corpus-dependent quantity. Hence, there is a very small number of very frequent words, and a large number of very rare words that occur once or twice in the corpus; between those extremes, there is a set of words in which discriminant words ought to be sought.

Extraction of the Specific Vocabulary

In order to extract the vocabulary that is specific to the topic, the ratio $R(m,t) = FT(m,t)/FC(m)$ is computed for each word m of each relevant text t. The words of the text are ranked in order of decreasing values of $R(m,t)$, the second half of the list is deleted, and a boolean vector $\boldsymbol{v}(t)$ is defined, such that $v_i(t) = 1$ if word i is present in the list, 0 otherwise. Finally, the vector $\boldsymbol{v} = \sum_t \boldsymbol{v}(t)$, is computed, where the summation is performed on all relevant documents: the specific vocabulary of the topic is the set of words that have a nonzero component in vector $\boldsymbol{v}$. Figure 1.38 shows that Zipf's law is obeyed on the corpus of Reuters releases, and that the words of the vocabulary specific to the topic *Falkland petroleum exploration* are indeed located in the middle of the distribution.

Final Selection

Within the specific vocabulary thus defined, which may be still large (one to several hundred words), a final selection is performed by the probe feature

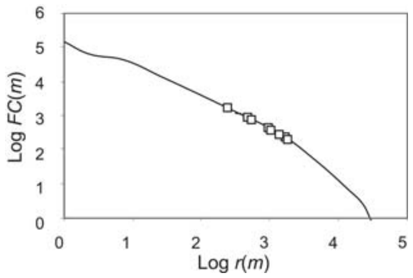


Fig. 1.38. An experimental verification of Zipf's law on the Reuters corpus, and location of the words of the vocabulary specific to the topic "Falkland petroleum exploration"

method, described in Chap. 2. After completion of that step, it turns out that, on the average over 500 different topics, the size of the specific vocabulary of a given topic is 25 words, which is a reasonable dimension for the input vector of a neural network. That representation, however, is not fully satisfactory yet. Since isolated words are ambiguous in such an application, the context must be taken into account.

1.4.5.2 Context Determination

In order to take into account the context in the representation of the texts, context words are sought in a window of 5 words on both sides of each word of the specific vocabulary.

- The words that are in the vicinity of the words of the specific vocabulary, in relevant texts, are defined as positive context words.
- The words that are in the vicinity of the words of the specific vocabulary, in irrelevant texts, are defined as negative context words.

In order to select the context words, the procedure that is used is identical to the selection procedure for the specific vocabulary. On the average over 500 topics, a topic is defined by 25 specific words, each of which having 3 context words.

1.4.5.3 Filter Design and Training

Filters Without Context

If the context is not taken into account, the inputs of the filter are the words of the specific vocabulary, encoded as indicated above. In accordance with the classifier design methodology described above, the structure of the classifier depends on the complexity of the discrimination problem. On the corpuses tested in the course of the development of the present application, the examples are linearly separable, so that networks made of a single neuron with sigmoid activation function solve the problem.

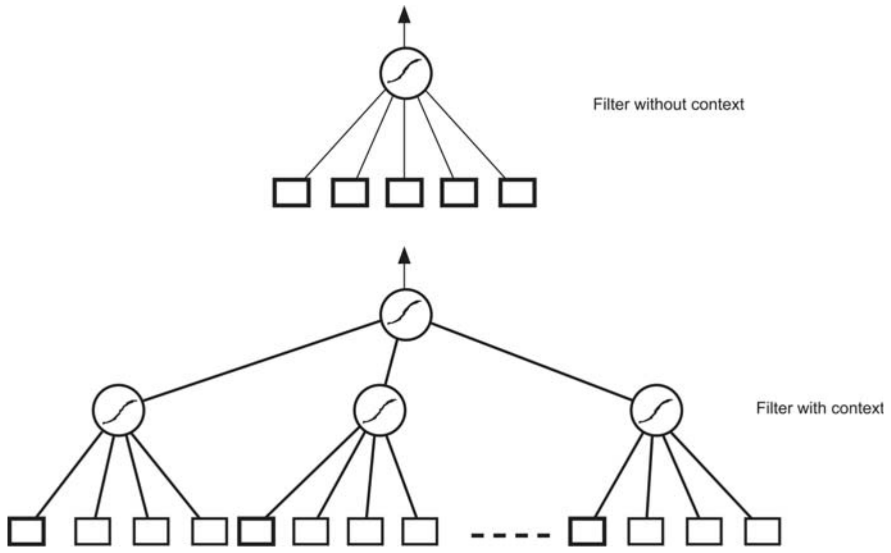


Fig. 1.39. A filter without context is a linear classifier whose inputs are the features that encode each word of the specific vocabulary (*rectangles in thick lines*); in a filter with context, the inputs are the features that encode the words of the specific vocabulary (*boxes in thick lines*), and, in addition, the features that encode the context words (*boxes in thin lines*)

Filters with Context

The context must have an influence of the feature that encodes each word of the specific vocabulary. Therefore, the filter represents each word of the specific vocabulary by a neuron with sigmoid activation function, whose inputs are

- the feature that encodes the word of interest,
- the features that encode the context words of that word.

The outputs of those neurons are separated linearly by a neuron with sigmoid activation function. Figure 1.39 shows a filter with context and a filter without context.

The introduction of context words as inputs increases the number of parameters of the classifier. Typically, for a topic with 25 words of specific vocabulary, and 3 context words per word of the specific vocabulary, the filter has 151 parameters. Since some topics are described by a small number of relevant texts, the use of a regularization method is mandatory. The weight decay method (described in Chap. 2) proved useful in the present application; its effect and implementation are explained in Chap. 2, in the section devoted to regularization techniques.

1.4.5.4 Validation of the Method

The annual competition organized by the TREC (Text REtrieval Conference) conference is a reference in the area of automatic language processing. The methodology that has been described was used in the "routing" task of the TREC-9 competition. The routing competition consists in ranking a large number of texts, in order of decreasing relevance for a large number of topics. For the TREC-9 routing competition, two text corpuses were used, relevant to 63 and 500 topics respectively, totaling 294,000 documents. Clearly, the task cannot be accomplished manually or semiautomatically: a fully automated procedure must be implemented. The above approach won the competition, for both corpuses. Figure 1.40 shows the scores of the participants [Stricker 2001].

1.4.6 An Application in Bioengineering: Quantitative Structure-Relation Activity Prediction for Organic Molecules

The investigation of quantitative structure-activity relations (QSAR) of molecules is a rapidly growing field thanks to progress in molecular simulation. The objective of QSAR is the prediction of chemical properties of molecules from structural data that can be computed *ab initio*, without actually synthesizing the molecule; thus, costly organic syntheses, leading to molecules that turn out not to have the desired property, can be avoided [Hansch 1995]. That approach is especially useful in the field of bio-engineering, for the prediction of pharmacological properties of molecules and for computer-aided drug discovery. It is also extremely valuable for solving conceptually similar problems, such as the prediction of properties of complex materials from their formulation, the prediction of thermodynamic properties of mixtures, etc.

Why are neural networks useful in that context? If there exists a deterministic relation between some features of the molecule and the property that must be predicted, then QSAR is amenable to a regression problem, i.e., to the determination of that unknown relation, from examples. If that relation is nonlinear, then neural networks can be advantageous, as argued above.

