UNIVERSITÁ DEGLI STUDI DI PADOVA

DEPARTMENT OF INFORMATION ENGINEERING
UNIVERSITY OF PADOVA

FACOLTÁ DI INGEGNERIA
CORSO DI LAUREA IN INGEGNERIA DELL'INFORMAZIONE

Tesi di laurea Triennale

Neural networks for Medical decision

Reti neurali per la
diagnosi in medicina

Studente:
Francesco RIGOBELLO

Relatore:
Prof.ssa Gianna Maria TOFFOLO

Anno Accademico 2011/2012 - Padova, 27 Settembre 2012

Index

1. Introduction to neural networks

1.1 Introduction

Research on neural networks has been motivated right from its inception by the recognition of the fact that the brain computes in an entirely different way from the conventional digital calculator. The brain is a highly complex, nonlinear, and parallel computer information processing system, that has the capability of organizing neurons so as to perform certain computation such as pattern recognition, perception or motor control, many times faster than a conventional computer.

1.1.1 Biological model of a neuron

Let us introduce the biological model of a neuron

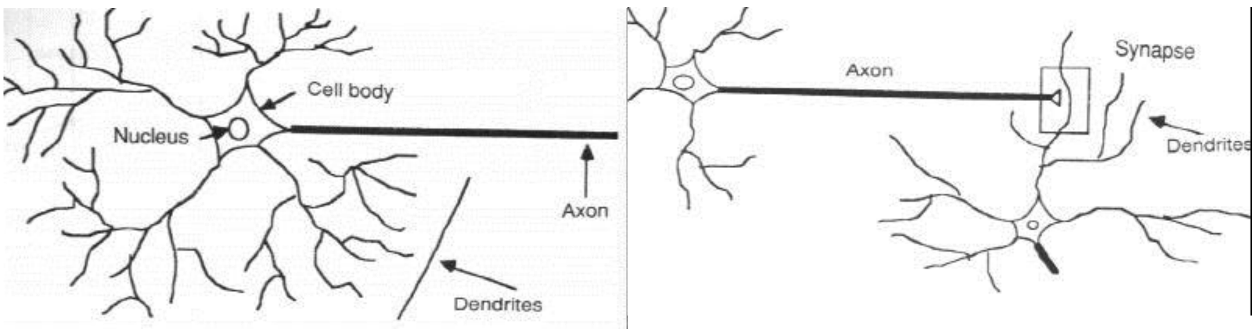


Fig. 1.1: *Dendrites and synapses*

The nervous cells, known ase neurons, are the building blocks of the nervous system.
A neuron is composed of a cell body and a large number of receptive zones, called dentrites, through which a neuron may receive electrical signals from other neurons.
Moreover, each neuron has a fine extension, called axon, used to transmitt electrical signals to other cells (e.g. to dentrites of other neurons).
The connection point between an axon terminal and a dentrite is called synapse. A synapse is a functional unit that mediates the interactions between neurons by enabling electrical trasmission through a chemical process in which neurotransmitter substances are liberated.
If the electrical signal that excitates a neuron is sufficiently large compared with its inhibitory input, the neuron "activates" and it is able to send a spike of electrical activity down its axon towards other cells.

A crucial property of the brain is its ability to learn, by modifying neurons connections according to experience aknowledged.
Moreover, the brain has not a centralized control; in fact, different parts work together affecting each other to realize a specific task.

All these brain features, suggested the evolution of models capable to emulate its behavior.

1.1.2 Artificial neural networks

An artificial neural network is a computational model that is inspired by the structure and functional aspects of biological neural networks.

To reproduce artificially such a biological model, we introduce a network architecture in which informations processing units, referred to as artificial neurons, are richly interconnected by weighted connection lines, known as synapses, which are structures that mediates interaction between neurons.
"Knowledge" is aquired by a network through a learning or training process in which connection strenghts, known as weights, are adjusted according to the input data examples and the relative outputs.
A well trained neural network is able to perform tasks such as predicting an output value, classyfying an object, approximating a function and recognizing or completing a known pattern, and thus constitutes a valid mathematical instrument in many fields to face problems that classical approach cannot solve.

1.1.3 Historical notes

The modern era of neural networks is said to have begun with the pioneering work of McCulloch and Pitts. McCulloch was a psychiatrist and neuroanatomist who had spent some 20 years thinking about the representation of an event in the nervous system; he was joined in 1942 by Pitt , a brilliant mathematician, and togheter, in 1943, they wrote a fundamental paper, in which a logical calculus of neural network was described.
Some 15 years after the publication of McCulloch and Pitt's classic paper, a new approach on the patter recognition problem was introduced by Rosenblatt in his work on the perceptron (1958). Roseblatt perceptron was a network with two layers of computational nodes and a single layer of interconnections. This model, however, was limited to the solution of linear problems whereas many problems in discrimination and analysis cannot be solved by a linear capability alone.
The capabilities of neural networks were expanded from linear to non linear domains in 1974 by Werbos. These multilayered perceptrons were trained via gradient descents methods, and the original algorithm became known as "back-error propagation".
The great computational potentialities of the multilayer perceptron were proved by Hornik, who showed how a network, with appropriate internal parameters, could approximate an arbitrary nonlinear function. Because classification tasks, prediction and decision support problems can be restated as function approximation problems, this discovery showed that neural networks have the potential for solving major problems in a wide range of application domains.
Neural networks are nowadays used in many different fields: from military application to medical image analysis, from financial industry to medical decision systems.

In this work, after a theoretical description of the neural networks models, we will focus on a diagnostical application.

1.2 Models of a neuron

A neuron is an information-processing unit that is fundamental to the operation of a neural network.

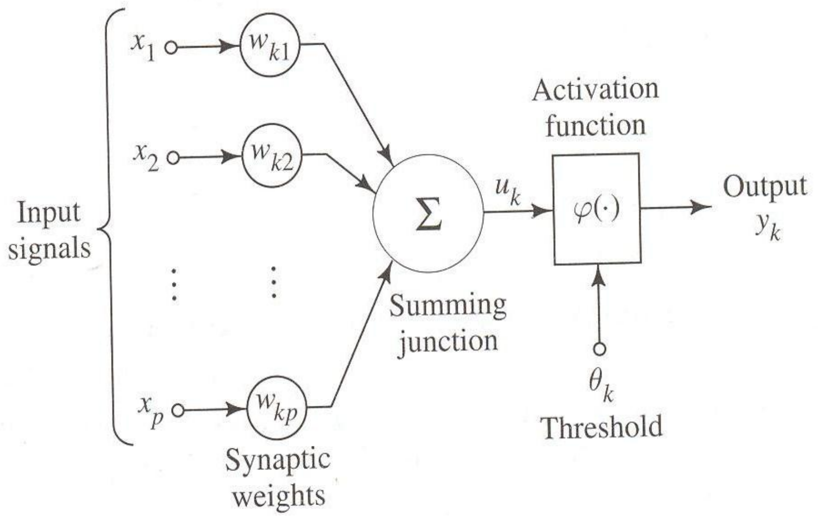


***Fig. 1.2**: model of a neuron*

We may indentify three basic elements of the neuron model:

1. A set of synapses or connecting links, each of which is characterized by a wieght or strenght of its own. Specifically, a signal x_j at the input of synapse *j* connected to neuron *k* is multiplied by the synaptic weight w_{kj}.
2. An adder for summing the input signals, weighted by the respective synapses of the neuron; the operations described here constitute a linear combiner.
3. An activation function for limiting the amplitude of the output of a neuron. We can also refer to the activation function as a squashing function in that it squashes (limits) the permissible amplitude range of the output signal to some finite value. Typically, the normalized amplitude range of the output of a neuron is written as the closed interval [0,1].