A prerequisite for such an approach is the availability of databases for training and testing the model. Because of the industrial importance of the problem, many databases of existing molecules for such properties as the boiling point, water solubility, or water-octanol partition coefficients (known as "LogP") are available. The latter property is important in pharmacology, because it gives a quantitative assessment of the ability of the molecule to cross biological barriers in order to be active; similarly, in the field of environment, the value of LogP of pesticides contributes to assessing their impact on environment.

Once the availability of appropriate databases is guaranteed, the relevant features that should be the inputs of the model must be determined. In the

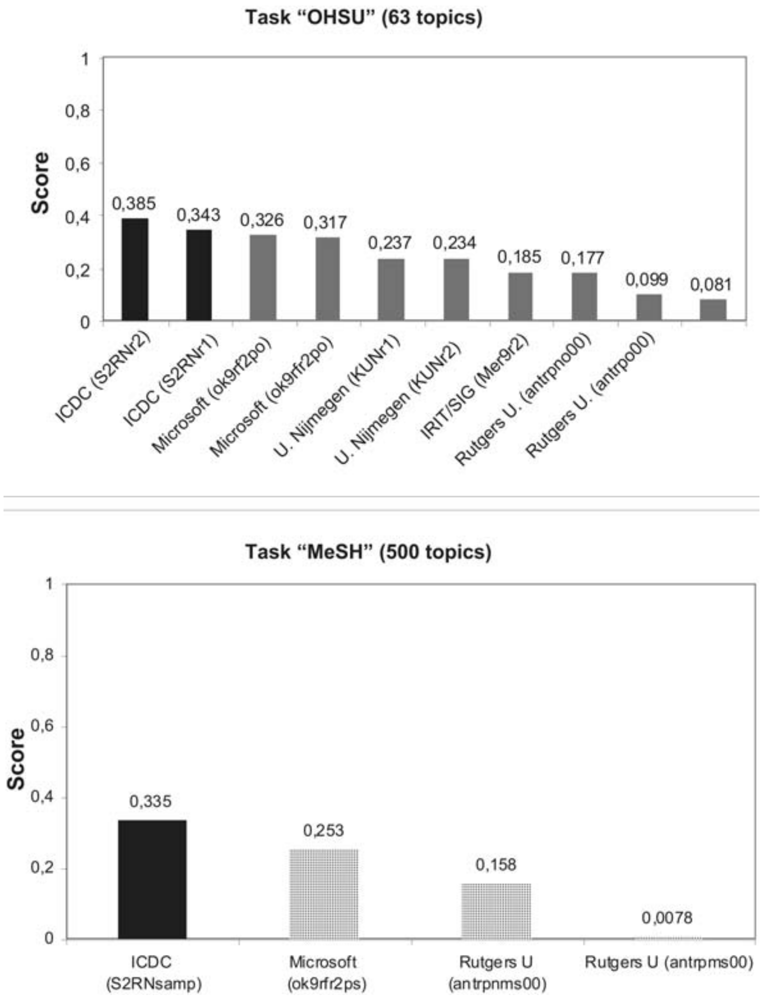


Fig. 1.40. Results of the TREC-90 routing task; *black*: results obtained by the above method; *grey*: results obtained by other methods

present case, the chemist's knowledge is of utmost importance. Classically, three categories of features are considered, i.e.,

- chemical features such as the molecular weight, the number of carbon atoms, etc.;
- geometrical features such as the volume of the molecule, its area, etc;
- electrical features such as the electric charges borne by each atom, dipole moments, etc.

For each property to be predicted, a set of candidate descriptors must be built, and selection techniques such as described in Chap. 2 must be applied. Because

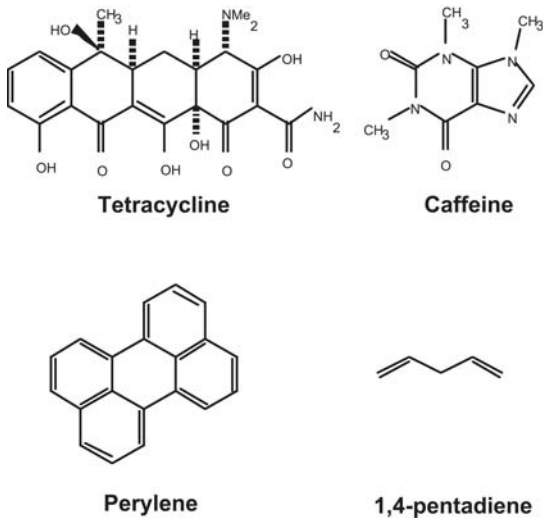


Fig. 1.41. Molecules that have chemical idiosyncrasies, whose properties may be poorly predicted by neural networks

of their parsimony, neural networks of very small size (5 to 7 hidden neurons) provide better results, on the same databases, than multilinear regression techniques that are used traditionally in the field [Duprat 1998].

Interestingly, the values of logP of some molecules were systematically either poorly learnt (when those molecules were in the training set) or poorly predicted (when present in the test set). In such a situation, one should first be suspicious of a measurement or typing-in error. If such is not the case, then one should conclude that the molecules have idiosyncrasies that are not shared by the other examples; in the present vase, it turns out that the molecules of interest are either strongly charged (tetracycline and caffeine, shown on Fig. 1.41), or, by contrast, interact very weakly with the solvent (perylene, 1-4 pentadiene, see Fig. 1.41). Thus, neural networks are able to detect anomalies; anomaly detection is actually one of the main areas of applications of neural networks.

1.4.7 An Application in Formulation: The Prediction of the Liquidus Temperatures of Industrial Glasses

In the same spirit as the previous application, albeit in a completely different field, thermodynamic parameters of materials can be predicted as a function of their formulation. Of specific interest is the prediction of the liquidus temperatures of oxide glasses. That temperature is the maximal temperature at which crystals are in thermodynamic equilibrium with the liquid; the prediction of the liquidus temperature is important for the glass industry, because the value of the viscosity at the liquidus temperature has a strong impact of the process of glass forming. Since the phase diagrams of glasses exhibit strong variations in the temperature domain of interest, many attempts at such predictions have been made (see for instance [Kim 1991]), and databases are available. Neural networks have been successfully used for the prediction of liquidus temperatures [Dreyfus 2003], especially (as expected) for glasses with more than three components.

Figure 1.42 illustrates, on the present industrial example, the parsimony of neural networks. It shows scatter plots, which are very convenient for assessing graphically the accuracy of the model: on the horizontal scale, the measured value of the quantity of interest is displayed, whereas the predicted values are displayed on the vertical scale. If prediction were perfect, all points should be aligned on the first bisector; actually, due to measurement inaccuracy and prediction errors, the points are more or less scattered; a good model should generate equivalent scatterings for the points of the training set and those of the validation or test set, and the vertical scattering should be on the order of the standard deviation of the noise. Clearly, such a tool is no substitute to the computation of the TMSE and VMSE as defined above, or of the leave-one-out score defined in Chap. 2, but it provides a quick means of comparing different models, for instance.

The model inputs are the contents of the glass in various oxides; the output is the estimated liquidus temperature. Figure 1.42(a) shows the results obtained on a silica glass (made of potassium oxide K_2O and aluminum oxide Al_2O_3, in addition to silicon oxide SiO_2, which is the main component), obtained with a network having 6 hidden neurons (25 parameters), and Fig. 1.42(b) shows the result obtained with a polynomial of degree 3, with a similar number of parameters (19). Clearly, with roughly the same number of parameters, the neural network performs much better. For comparison, Fig. 1.42(c) shows the scatter plot for a linear model.

1.4.8 An Application to the Modeling of an Industrial Process: The Modeling of Spot Welding

Spot welding is the most widely used welding process in the car industry: millions of such welds are made every day. The process is shown schematically on Fig. 1.13: two steel sheets are welded together by passing a very large current (tens of kiloamperes) between two electrodes pressed against the metal surfaces, typically for a hundred milliseconds. The heat thus produced melts a roughly cylindrical region of the metal sheets. After cooling, the diameter of the melted zone—typically 5 mm—characterizes the effectiveness of the process; a weld spot whose diameter is smaller than 4 mm is considered mechanically unreliable; therefore, the spot diameter is a crucial element in the safety of a vehicle. At present, no fast, nondestructive method exists for measuring the spot diameter, so that there is no way of assessing the quality of the weld immediately after welding. Therefore, a typical industrial strategy consists

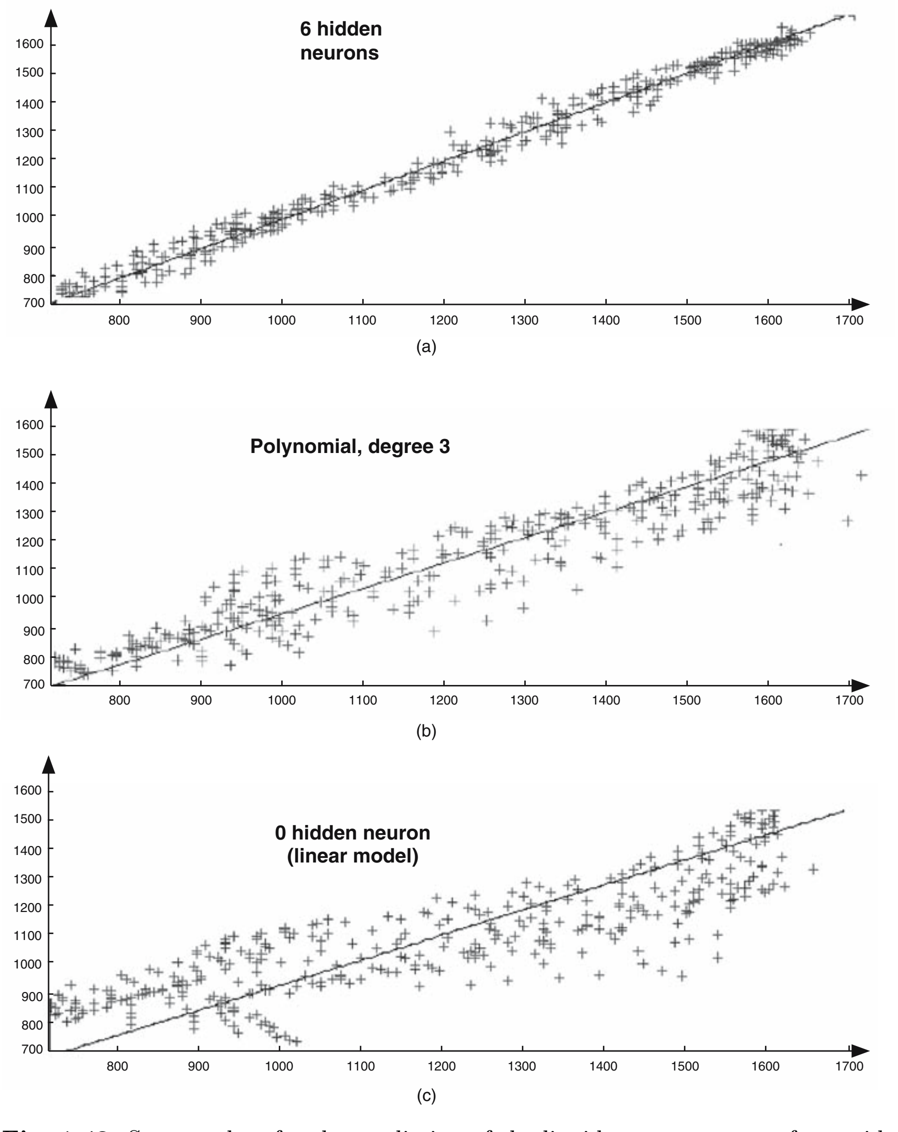


Fig. 1.42. Scatter plots for the prediction of the liquidus temperature of an oxide glass, as a function of its composition, for three different models

- in using an intensity that is much larger than actually necessary, which results in excessive heating, which in turn leads to the ejection of steel droplets from the welded zone (hence the sparks that can be observed during each welding by robots on car assembly chains);

- in making a much larger number of welds than necessary, just to be sure that a sufficient number of valid spots are produced.

Both the excessive current and the excessive number of spots result in a fast degradation of the electrodes, which must be changed or redressed frequently.

For all the above reasons, the modeling of the process, leading to a reliable on-line prediction of the weld diameter, is an important industrial challenge. Modeling the dynamics of the welding process from first principles is a very difficult task, for several reasons, including

- the computation time necessary for the integration of the partial differential equations of the knowledge-based model is many orders of magnitude larger than the duration of the process, which precludes real-time prediction of the spot diameter;
- many physical parameters appearing in the equations are not known reliably.

Those arguments lead to considering black-box modeling as an alternative. Since the process is nonlinear and has several input variables, neural networks are natural candidates for predicting the spot diameter from measurements performed during the process, immediately after weld formation, for on-line quality control [Monari 1999].

The main concerns for the modeling task are the choice of the model inputs, and the limited amount of examples available in the database, because gathering data is costly.

In [Monari 1999], the quantities that were candidates for input selection were mechanical and electrical signals that can be measured during the welding process. Input selection was performed by the techniques described in Chap. 2. The experts involved in the knowledge-based modeling of the process validated that set.

Because no simple nondestructive weld diameter measurement exists, the database is built by performing a number of welds in well-defined condition, and subsequently tearing them off; the melted zone, remaining on one of the two metal sheets, is measured. That is a lengthy and costly process, so that the initial training set was made of 250 examples only. Using experimental design through the confidence interval estimates described in Chap. 2, a training set extension strategy was defined in order to increase the database size. Half of the resulting data was used for training, and the other half for testing (the model selection method was virtual leave-one-out which, as explained in Chap. 2, does not require any validation set).

Figure 1.43 shows typical scatter plots, where each prediction is shown together with its confidence interval. The estimated generalization error, estimated from the virtual leave-one-out score defined in Chap. 2, was 0.27 mm, whereas the TMSE was 0.23 mm. Since those quantities are on the order of the measurement uncertainty, the results are satisfactory.

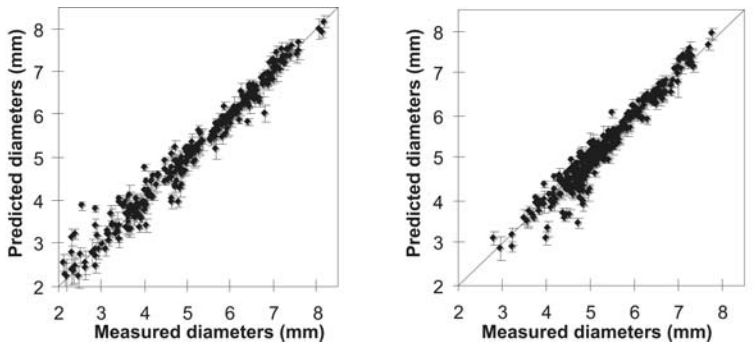


Fig. 1.43. Scatter plots for the prediction of the diameters of welding spots

1.4.9 An Application in Robotics: The Modeling of the Hydraulic Actuator of a Robot Arm

The previous applications involved feedforward neural networks only. We now turn to dynamic modeling, with recurrent neural networks.