The model shown in figure also includes an externally applied threshold θ_k that has the effect of lowering the net input of the activation function.
In mathematical terms, we may describe a neuron k by writing the following pair of equations:

$$u_k = \sum_{j=1}^{p} w_{kj} x_j$$

and

$$y_k = \varphi(u_k - \theta_k)$$

where

$x_1, x_2, .., x_p$ are the input signals;
$w_{k1}, w_{k2}, .., w_{kp}$ are the synaptic weights of neuron k;

u_k is the linear combiner output;
θ_k is the threshold;
$\varphi(-)$ is the activation function;
y_k is the output signal of the neuron.

The use of threshold θ_k has the effect of applying an affine transformation to the output u_k of the linear combiner of the model, as shown by

$$v_k = u_k - \theta_k$$

In particular, depending on whether the threshold θ_k is positive or negative, the relationship between the effective internal activity level or activation potential v_k of neuron k and the linear combiner output u_k is modified in the manner illustrated in figure.

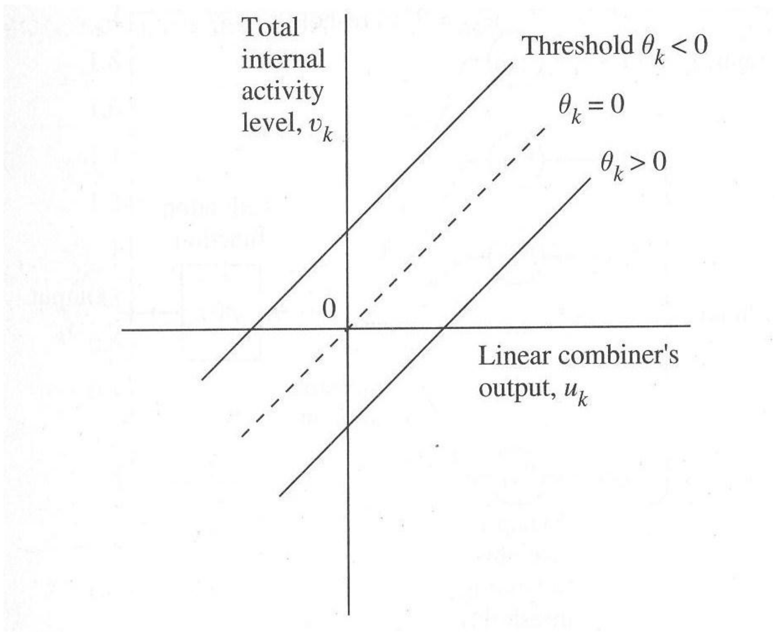

Total internal activity level, v_k
Threshold $\theta_k < 0$
$\theta_k = 0$
$\theta_k > 0$
0
Linear combiner's output, u_k

***Fig. 1.3**: Transformation produced by the presence of a threshold*

The threshold θ_k is an external parameter of artificial neuron k. We may formulate equivalently the combination of the previous equations as follows:

$$u_k = \sum_{j=0}^{p} w_{kj} x_j$$

and

$$y_k = \varphi(v_k)$$

In the last equation we have added a new synapse, whose input is

$$x_0 = -1$$

and whose weight is

$$w_{k0} = \theta_k$$

We may therefore reformulate the model of neuron k as in the following figure.

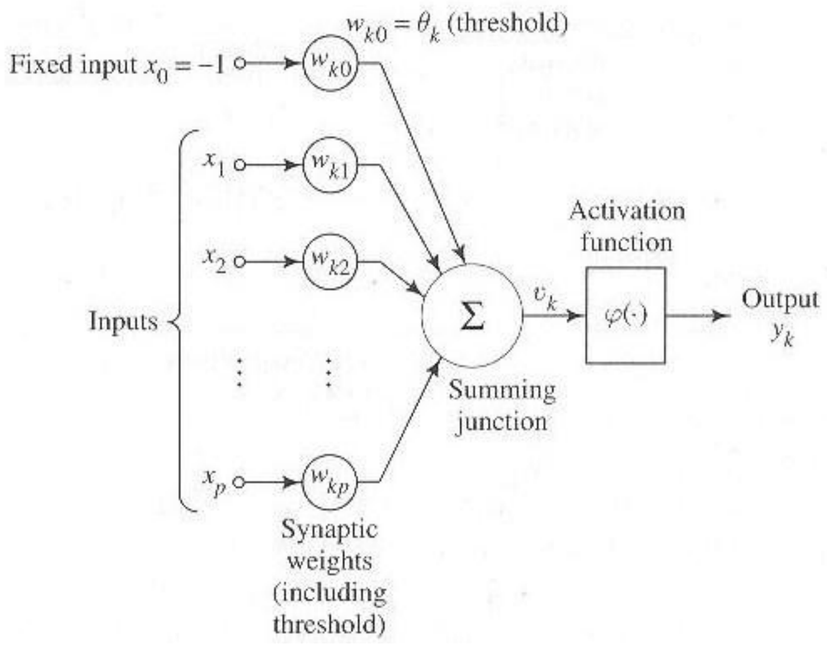


Fig. 1.4: *another model of a neuron*

The effect of the threshold is represented by doing two things:

- adding a new input signal fixed at -1,
- adding a new synaptic weight equal to the threshold ϑ_k.

1.2.1 Types of Activation Function

The activation function denoted by *φ(-)*, defines the output of a neuron in terms of the activity level at its input. We may identify three basic types of activation functions:

1) Threshold Function

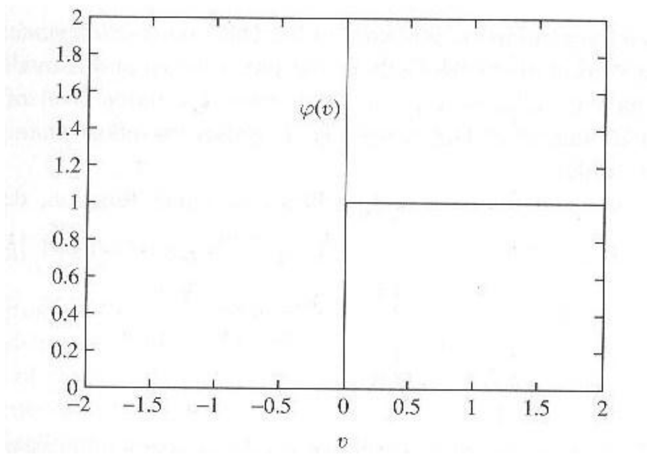


Fig. 1.5: *Threshold function*

We have

$$\varphi(v) = \begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{if } v < 0 \end{cases}$$

Correspondingly, the output of neuron k employing such a threshold function is expressed as

$$y_k = \begin{cases} 1 & \text{if } v_k \geq 0 \\ 0 & \text{if } v_k < 0 \end{cases}$$

where v_k is the internal activity level of the neuron; that is,

$$v_k = \sum_{j=1}^{p} w_{kj} x_j - \theta_k$$

Such a neuron is referred to in the literature as the McCulloch-Pitts model (1943). In this model, the output of a neuron takes on the value of 1 if the total internal activity level of that neuron is nonnegative and 0 otherwise (all-or-none property).