We consider a hydraulic actuator that controls the position of a robot arm; therefore, the position of the arm depends on the hydraulic pressure in the actuator, which in turn depends of the angular position of a valve. A dynamic model of the relation between the hydraulic pressure and the opening of the valve was sought, in the framework of an informal competition between research groups involved in nonlinear modeling. A control sequence $\{u(k)\}$, i.e., a sequence of angles of the valve, and the corresponding sequence of the quantity to be modeled $\{y_p(k)\}$, i.e., the hydraulic pressure, are shown on Fig. 1.44. That sequence contains 1,024 samples, the first half of which, according to the rule of the game, was to be used as a training set and the second one as the validation set. Since no prior information was available from the physics of the process, a black-box model must be designed.

A cursory look at the data shows that a linear model of the process would certainly not be appropriate; the oscillations observed as responses to control variations that are almost steps suggests that the model is at least of second order. The training and validation sequences are approximately of the same type and same amplitude, but the amplitude of the control signal is larger in the validation set (around times 600 and 850) than in the training set. Thus, the conditions are not very satisfactory.

The example is analyzed in detail in Chap. 2. The best results [Oussar 1998] were obtained with a second-order state-space model, one state variable of which is the model output, of the form

$$\begin{aligned} y(k+1) = x_1(k+1) &= \boldsymbol{\psi}_1(x_1(k), x_2(k), u(k)) \\ x_2(k+1) &= \boldsymbol{\psi}_2(x_1(k), x_2(k), u(k)), \end{aligned}$$

with two hidden neurons. It is shown on Fig. 1.45.

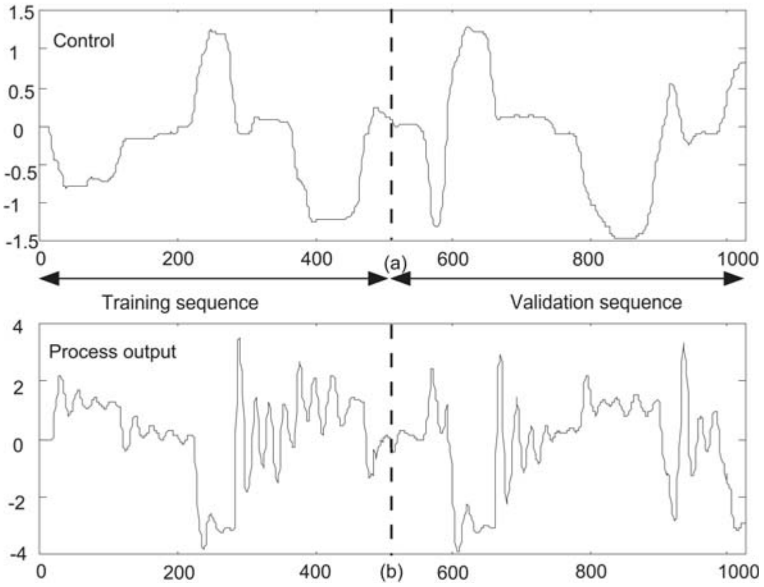


Fig. 1.44. Training and test sequences for a robot arm

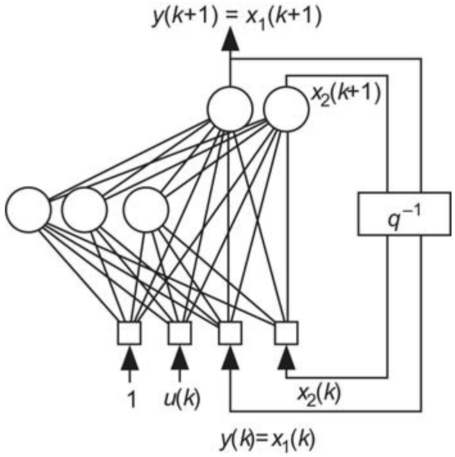


Fig. 1.45. State-space neural model of the hydraulic actuator. The output is one of the state variables

The mean square error obtained with that model is 0.07 on the training set and 0.12 on the validation set, which is very satisfactory given the available sequences. The modeling errors may be due to disturbances that are not measured, hence not present as inputs of the model. The results are shown on Fig. 1.46.

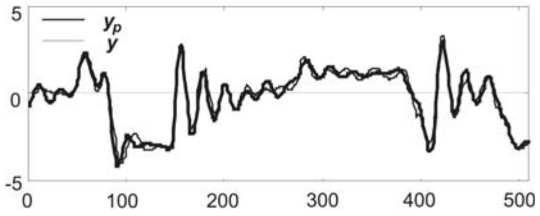


Fig. 1.46. State-space modeling of the hydraulic actuator

1.4.10 An Application of Semiphysical Modeling to a Manufacturing Process

As mentioned above, semi-physical modeling is a modeling methodology that allows the designer to make use both of prior knowledge resulting from a physical or chemical analysis of the process, and of available measurements. It is explained in detail in Chap. 2. In the present section, we describe its application to an industrial problem: the drying of the adhesive Scotch tape manufactured by 3M.

An adhesive tape is made of a plastic film—the substrate—coated with a liquid, which is passed in an oven, in a gas atmosphere in which the partial pressure of the solvent is much lower than the equilibrium pressure of the solvent at the oven temperature; therefore, the solvent evaporates, so that the solvent concentration at the surface becomes lower than the solvent concentration in the bulk. As a consequence, the solvent diffuses to the surface so as to compensate that concentration gradient, and evaporates at the surface. The process stops when the film is dried, so that the adhesive polymer alone stays on the substrate.

In a traditional process, organic solvents are used. However, for safety and environmental reasons, organic solvents are replaced by water. A very accurate model of drying in the presence of an organic solvent is available [Price 1997]; it is made of thirteen nonlinear, coupled algebraic and differential equations. When the organic solvent is replaced by water, some equations of the model are no longer valid, so that the whole model becomes inaccurate.

Polymers in aqueous solutions are not as well understood as polymers in organic solvents, so that no satisfactory physical model of the drying of water-based adhesives is available. However, sequences of measurements of sample weight as a function of time and oven temperature are available: the design of a semi-physical model is therefore possible and appropriate.

The equations of the model express the following phenomena:

- mass conservation in the bulk of the solvent: that equation is naturally still valid in the case of water-based adhesives;
- the diffusion of solvent towards the free surface (Flick's law); the validity of that equation is not arguable, but it involves a quantity (the diffusion

coefficient) whose variation as a function of concentration and temperature is given by the free-volume theory, whose validity can be disputed;
- mass conservation at the surface: any solvent molecule that reaches the surface, and evaporates, gives a contribution to the solvent partial pressure in the gas; that law remains valid;
- the boundary condition at the coating-substrate interface: since the substrate is impermeable to organic solvents and to water alike, that condition does not change;
- the value of the partial pressure of the solvent in the gas is the "driving force" of the whole process; it is given by an equation whose validity is not disputed.

Therefore, it turns out that the variation of the diffusion coefficient must be expressed by a black-box neural network, within the whole physical model. That has been done with the methodology that is described in detail in Chap. 2. Note that the equations of the model are not ordinary differential equations, but partial differential equations; that does not preclude the application of the method.

The reader interested in the details of the model and in the results will find them in [Oussar 2001]. Another industrial application of semi-physical modeling—the automatic detection of faults in an industrial distillation column—can be found in [Ploix 1997]. It is worth mentioning that applications or semi-physical modeling are in routine use in a major French manufacturing company for the design of new materials and products.

1.4.11 Two Applications in Environment Control: Ozone Pollution and Urban Hydrology

The two applications that are described in the present section are related to the prevision of nonlinear phenomena in environmental science.

1.4.11.1 Prevision of Ozone Pollution Peaks

Ozone concentration measurements are more and more widespread, and elaborate knowledge-based models of atmospheric pollution become available, so that the prediction of ozone peaks becomes feasible. The present section reports an investigation that was carried out at ESPCI within a work group to which measurements related to industrial area of Lyon (France) were made available. The objective was to assess the efficiency of machine learning techniques for designing black-box models for the prediction of ozone pollution peaks in that area.

The available data set was made of hourly measurements of a reliable ozone sensor between 1995 and 1998. Data pertaining to years 1995 to 1997 were used for training, and data of 1998 for validation. The task was to predict, 24 hours ahead of time, whether pollution would excess the legal alert threshold (180 $\mu g/m^3$ at the time of the investigation).

Two different approaches could be considered:

- classification: assign the next day to one of the classes “polluted” (threshold will be exceeded) or “non polluted,” as a function of the data available at 2 pm GMT,
- prevision: predict the ozone concentration, 24 hours ahead.

Since the definition of the class “polluted” depends on the legal definition of the alert threshold, which may vary for administrative, political or economical reasons, it was deemed preferable by the ESPCI group to follow the second approach. A black-box model was designed, which performs the prediction of ozone concentration during the next 24 hours, from information available at 2 pm GMT.

The naive idea consisted in using a dynamic nonlinear model such as a recurrent neural network. However, it turned out that such an approach would not be appropriate, because the process is not time-invariant: the physico-chemical phenomena that determine ozone concentration depend on the time of the day. Therefore, a set of 24 cascaded neural networks was designed, each network specializing in the prediction of ozone concentration at a given hour of the day (Fig. 1.47): network $\#N$ predicts the concentration at 2 pm $+N$ GMT; for each network, the candidate inputs are:

- the predictions of the previous $N-1$ networks;
- the set of available data:
  1. the measurements of sensors of NO and NO_2 at 2 pm GMT,
  2. the temperature at 2 pm on day D,
  3. the maximal temperature measured on day D, and the maximal temperature predicted for day $D+1$ by the national weather forecast service,
  4. geopotentials on day D,
  5. the time series of ozone concentrations performed before 2 pm on day D.

For each network, input selection among the above candidate variables was performed with techniques described in Chap. 2. Thus, the inputs of a given network, specialized in a given time of the day, are the appropriate inputs for that time of the day only.

Clearly, that approach can be adapted to other data sets, and can integrate expert knowledge, in a semi-physical model, when it will be available.

The mean prediction error on the validation year (1998) is $23\,\mu g/m^3$. Figure 1.48 illustrates the difficulty of the problem: despite a very accurate prediction during 20 hours, that day appears as a “false negative” since the measurement exceeds (by a very small amount) the alert threshold. Presumably, when such tools will be in routine operation (which was not yet the case when the present book was written), more subtle alert procedures than the simple threshold strategy will be implemented.

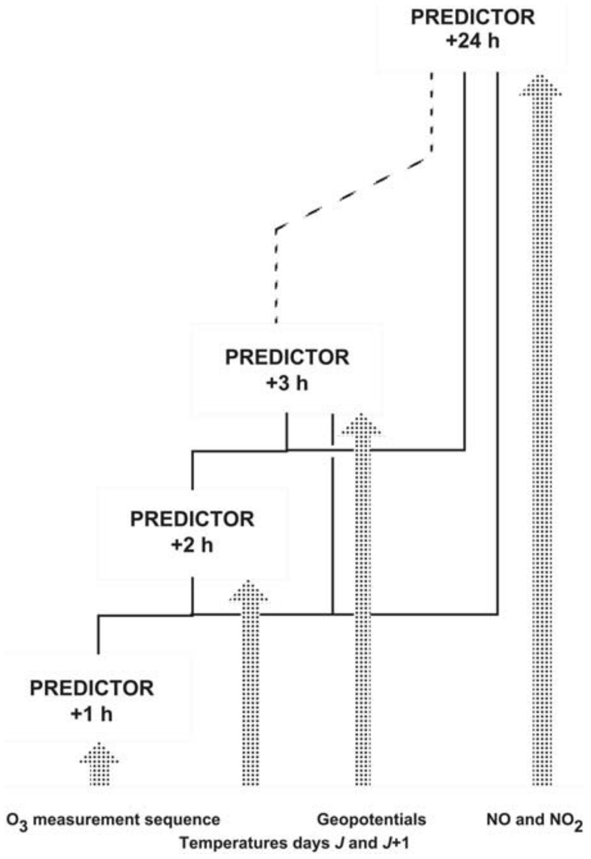


Fig. 1.47. The structure of a neural network for the prediction of ozone concentration, 24 hours ahead

1.4.11.2 Modeling the Rainfall-Water Height Relation in an Urban Catchment

The Direction de l'Eau et de l'Assainssement has developed a sophisticated system for measuring water heights in the sewers of an area of the suburbs of Paris, and has performed systematic measurements of rainfalls and of the corresponding water heights. The objective is to optimize the sewer network and to anticipate serious problems that are likely to arise in the case of severe rains. Hence the reliability of the water height sensors in the sewers is crucial for the reliability of the whole system: therefore, the automatic detection of faults in the water height sensors is mandatory [Roussel 2001].

Neural networks can be accurate models of nonlinear phenomena, which makes them useful tools for fault detection: if an accurate, real-time model of normal operation of a process is available, the observation of a statistically

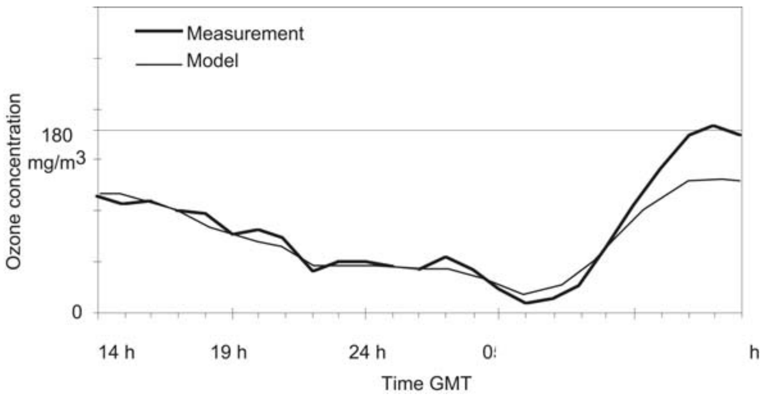


Fig. 1.48. Measured and predicted ozone concentrations on a day of 1998 (false negative)

significant discrepancy between the predictions of the model and the results of measurements results from a fault, such as, in the present example, a sensor failure.

Two kinds of faults can be present,

- stuck-at faults: the sensor outputs a constant value,
- drift: the sensor adds a slow drift to the real height value.

Both types of faults can be detected with recurrent neural networks, especially with NARMAX models (described in detail in Chaps. 2 and 4). Figure 1.49 displays the various behaviors of the modeling error, depending on whether the sensor is in normal operation or in drift failure mode.