2) *Piecewise-Linear Function*

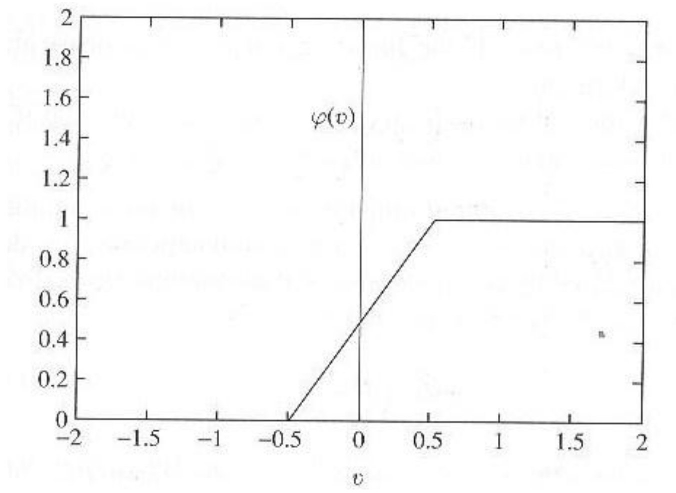


Fig. 1.6*: Piecewise-linear function*

We have

$$\varphi(v) = \begin{cases} 1, & v \geq \frac{1}{2} \\ v, & -\frac{1}{2} > v > -\frac{1}{2} \\ 0, & v \leq -\frac{1}{2} \end{cases}$$

where the amplification factor inside the linear region of operation is assumed to be unity. This form of an activation function may be viewed as an approximation to a nonlinear amplifier.

The following two situations may be viewed as special forms of the piecewise-linear function:

- A linear combiner arises if the linear region of operation is maintained without running into saturation.
- The piecewise-linear function reduces to a threshold function if the amplification factor of the linear region is infinitely large.

3) *Sigmoid Function*

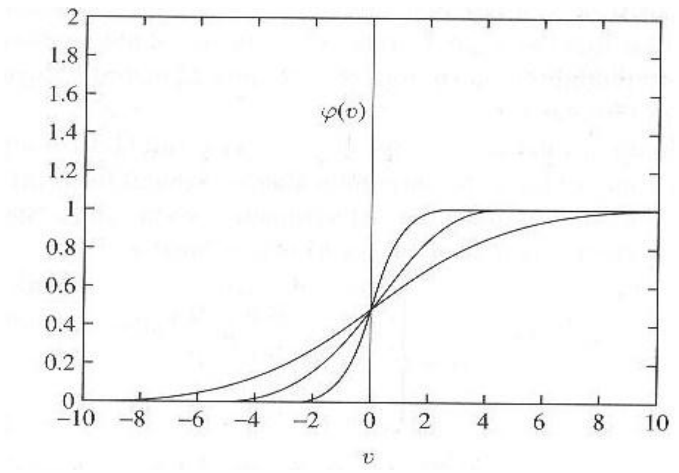


Fig. 1.7: *Sigmoid function*

The sigmoid function is by far the most common form of activation function used in the construction of artificial neural networks. It is defined as a strictly increasing function that exhibits smoothness an asymptotic properties.

An exemple is the *logistic function*, defined by

$$\varphi(v) = \frac{1}{1 + \exp(-av)}$$

where a is the *slope parameter* of the sigmoid function.
By varing the parameter a, we obtain sigmoid of different slopes. In fact, the slope at the origin equals a/4. In the limit, as the slope parameter approaches infinity, the sigmoid function becomes simply a threshold function. Whereas a threshold function assumes the value of 0 or 1, a sigmoid function assumes a continuous range of values from 0 to 1.

It is sometimes desiderable to have the activation function range from $-1$to $+1$,
in which case the activation function assumes an antisymmetric form with respect to the origin.
Specifically, the threshold function is redefined as

$$\varphi(v) = \begin{cases} 1 & \text{if } v > 0 \\ 0 & \text{if } v = 0 \\ -1 & \text{if } v < 0 \end{cases}$$

which is commonly referred to as the *signum function*.
For a sigmoid we may use the *hyperbolic tangent function*, defined by

$$\varphi(v) = \tanh\left(\frac{v}{2}\right) = \frac{1 - \exp(-v)}{1 + \exp(-v)}$$

1.3 Neural network as directed graphs

Signal-flow graphs with a well-defined set of rules were originally developed by Mason (1953, 1956) for linear networks. The presence of nonlinearity in the model of a neuron, however, limits the scope of their aplication to neural networks, but they still provide a neat method for the portrayal of the flow of signals in a neural network.

A *signal-flow graph* is a network of directed *links* (branches) that are interconnected at certain points called *nodes*:

A node j has an associated node signal x_j.
A directed link originates at node j and terminates on node k; it has an associated transfer function that specifies the manner in which the signal y_k at node k depends on the signal x_j at node j.

The flow of signals in the various parts of the graph is dictated by three basic rules:

RULE 1. A signal flows along a link only in the direction defined by the arrow on the link.

Two different types of links may be distinguished:

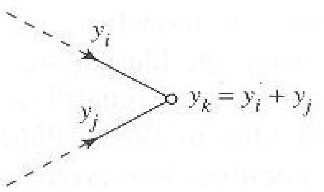
- *Synaptic links*, regulated by a *linear* input-output relation: the node signal x_j is multiplied by the synaptic weight w_{kj} to produce the node signal y_k.

$$\begin{tikzcd} x_j \arrow[r, "w_{kj}"] & y_k = w_{kj} x_j \end{tikzcd}$$

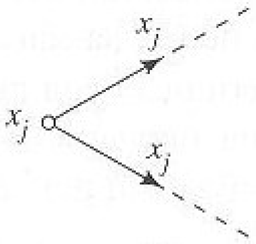
- *Activation links*, regulated in general by a *nonlinear* input-output relation: *φ(-)* is the nonlinear activation function.

$$\begin{tikzcd} x_j \arrow[r, "\varphi(\cdot)"] & y_k = \varphi(x_j) \end{tikzcd}$$

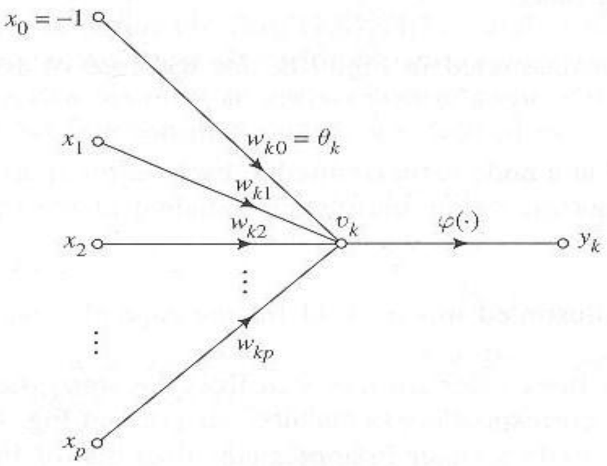
RULE 2. A node signal equals the algebraic sum of all signals entering the pertinent node via the incomnig links.



RULE 3. The signal at a node is transmitted to each outgoing link originating from that node, with the transmission being entirely indipendent of the transfer functions of the outgoing links.



Using these rules we may construct the signal-flow graph corresponding to the model of a neuron.



We may now introduce a mathematical definition of a neural network, based on the signal-flow graph of the model of a neuron:

A neural network is a directed graph consisting of nodes with interconnecting synaptic and activation links, characterized by four properties:

1. *Each neuron is represented by a set of linear synaptic links, an externally applied threshold, and a non linear activation link. The threshold is represented by a synaptic link with an input signal fixed at a value of*-1.
2. *The synaptic links of a neuron weight their respective input signals.*
3. *The weighted sum of the input signals defines the total internal activity level of the neuron in question.*
4. *The activation link squashes the internal activity level of the neuron to produce an output*

that represents the state variable of the neuron.

When the focus is restricted to signal flow from neuron to neuron, and not inside each neuron, we may use a reduced form of graph, which is characterized as it follows:

1. *Source* nodes supply input signals to the graph.
2. Each neuron is represented by a single node called a *computation* node.
3. The communication links interconnecting the source and computation nodes carry no weight; they only provide direction of signal flow in the graph.