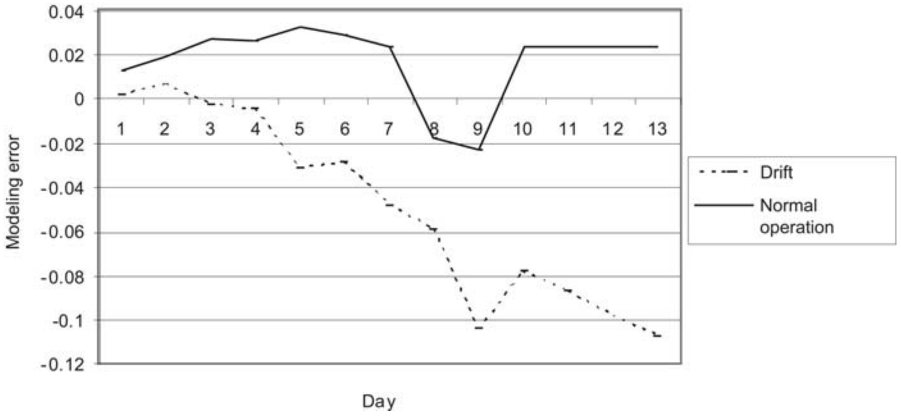


Fig. 1.49. Sensor fault detection in a sewer system

1.4.12 An Application in Mobile Robotics

Process control is the science that determines the control actions to which a process must be submitted in order to guarantee that it will operate in a prescribed fashion, in spite of unmeasured and unpredictable disturbances.

As an example of the role that neural networks can play in mobile robotics, we describe the automatic control of a 4WD Mercedes vehicle, which was equipped by the French company SAGEM with sensors and actuators that are necessary for making the vehicle autonomous. Controlling the vehicle consists in sending the appropriate signals to the actuators of the steering wheel, the throttle, and the brakes, in order to keep the vehicle on a prescribed trajectory with a prescribed velocity profile, in spite of disturbances such as wind gusts, sliding, slopes, etc.

Neural networks are good candidates as ingredients in nonlinear process control systems. They can implement any (sufficiently regular) nonlinear function. As a result, they can be useful in two different ways,

- as models of the process, since the design of a control system generally requires the availability of a model; neural networks are particularly useful in internal model control, as described in Chap. 5;
- as controllers, i.e., for computing the control signals (e.g., the angle by which the driving wheel must turn, and the velocity at which it should turn) from the setpoint (e.g., the desired heading of the vehicle).

The vehicle that was controlled was a 4 wheel-drive vehicle equipped with actuators (electric motor for actuating the driving wheel, hydraulic actuator for brakes, electric motor for actuating the throttle) and two categories of sensors,

- sensors that measure the state of the vehicle (proprioceptive sensors): odometers on the wheels, angular sensors for the driving wheel and for the throttle, pressure sensor in the brake system;
- sensors that measure the position of the vehicle with respect to the universe (exteroceptive sensors): an inertial platform.

The navigation and piloting system is made of the following elements:

- a planning module, which determines the desired trajectory of the vehicle, and its velocity profile, given the start and arrival points and the existing roads;
- a guiding module, which computes the heading and speed setpoint sequences;
- a piloting module, which computes the desired positions of the actuators;
- a control module of the actuators.

In that structure, neural networks are present at the level of the piloting module, where they compute the desired position sequences of the actuators as a function of the heading and speed setpoint sequences [Rivals 1994, 1995].

The application requested the design and implementation of two control systems that must fulfill two tasks,

- the control of the driving wheel, in order to keep the vehicle on the desired trajectory: a neural controller was designed, that performs a maximum lateral error of 40 cm, for curvatures up to 0.1 m^{-1}, and lateral slopes up to 30% in rough terrain; some elements of that controller used semi-physical modeling;
- the control of the throttle and the brake, in order to comply with the desired velocity profile.

All neural networks implemented within that application, whether models or controllers, are very parsimonious (less than ten hidden neurons). Their implementation on board did not require any special-purpose hardware: they were implemented as software on a standard microprocessor board that was also used for other purposes.

1.5 Conclusion

In the present chapter, we endeavored to explain why, and for what purposes, neural networks can be advantageously used. Some typical applications were presented (others are described in various chapters), so that model designers can get an intuition of what they can expect from that technique.

Before proceeding to more mathematical topics, it may be useful to emphasize the main points that should always be kept in mind when designing neural networks, i.e.,

- Neural networks are machine learning tools, that allow to fit very general nonlinear functions to sets of experimental data; just as for any statistical method, the availability of appropriate data is mandatory.
- Neural networks with supervised learning are parsimonious approximators, that can serve as static models (feedforward neural networks) or as dynamic models (recurrent neural networks).
- Neural networks with supervised learning can be high-quality classifiers, whose performances can reach those of the theoretical Bayes classifier; however, in the framework of classification for pattern recognition, the representation of the patterns to be recognized is often crucial for the performance of the whole system; in that context, neural networks with unsupervised learning may provide very valuable information for designing an efficient data representation.
- it is generally desirable, and often possible, to take advantage of all existing mathematical knowledge on the process to be modeled or patterns to be classified: neural networks are not necessarily black boxes.

The next chapters provide the mathematical background and the algorithmic information that are necessary for an efficient design of neural network models.

The foreword and the reading guide will help the reader to navigate through the chapters as a function of his own topics of interest.

1.6 Additional Material

1.6.1 Some Usual Neurons

Two categories of neurons can be distinguished, depending on the role of their parameters.

1.6.1.1 Neurons with Parameterized Inputs

The most popular neurons are neurons with parameterized inputs, in which one parameter is assigned to each input. The output of a neuron having n inputs $\{x_i\}$, $i = 0$ to $n-1$, is therefore given by an equation of the form $y = f\{x_i, w_i\}$, $i = 0$ to $n-1$, where $\{w_i\}$, $i = 0$ to $n-1$ are the parameters of the model.

In most cases, function f is the composition of two operations,

- the computation of the potential v of the neuron, which is the sum of the inputs of the neuron, weighted by the corresponding parameters,

$$v = \sum_{i=0}^{n-1} w_i x_i;$$

- the computation of a nonlinear function of the potential, termed activation function; that function is generally s-shaped, hence the generic name of sigmoid; preferably the activation function is symmetric with respect to the origin, such as the tanh function or the inverse tangent function, except if some prior knowledge on the problem prompts the implementation of different, more appropriate functions.

The set of inputs of the neuron generally includes a specific input, termed bias, the value of which is constant, equal to 1. It is usually assigned the index 0, so that the potential is of the form

$$v = w_0 + \sum_{j=1}^{n-1} w_j x_j .$$

Thus, the expression of the output of the neuron is: $y = f[w_0 + \sum_{j=1}^{n-1} w_j x_j]$.

Figure 1.50 shows the output of a neuron with three inputs ($x_0 = 1$, x_1, x_2) with parameters $w_0 = 0$, $w_1 = 1$, $w_2 = -1$.

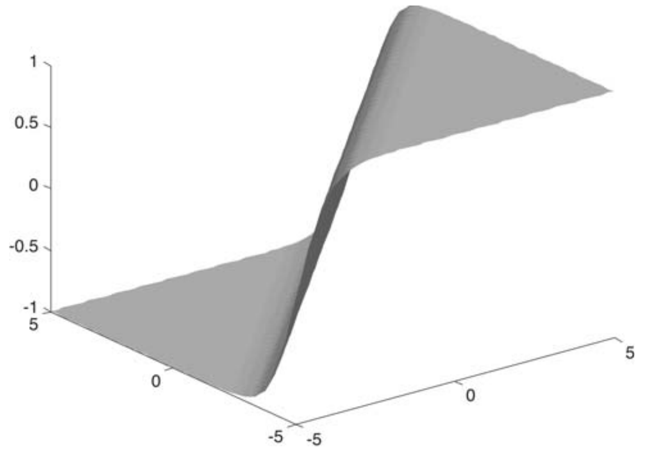


Fig. 1.50. Output of a neuron with 3 inputs $\{x_0 = 1, x_1, x_2\}$ with weights $\{w_0 = 0, w_1 = +1, w_2 = -1\}$, whose activation function is a tanh function: $y = \tanh(x_1 - x_2)$

Two variants of that type of neuron are

- high-order neural networks, whose potential is not an affine function of the inputs, but a polynomial function; they are the ancestors of the support vector machines (or SVM) used essentially for classification, described in Chap. 6;
- MacCulloch-Pitts neurons, or perceptrons, which are the ancestors of present-day neurons; Chap. 6 describes in detail their use for discrimination.