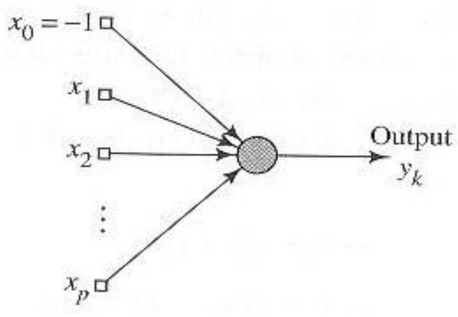


***Fig. 1.8**: Architectural graph of a neuron*

A partially complete directed graph defined in this way is referred to as an *architectural graph* describing the layout of the neural network.

1.4 Feedback

Feedback, in a dynamic system, is a process in which the output of an element in the system influences in part the input applied to that particular element.
It plays a major role in the study of a special class of neural networks known as *recurrent networks*.

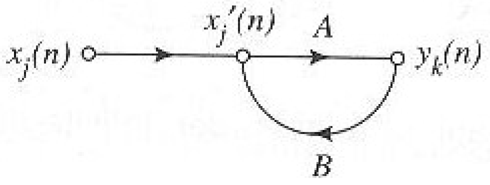


***Fig. 1.9**: Single-loop feedback system*

The figure above shows the signal flow graph of a single-loop feedback system, where the input signal $x_j(n)$, internal signal $x'_j(n)$, and output signal $y_k(n)$ are functions of the discrete-time variable n. The system is assumed to be linear, consisting of a forward channel and a feedback channel that are characterized by the operators A and B, respectively. In particular, the output of the forward channel determines in part its own output throught the feedback channel.

We can now state the following input-output relationships:

$$y_k(n) = A[x_j'(n)]$$

$$x_j'(n) = x_j(n) + B[y_k(n)]$$

from the previous equations, eliminating $x_j'(n)$:

$$y_k(n) = [x_j(n)]$$

where $A/1 - AB$ is referred to as the closed-loop operator of the system, and to AB as the open-loop operator.

For example, we consider the single-loop feedback system shown in figure, where A is a fixed weight w, B is a unit-delay operator z^{-1}, whose output is delayed with respect to the input by one time unit.

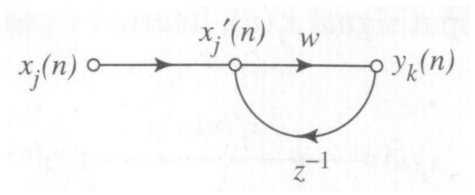


Fig. 1.10: *Single-loop feedback system*

We may then express the closed-loop operator of the system as

$$\frac{A}{1 - AB} = \frac{w}{1 - wz^{-1}} = w(1 - wz^{-1})^{-1}$$

Using the binomial expansion for $(1 - wz^{-1})^{-1}$, and subsituting eq.X in Y we get

$$y_k(n) = w \sum_{l=0}^{\infty} w^l z^{-l} \, [x_j(n)]$$

From the definition of z^{-1} we have

$$z^{-l} = x_j(n - l)$$

where $x_j(n - l)$ is a sample of the input signal delayed by l time units.
According to this, we may express the output signal $y_k(n)$ as an infinite weighted summation of present and past samples of the input signal $x_j(n)$:

$$y_k(n) = \sum_{l=0}^{\infty} w^{l+1} \, x_j(n - l)$$

Finally, we may distinguish two specific cases, depending on the value of the weight w, which influences the dynamic behavior of the system:

1. $|w| < 1$: the signal $y_k(n)$ is exponentially *convergent,* and we state that the system is stable. (FIG a)

2. $|w| \geq 1$: the output $y_k(n)$ is *divergent*, and the system is unstable. If $|w| = 1$ the divergence is linear, if $|w| > 1$ the divergence is exponential. (FIG

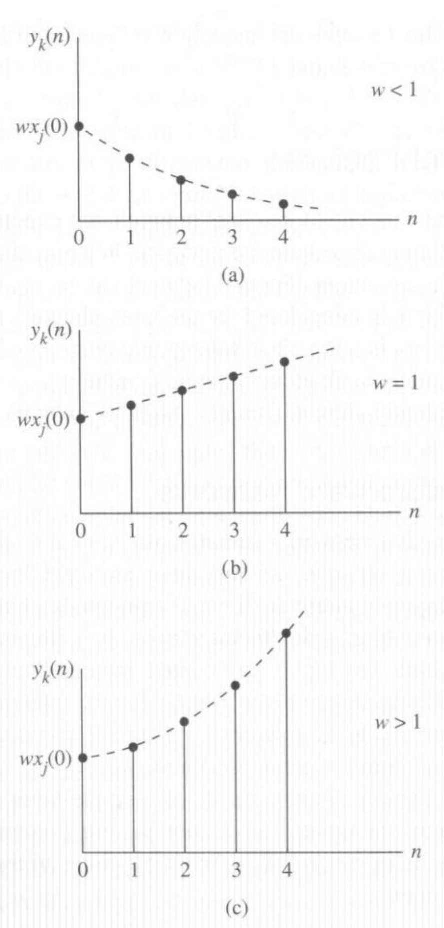


FIGURE 1.13 Time responses of Fig. 1.12 for three different values of forward weight *w*.

1.5 Network architectures

1) Single-Layer Feedforward Networks

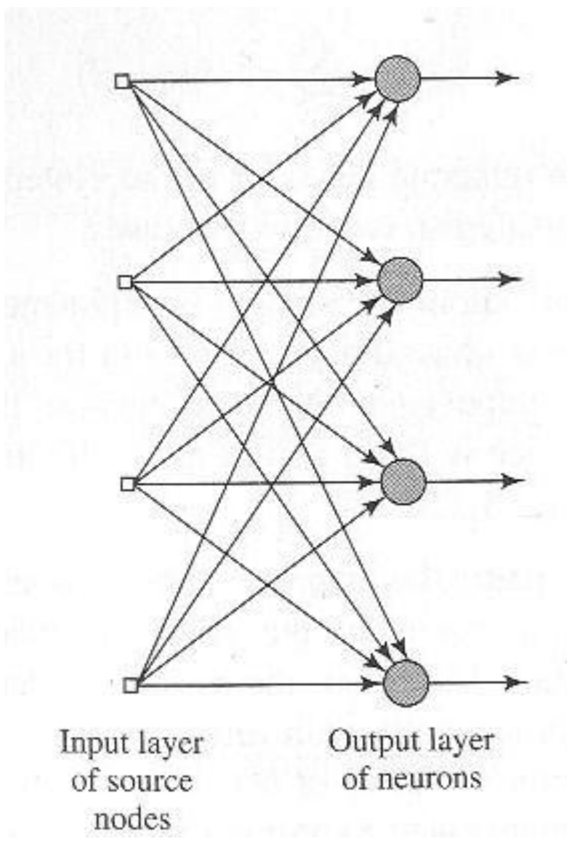


***Fig. 1.12**: Single-layer feedforward network*

A *layered* neural network is a network of neurons organized in the form of layers.
In its simplest form we have an input layer of source nodes that projects onto an output layer of neurons (computation nodes), but not vice versa. So we may say that this network is strictly of a *feedforward* type.
The designation "single layer" refers to the output layer, because we do not count the input layer since no computation is performed there.
A linear associative memory is an example of a single-layer network, where an output pattern (vector) is associated to an input pattern (vector), and information is stored in the network by virtue of modifications made to the synaptic weights.

2) Multilayer Feedforward Networks

This class of feedforward networks distinguishes itself by the presence of one or more *hidden layers*, whose computation nodes are correspondingly called *hidden neurons* or hidden units; their function consists in intervening between the external input and the network output. Typically, the neurons in each layer of the network have as their inputs the output signals of the preceding layer only. The set of output signals of the neurons in the final layer constitutes the overall response of the network to the activation pattern supplied by the source nodes in the first layer.

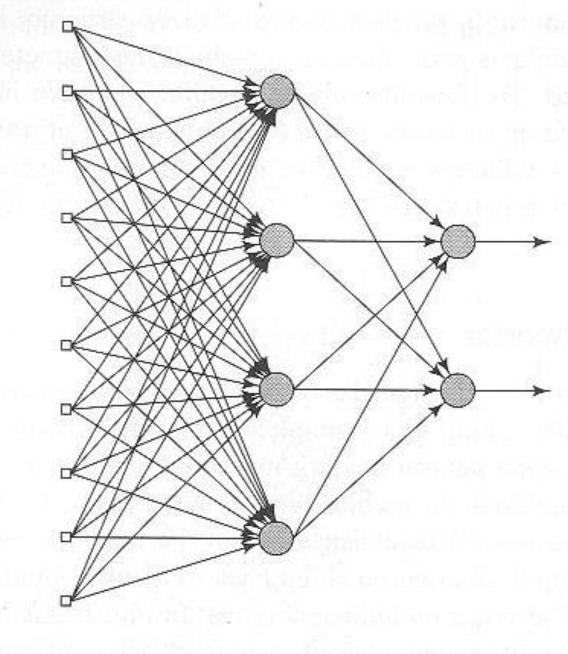


***Fig. 1.13**: Fully connected feedforward network with one hidden layer*

The architectural graph of the figure illustrates the layout of a multilayer feedforward network for the case of a single hidden layer. This network is referred to as a 10-4-2 network in that it has 10 source nodes, 4 hidden neurons, and 2 output neurons.
This neural network is also said to be *fully connected* in the sense that every node in each layer of the network is connected to every other node in the adjacent forward layer. If, however, some of the communications links are missing we say that the network is *partially connected* .
A form of partially connected network of particular interest is a locally connected network.

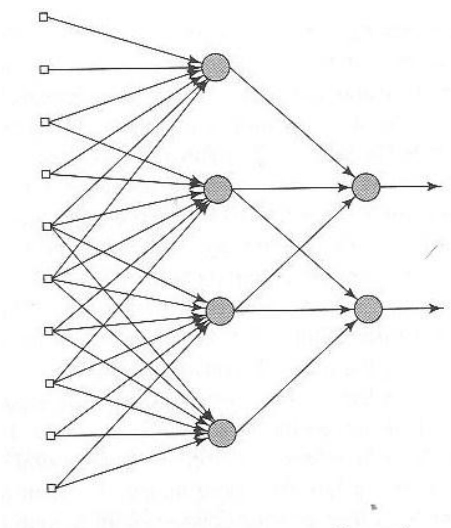


***Fig. 1.14**: Partially connected feedforward network*

Each neuron in the hidden layer is connected to a partial set of source nodes that lies in its immediate neighborhood; such a set of localized nodes is said to constitute the receptive field of the neuron. Similarly, each node in the output layer is connected to a local set of hidden neurons. This network has the same nodes of the previous one, but we may say that it has a specialized structure, in which each hidden neuron responds essentially to local variations of the source signal.

3) Recurrent Networks

A *recurrent* neural network is a feedforward network that has at least one *feedback* loop.

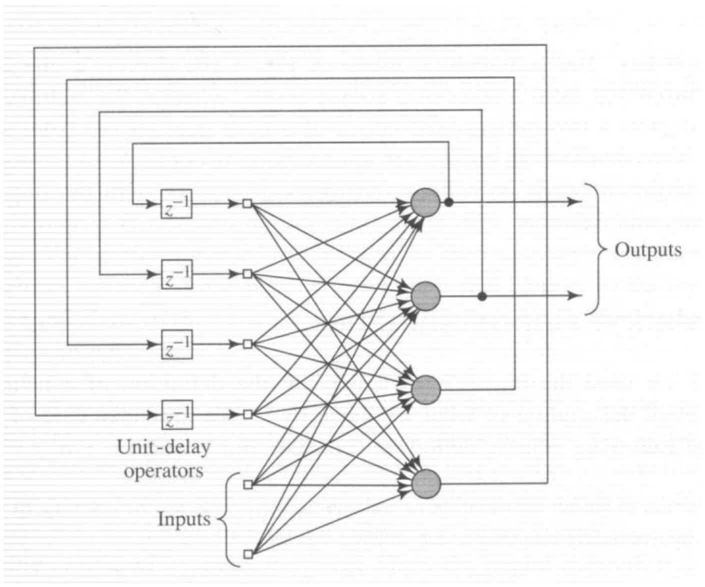


Fig. 1.15: *Recurrent network with hidden neurons*

For example (see figure), we consider a recurrent network consisting of two layers of neurons.
In this structure there are feedback and self-feedback loops, originated from the hidden neurons; those feedback connections have a strong impact on the learning capability of the network, and on its performance.
Moreover, we notice the use of *unit-delay elements* (denoted by z^{-1}), with a nonlinear dynamical behavior.

2. Learning process

One of the most interesting properties of a neural network is its ability of *learning* from its environment, and *improving* its performance throught learning. This improvement is realized(?) throught an iterative process of adjustement applied to its synaptic weights and thresholds.

A correct definition of learning in the field of neural networks may be:

Learning is a process by which the free parameters of a neural network are adapted throught a continuing process of stimulation by the environment in which the network is embedded. The type of learning is determined by the manner in which the parameter changes take place.

According to this definition, a learning process follows this sequence of events:

1. The environment stimulates the neural network.
2. The neural network undergoes changes as a result of this stimulation.
3. The neural network responds in a new different way to the environment, because of the changes that have occurred in its internal structure.

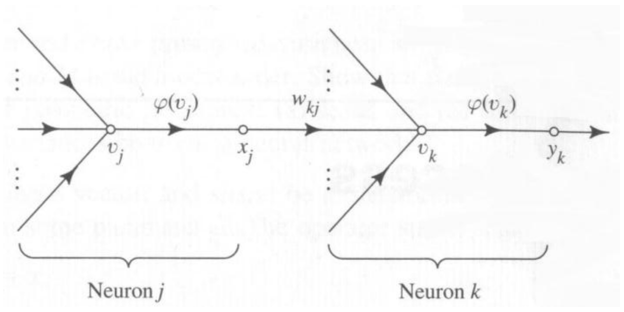


Fig. 2.1*: A pair of neurons imbedded in a neural network*

To be more specific, consider the situation depicted in the figure, which represents a pair of node signals x_j and $v_k$connected by a synaptic weight w_{kj}.
Signals x_j and v_k play the role respectively of the output of neuron *j* and of the internal activity of neuron *k*, and in this context they are commonly referred to as *presynaptic* and *postsynaptic* activities.
Let $w_{kj}(n)$ be the value of the synaptic weight w_{kj} at time *n*; at time n an adjustement $\Delta w_{kj}(n)$ is applied to the synaptic weight, yielding the updated value $w_{kj}(n+1)$.
Considering $w_{kj}(n)$ and $w_{kj}(n+1)$ as the old and new values of the synaptic weight w_{kj}, respectively, we may write:

$$w_{kj}(n+1) = w_{kj}(n) + \Delta w_{kj}(n) \quad (2.1)$$

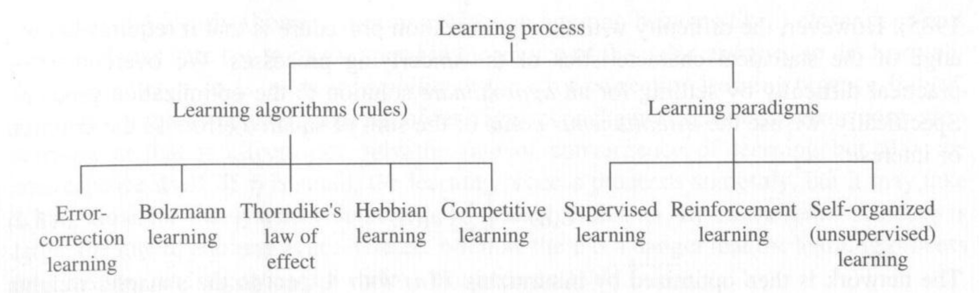
The equation sums up the overall effect of events 1 and 2 of the definition presented above:

- The adjustement $\Delta w_{kj}(n)$ is calculated as a result of stimulation by the environment (event 1).
- The updated value $w_{kj}(n+1)$ defines the change made in the network as a result of this stimulation (event 2).