1.6.1.2 Neurons with Parameterized Nonlinearities

The parameters of those neurons are assigned to their nonlinearity: they are present in function f. Thus, the latter may be a "radial basis function" (RBF) or a wavelet.

Example

Gaussian radial basis function,

$$y = \exp\left[\sum_{i=1}^{n}(x_i - w_i)^2 \Big/ 2w_{n+1}^2\right].$$

The parameters $\{w_i, i = 1 \text{ to } n\}$ are the coordinates of the center of the Gaussian in input space; parameter w_{n+1} is its standard deviation. Figure 1.51 shows an isotropic Gaussian RBF, centered at the origin, with standard deviation $1/\sqrt{2}$.

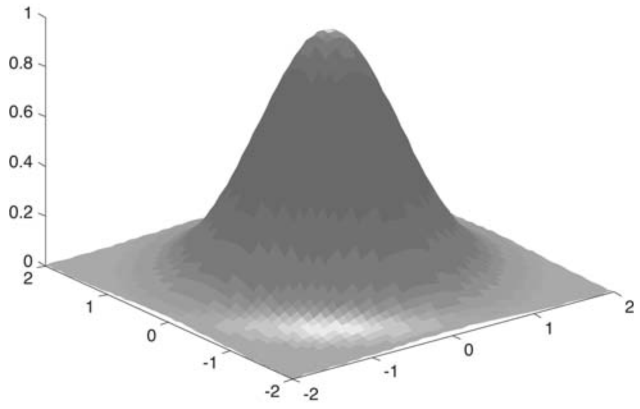


Fig. 1.51. Gaussian isotropic $RBF_y = \exp[-(x_1^2 + x_2^2)] : w_0 = w_1 = 0, w_3 = 1/\sqrt{2}$

The term radial basis function arises from approximation theory; they can be chosen so as to form a mathematical basis of functions. In regression, RBF's are generally not chosen so as to satisfy that requirement; however, following the current use, we will keep the term radial basis function.

1.6.2 The Ho and Kashyap Algorithm

The Ho and Kashyap algorithm finds, in a finite number of iterations, whether two given sets of observations are linearly separable in feature space. If they are, the algorithm provides a solution (among an infinity of possible solutions), which is not optimized (as opposed to algorithms that are explained in Chap. 6). Therefore, that algorithm is mainly used for finding out whether sets are linearly separable. If such is the case, it is advisable to use one of the optimized algorithms described in Chap. 6.

Consider two sets of examples, having n_A and n_B elements respectively, belonging to two classes A and B; if the examples are described by n features, each of them is described by an n-vector. We denote by $\boldsymbol{x}_k^A$ the vector that represents the k-th example of class A ($k = 1$ to n_A), and by $\boldsymbol{w}$ the vector of parameters of a linear separator; if such a separator exists, i.e., if the examples are linearly separable, then one has

$$\begin{aligned} \boldsymbol{x}_k^A \boldsymbol{w} &> 0 \quad \text{for all } k, \\ \boldsymbol{x}_k^B \boldsymbol{w} &< 0 \quad \text{for all } k. \end{aligned}$$

We define matrix M whose rows are the vectors that represent the examples of A and the opposites of the vectors representing the examples of B, i.e.,

$$M = [\boldsymbol{x}_1^a, \boldsymbol{x}_2^a, \ldots, \boldsymbol{x}_{n_a}^a, \boldsymbol{x}_1^b, \boldsymbol{x}_2^b, \ldots, \boldsymbol{x}_{nb}^b]^{\mathrm{T}},$$

where superscript $^\mathrm{T}$ denotes the transpose of a matrix. Then a linear separator exists if and only if there exists a vector $\boldsymbol{w}$ such that

$$M\boldsymbol{w} > 0,$$

or, equivalently, if there exists a vector $\boldsymbol{y} > 0$ and a vector $\boldsymbol{w}$ such that $M\boldsymbol{w} = \boldsymbol{y}$.

Then one has $\boldsymbol{w} = M^*\boldsymbol{y}$, where M^* is the pseudo-inverse of matrix M : $M^* = M^\mathrm{T}(MM^\mathrm{T})^{-1}$, which can be computed by the Choleski method [Press 1992].

The Ho and Kashyap algorithm is as follows:

```
Initialization (iteration 0):
$\boldsymbol{w}(0) = M^*\boldsymbol{y}(0)$ where $\boldsymbol{y}(0)$ is an arbitrary positive vector
Iteration $i$
$\alpha(i) = M^*\boldsymbol{w}(i) - \boldsymbol{y}(i)$
$\boldsymbol{y}(i+1) = \boldsymbol{y}(i) + \rho(\alpha(i) + |\alpha(i)|)$ where $\rho$ is a positive scalar smaller than
1 $\boldsymbol{w}(i+1) = \boldsymbol{w}(i) + \rho M^*(\alpha(i) + |\alpha(i)|)$
If one of the components of $\boldsymbol{y}(i) < 0$ then the examples are not linearly separable.
If all components of $M\boldsymbol{w}(i) > 0$ then the examples are linearly separable and $\boldsymbol{w}(i)$ is a solution.
```

The algorithm converges after a finite number of iterations.

References

1. Antoniadis A., Berruyer J., Carmona R. [1992], *Régression non linéaire et applications*, Economica
2. Barron A. [1993], Universal approximation bounds for superposition of a sigmoidal function, *IEEE Transactions on Information Theory*, 39, pp 930–945
3. Baum E.B., Wilczek F. [1988], Supervised learning of probability distributions by neural networks, *Neural Information Processing Systems*, pp 52–61
4. Benveniste A., Juditsky A., Delyon B., Zhang Q., Glorennec P.-Y. [1994], Wavelets in identification, *10th IFAC Symposium on Identification*, Copenhague
5. Bishop C. [1995], *Neural networks for pattern recognition*, Oxford University Press
6. Bridle J.S. [1990], Probabilistic interpretation of feedforward classification network outputs, with relationship to statistical pattern recognition, *Neurocomputing: algorithms, architectures and applications*, pp 227–236 Springer
7. Broomhead D.S., Lowe D. [1988], Multivariable functional interpolation and adaptive networks, *Complex Systems*, 2, pp 321–355

8. Cover T.M. [1965], Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition, *IEEE Transactions on Electronic Computers*, 14, pp 326–334
9. Draper N.R., Smith H. [1998], *Applied regression analysis*, John Wiley & Sons
10. Duprat A., Huynh T., Dreyfus G. [1998], Towards a principled methodology for neural network design and performance evaluation in QSAR; application to the prediction of LogP, *Journal of Chemical Information and Computer Sciences*, 38, pp 586–594
11. Dreyfus C., Dreyfus G. [2003], A machine-learning approach to the estimation of the liquidus temperature of glass-forming oxide blends, *Journal of Non-Crystalline Solids*, 318, pp 63–78
12. Hampshire J.B., Pearlmutter B. [1990], Equivalence proofs for multilayer perceptron classifiers and the Bayesian discriminant function, *Proceedings of the 1990 connectionist models summer school*, pp 159–172, Morgan Kaufmann
13. Hansch C., Leo A. [1995], *Exploring QSAR, Fundamentals and applications in chemistry and biology*; American Chemical Society
14. Ho E., Kashyap R.L. [1965], An algorithm for linear inequalities and its applications, *IEEE Transactions on Electronic Computers*, 14, pp 683–688
15. Hopfield J.J. [1987], Learning algorithms and probability distributions in feed-forward and feedback neural networks, *Proceedings of the National Academy of Sciences*, 84, pp 8429–433
16. Hornik K., Stinchcombe M., White H. [1989], Multilayer feedforward networks are universal approximators, *Neural Networks*, 2, pp 359–366
17. Hornik K., Stinchcombe M., White H. [1990], Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks, *Neural Networks*, 3, pp 551–560
18. Hornik K. [1991], Approximation capabilities of multilayer feedforward networks, *Neural Networks*, 4, pp 251–257
19. Kim S.S., Sanders T.H. Jr. [1991], Thermodynamic modeling of phase diagrams in binary alkali silicate systems, *Journal of the American Ceramics Society*, 74, pp 1833–1840
20. Knerr S., Personnaz L., Dreyfus G. [1990], Single-layer learning revisited: a stepwise procedure for building and training a neural network, *Neurocomputing: algorithms, architectures and applications*, pp 41–50, Springer
21. Knerr S. [1991], *Un méthode nouvelle de création automatique de réseaux de neurones pour la classification de données: application à la reconnaissance de chiffres manuscrits*, Thèse de Doctorat de l'Université Pierre et Marie Curie, Paris
22. Knerr S., Personnaz L., Dreyfus G. [1992], Handwritten digit recognition by neural networks with Single-layer Training, *IEEE Transactions on Neural Networks*, 3, pp 962–968
23. LeCun Y., Boser B., Denker J.S., Henderson D., Howard R.E., Hubbard W., Jackel L.D. [1989], Backpropagation applied to handwritten zip code recognition, *Neural Computation*, 1, pp 541–551
24. Mallat S. [1989], A theory for multiresolution signal decomposition: the wavelet transform, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11, pp 674–693
25. McCulloch W.S., Pitts W. [1943], A logical calculus of the ideas immanent in nervous activity, *Bulletin of Mathematical Biophysics*, 5, pp 115–133

26. Marcos S., Macchi O., Vignat C., Dreyfus G., Personnaz L., Roussel-Ragot P. [1992], A unified framework for gradient algorithms used for filter adaptation and neural network training, *International Journal of Circuit Theory and Applications*, 20, pp 159–200
27. Minsky M., Papert S. [1969] *Perceptrons.* MIT Press
28. Monari G. [1999], *Sélection de modèles non linéaires par leave-one-out; étude théorique et application des réseaux de neurones au procédé de soudage par points*, Thèse de Doctorat de l'Université Pierre et Marie Curie, Paris. Disponible sur le site *http://www.neurones.espci.fr.*
29. Moody J., Darken C.J. [1989], Fast learning in networks of locally-tuned processing units, *Neural Computation*, 1, pp 281–294
30. Nerrand O., Roussel-Ragot P., Personnaz L., Dreyfus G., Marcos S. [1993], Neural networks and nonlinear adaptive filtering: unifying concepts and new Algorithms, *Neural Computation*, 5, pp 165–197
31. Oukhellou L., Aknin P. [1997], Modified Fourier Descriptors: A new parametrization of eddy current signatures applied to the rail defect classification, *III International workshop on advances in signal processing for non destructive evaluation of materials*
32. Oukhellou L., Aknin P., Stoppiglia H., Dreyfus G. [1998], A new decision criterion for feature selection: application to the classification of non destructive testing signatures, *European Signal Processing Conference* (EUSIPCO'98)
33. Oussar Y. [1998], *Réseaux d'ondelettes et réseaux de neurones pour la modélisation statique et dynamique de processus*, Thèse de Doctorat de l'Université Pierre et Marie Curie, Paris. Disponible sur le site *http://www.neurones.espci.fr*
34. Oussar Y., Dreyfus G. [2000], Initialization by selection for wavelet network training, *Neurocomputing*, 34, pp 131–143
35. Oussar Y., Dreyfus G. [2001], How to be a gray box: dynamic semi-physical modeling, *Neural Networks*, vol. 14, pp 1161–1172
36. Ploix J.L., Dreyfus G. [1997], Early fault detection in a distillation column: an industrial application of knowledge-based neural modelling, *Neural Networks: Best Practice in Europe*, pp 21–31, World Scientific
37. Powell M.J.D. [1987], Radial basis functions for multivariable interpolation: a review, *Algorithms for approximation*, pp 143–167
38. 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
39. Price D., Knerr S., Personnaz L., Dreyfus G. [1994], Pairwise neural network classifiers with probabilistic outputs, *Neural Information Processing Systems*, 7, pp 1109–1116, Morgan Kaufmann
40. Price P.E., Wang S., Romdhane I.H. [1997], Extracting effective diffusion parameters from drying experiments. *AIChE Journal*, 43, pp 1925–1934
41. Rivals I., Canas D., Personnaz L., Dreyfus G. [1994], Modeling and control of mobile robots and intelligent vehicles by neural networks, *Proceedings of the IEEE Conference on Intelligent Vehicles*, pp 137–142
42. Rivals I. [1995], *Modélisation et commande de processus par réseaux de neurones: application au pilotage d'un véhicule autonome*, Thèse de Doctorat de l'Université Pierre et Marie Curie, Paris. Disponible sur le site *http://www.neurones.espci.fr.*
43. Roussel P., Moncet F., Barrieu B., Viola A. [2001], Modélisation d'un processus dynamique à l'aide de réseaux de neurones bouclés. Application à la

modélisation de la relation pluie-hauteur d'eau dans un réseau d'assainissement et à la détection de défaillances de capteurs, *Innovative technologies in urban drainage*, 1, 919–926, G.R.A.I.E

44. Seber G.A.F., Wild C.J. [1989], *Non-linear regression*, Wiley Series in Probability and Mathematical Statistics, John Wiley & Sons
45. Singhal A. [1996], Pivoted length normalization. *Proceedings of the 19th Annual International Conference on Research and Development in Information Retrieval (SIGIR'96)*, pp 21–29
46. Stoppiglia H. [1997], *Méthodes statistiques de sélection de modèles neuronaux; applications financières et bancaires*, Thèse de Doctorat de l'Université Pierre et Marie Curie, Paris. Disponible sur le site *http://www.neurones.espci.fr.*
47. Stricker M. [2000], *Réseaux de neurones pour le traitement automatique du langage: conception et réalisation de filtres d'informations*, Thèse de Doctorat de l'Université Pierre et Marie Curie, Paris. Disponible sur le site *http://www.neurones.espci.fr.*
48. Stricker M., Vichot F., Dreyfus G., Wolinski F. [2001], Training context-sensitive neural networks with few relevant examples for the TREC-9 routing, *Proceedings of the TREC-9 Conference.*
49. Vapnik V. [1995], *The nature of statistical learning theory*, Springer
50. Wolinski F., Vichot F., Stricker M. [2000], Using learning-based filters to detect rule-based filtering Obsolescence, *Conférence sur la Recherche d'Information Assistée par Ordinateur, RIAO'2000*, Paris
51. Zipf G. K. [1949], *Human Behavior and the Principle of Least Effort.* Addison-Wesley.