The third event corresponds to the reevaluation of the new network's response, when operating with the updated set of parameters $\{w_{kj}(n+1)\}$.

2.1 Inside a learning process

Before we study learning processes, we have to make a foundamental distintcion between learning algorithms and paradigms.



A learning algorithm (rule) is a prescribed set of well defined rules for the solution of a learning problem. There are several learning algorithms, each of which offers advantages of its own. They essentially differ from each other in the way in which the adjustement Δw_{kj} is formulated. However, another important factor to be cosidered is the way in which a learning machine (a neural network) relates to its environment: in this context we speak of a learning paradigm, referring to a model of the environment in which the neural network operates.

Now we will make an example of a learning rule and a learning paradigm that will be useful in future, when considering the back-propagation algorithm for multilayer perceptrons.

2.2 An example of learning process algorithm: Error-Correction learning

Consider a neuron k at time n and let $d_k(n)$ denote some desired response for it. Let the corrisponding value of the actual response of this neuron be denoted by $y_k(n)$, this response is produced by a stimulus (vector) $\boldsymbol{x}(n)$ applied to the input of the network in which neuron *k* is embedded.

Typically, the actual response $y_k(n)$ of neuron k differs from the desired response $d_k(n)$. Hence, we may define an error signal as the difference between the target response and the actual response:

$$e_k(n) = d_k(n) - y_k(n) \text{ (2.2)}$$

Error-correction learning may be considered as an optimization problem, with the aim of minimizing a *cost function*, based on the error signal $e_k(n)$.
A commonly used criterion for the cost function is the mean-square-error criterion, defined as the mean-square value of the sum of squared errors:

$$J = E\left[\frac{1}{2}\sum_k e_k^2(n)\right] \; (2.3)$$

where E is the statistical expectation operator, and the summation is over all the neurons in the output layer of the network. Note that the factor ½ is used so as to simplify subsequent derivations resulting from the minimization of J with respect to free parameters of the network; moreover, we assume that the underlying processes are wide-sense stationary.

However, this optimization procedure requires knowledge of the statistical characteristics of the processes, and we try to overcome this practical difficulty by settling for an approximate solution to the optimization problem.
To realize this intention we use the istantaneous value of the sum of squared errors as the criterion of interest:

$$\xi(n) = \frac{1}{2}\sum_k e_k^2(n) \; (2.4)$$

We may thus optimize the network by minimizing $\xi(n)$ with respect to the synaptic weights. According to the error-correction learning rule, the adjustement $\Delta w_{kj}(n))$ is now given by:

$$\Delta w_{kj}(n) = \eta e_k(n) x_j(n) (2.5)$$

where η is a positive constant that determines the rate of learning.

The three equations (2.1), (2.2), (2.5), are represented in the signal flow graph below:

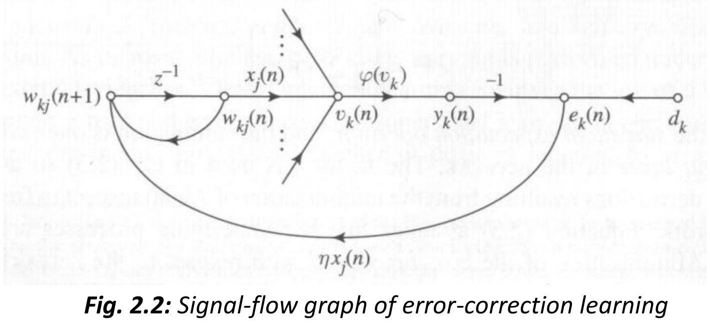


Fig. 2.2: *Signal-flow graph of error-correction learning*

Error signal $e_k(n)$, computed from eq. (2.2), is used to get the correction $\Delta w_{kj}(n)$ applied to the synaptic weight w_{kj} of neuron k (eq. (2.5)). Finally, throught eq. (2.1) we compute the new

updated value $w_{kj}(n+1)$ of the synaptic weight considered.
Furthermore the graph includes a storage element represented by the *unit-delay operator* z^{-1}:

$$z^{-1}[w_{kj}(n+1)] = w_{kj}(n) \quad (2.6)$$

and also the representation of the equations of the model of a neuron k

$$v_k(n) = \sum_j x_j(n) w_{kj}(n) \quad (2.7)$$

$$y_k(n) = \varphi(v_k(n)) \quad (2.8)$$

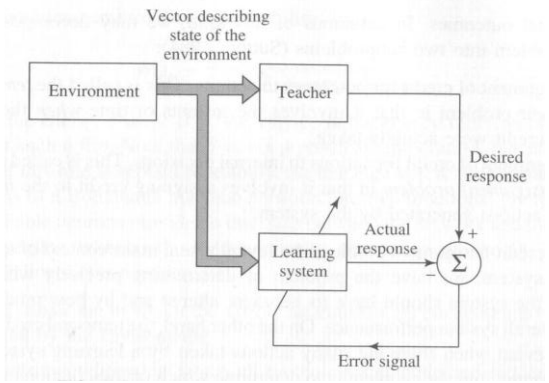
The learning-rate parameter η plays a major role: its value has to be chosen very carefully, since the stability of the whole process depends on it.
In fact, stated that error-correction learning behaves like a closed feedback system, η has a profound impact on the performace of the process, affecting both the the rate of convergence of learning and the convergence itself:

- if η is small, the learning process proceeds smoothly, but it may take a long time for the system to converge to a stable solution;
- if η is large, the rate of learning is accelerated, but there is a danger that the learning process may deverge and the system therefore becomes unstable.

We may visualize the cost function J as a multidimensional surface referred to as an error-performance surface.
The objective of error-correction learning algorithm is to start from a point on the surface, determined by the initial values of the synaptic weights, and then, step-by-step, move toward a point of global minimum.

2.3 An example of learning process paradigm: Supervised learning

The main resource of supervised or active learning is the presence of an external *teacher*, which has the knowledge of the environment, in terms of input-output examples. The environment, however, is unknown to the neural network of interest.



We define as *desired* or *target response* of the network, the response provided by the teacher corrisponding to a training vector drawn from the environment. Indeed, the desired response represents the optimum action to be performed by the network.
We define as *error signal*, the difference between the actual response of the network and the desired one.
The network parameters are adjusted under the action of the training vector and the error signal, with the aim of making the neural network emulating the teacher. This form of supervised learning is indeed the error-correction learning considered previously.
As a misure of the performance of the system, we may think in terms of the mean-squared error defined as a function of the free parameters of the network and this function may be visualized as an error surface, with the free parameters as coordinates. The true error surface is averaged over all possible input-output examples.
For such a system, the operating point moves torward a point of local or global minimum, to get the best performance over time and therefore learn from the teacher.
A foundamental element to reach this aim is the information carried by the gradient of the error surface; the gradient is a vector that points in the direction of *steepest descendent*, and the extimation of its istantaneous value is used by the system in the case of supervised learnig from examples.

2.4 Procedure to set network parameters and performance

We presented as a foundamental task for a neural network its ability to learn a model of the world in which is embedded.

Knowledge about the environment in interest is given by a set of examples, consisting of input-output pairs: an input signal and the corrisponding desired response of the neural network.
These data examples are then dived in three sets:

- the training set
- the validation or testing set
- the verification set

The training procedure consists firstly of a preliminary design of an appropriate architecture for the neural network, followed by a phase in which the training set is used for the adjustement of weights by means of a learning algorithm.
Secondly, the validation data set is used to decide when to stop the training, so as to avoid the so called "overtraining", which could make the network unable to identify new data belonging to the same classes of the training examples.
Finally, the recognition performance of the trained network is tested with data that has never been seen before: the verification set. The results on the verification set can be considered a true prediction of the neural network response on new data and the performance of the network on these data provides a proper benchmark evaluation metric for its performance as a predictor or classifier.

3. The Perceptron

The perceptron is the simplest form of a neural network used for the classification of a special type of patterns said to be linearly separable.
Basically, such a network consists of a single neuron with adjustable synaptic weigths and thresholds.

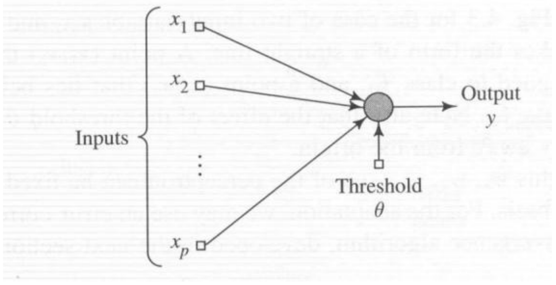


***Fig. 3.1**: Single-layer perceptron*

The first developer of an algorithm used to set this free parameters was Rosenblatt in 1958, who proved that if the vectors used to train the perceptron are chosen from two linearly separable classes, then the perceptron algorithm converges and positions the decision surface in the form of a hyperplane between the two classes.

As said before the perceptron is used to perform pattern classification; the number of classes that can be classified depends on the fact that the output layer includes one or more than one neuron.

We initially consider a single-layer perceptron with a single neuron.

Recalling the McCulloch-Pitts model of a neuron, consisting of a linear combiner followed by a hard limiter, we may write the linear combiner output (i.e. hard limiter input) as:

$$v = \sum_{i=1}^{p} w_i x_i - \theta$$

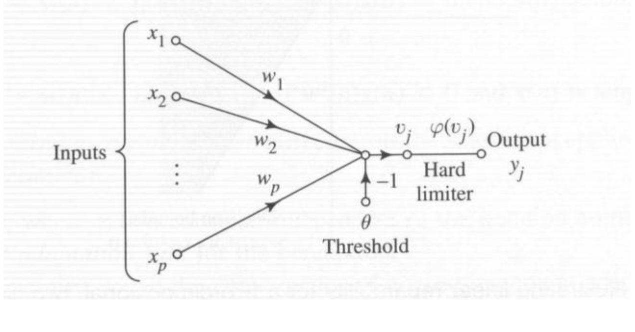


***Fig. 3.2**: Signal-flow graph of the perceptron*

The purpose of the perceptron is to classify the set of externally applied stimuli $x_1, x_2, .., x_p$ into one of the two classes C_1 or C_2. The decision rule for the classification is to assign the point represented by the inputs $x_1, x_2, .., x_p$ to class C_1if the perceptron output *y* is +1 and to class C_2 if it is -1.

The *p* input variables $x_1, x_2, .., x_p$ span a *p*-dimensional signal space, in which it is customary to represent a map of the decision regions.
In we consider an elementary perceptron, there are two decision regions separated by a hyperplane defined by the equation:

$$\sum_{i=1}^{p} w_i x_i - \theta = 0$$

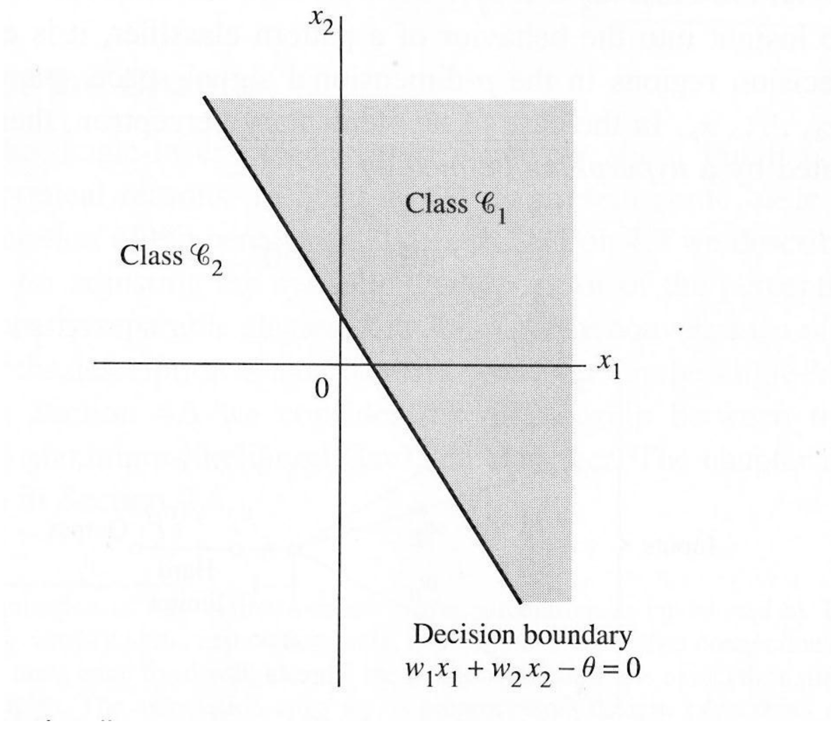


Fig. 3.3*: Linear reparability for a two dimensional, two-class pattern-classification problem*

The example in Fig.(3.3) shows the case of two input variables x1 and x2.
We assign to class C_1 and C_2 all the points (x_1x_2) that lie respectively above and below the boundary line.
The synaptic weights $w_1, w_2, .., w_p$ can be fixed or adapted on an iteration-by-iteration basis.
For the adaptation we may use an error-correction rule known as the perceptron convergence algorithm.

4. Multilayer Perceptrons

Multilayer perceptrons (MLPs), represent a generalization of the single-layer perceptron considered before. We refer to them as a class of multilayer feedforward networks, consisting of a cascade of a set of sensory units (source nodes) that constitute the input layer, one or more hidden layers of computation nodes, and an output layer of computation nodes.

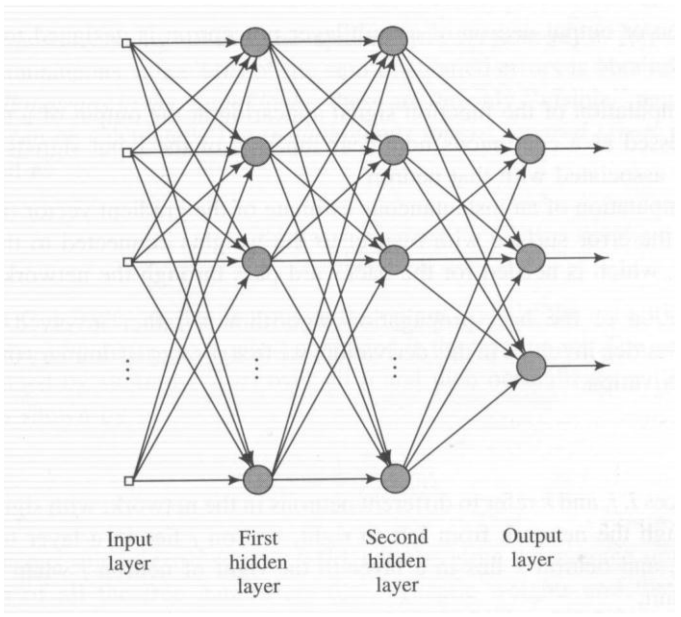


Fig. 4.1: *Multilayer perceptron with two hidden layers*

The main distinctive characteristics of a multilayer perceptron are:

1. Each neuron in the network is modelized at the output as including a nonlinearity.
   It is important to notice that this nonlinearity is smooth (i.e. differentiable everywhere), differently from the hard-limiting used in Rosenblatt's model.
   A function that satisfies nonlinearity in the form of *sigmoidal nonlinearity* is the *logistic function*:

$$y_j = \frac{1}{1+e^{-v_j}}$$

   where v_j is the net internal activity level of neuron *j*, and y_j is the output of the neuron.
   Nonlinarity is foundamental, because, otherwise, the input-output relation of the network could be reduced to that of a single layer perceptron.
   Moreover, the use of this function is biologically motivated, since it attempts to account for the refractory phase of real neurons.
2. The network contains layers of hidden neurons, that enable to learn complex tasks by extracting progressively more meaningful features from the vectors in input.
3. The network has a high degree of *connectivity*, determined by the synapses.

All this elements together with the training ability of a multilayer perceptron, contribute to his computing power.

However, the presence of forms of nonlinearity, the high connectivity of the network and the presence of hidden layers make the theoretical analysis of such a network harder to undertake. Moreover, the learning process gets more difficult because the search has to be conducted in a much larger space of possible functions.

The succes of multilayer perceptrons is due to their capability of solving difficult and diverse problems by training them in a supervised manner with an algorithm, based on the error-correction learning rule, known as the *error back-propagation algorithm*. The learning process performed with the algorithm is called *back-propagation learning.*

4.1 Back-Propagation Algorithm

First of all, we identify two kinds of signals used in multilayer perceptrons:

- *Function signals*. We refer to a function signal as an input signal (stimulus), that comes in at the input end of the network, propagates forward neuron-by-neuron, and emerges at the output end of the network as an output signal.
  Such a signal is a "function signal" in the sense that it provides a useful function at the output, and that it is calculated as a function of the inputs and associated weigths at each neuron of the network through which it passes.
- *Error signals*. This kind of signal originates at the output end of a neuron and propagates backward, layer-by-layer, through the network; it is foundamental to notice that its computation by every neuron involves an error-dependent function.

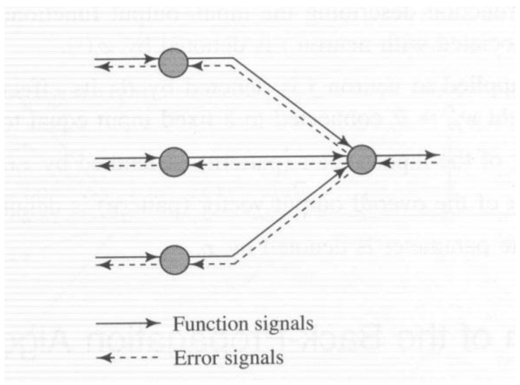


Fig. 4.2*: Directions of two basic signal flows in a multilayer perceptron*

Furthermore, neurons from hidden or output layers are designed to perform two computations:

1. The computation of the function signal at the output of a neuron, which is expressed as a continuous nonlinear function of the input signals and synaptic weight associated with that neuron
2. The computation of an istantaneous estimate of the gradient vector, which is needed for the backward pass through the network

Before we derive the back-propagation algorithm, let us introduce the following notation:

$\xi(n)$: istantaneous sum of error sqares at iteration n
$d_j(n)$: desired response for neuron j
$e_j(n)$: error signal at the output for neuron j for iteration n
$y_j(n)$: function signal at the output of neuron j
$w_{ji}(n)$: synaptic weight connecting the output of neuron i to the input of neuron j at iteration n
$\Delta w_{ji}(n)$: correction applied to the previous weight
$v_j(n)$: net internal activity level of neuron j; it constitutes the signal applied to the nonlinearity associated with neuron j

Also, in order to account for the bias weight we define the 0-th component of the input vector to each layer to be equal to 1; that is $u_{l,0} = 1$ ($w_{l,j,0}$ are the bias weights).

4.1.1 Derivation of the Back-Propagation Algorithm

The error signal at the output of neuron *j* at *n*-th iteration is defined by:

$$e_j(n) = d_j(n) - y_j(n), \quad \text{neuron j is an output node (4.1)}$$

The istantaneous sum of squared errors of the network is:

$$\xi(n) = \tfrac{1}{2}\textstyle\sum_{j\in C} e_j^2(n) (4.2)$$

where the istantaneous value of the squared error for neuron *j* is defined as $\frac{1}{2}e_j^2(n)$, and C is the set that includes all the neurons in the output layer.
Let *N* denote the total number of patterns in the training set. We define the average squared error as the summation of $\xi(n)$ over all n and then normalizing with respect to the set size *N*:

$$\xi_{av} = \frac{1}{N}\sum_{n=1}^{N} \xi(n) \ \ (4.3)$$

For a given training set, ξ_{av} represents the cost function as the measure of training set learning performance. The objective of the learning process is to adjust the free parameters so as to minimize it. To do so, we use a simple method of training in which the weights are updated on a pattern-by-pattern basis. The adjustements to the weights are made in accordance with the respective errors computed for each pattern presented to the network.
The arithmetic average of these individual weights changes over the training set is therefore an estimate of the true change that would result from modifying the weights based on minimizing the

cost function ξ_{av} over the training set.

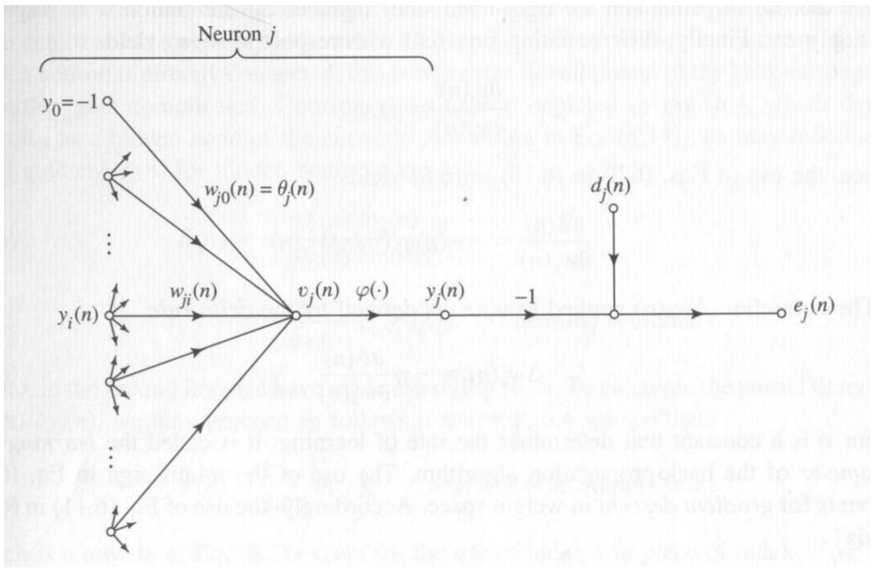


***Fig. 4.3**: Details of output neuron j*

Let us consider the situation depicted in fig (4.3), where neuron j is fed by a set of function signals coming from a layer to his left.
The net internal activity level $v_j(n)$ at the input of the nonlinearity associated with neuron j is therefore given by:

$$v_j(n) = \sum_{i=0}^{p} w_{ij}(n) y_i(n) (4.4)$$

where p is the total number of inputs applied to neuron j.
The synaptic weight w_{j0} (corresponding to the fixed input $y_0 = -1$) equals the threshold θ_j applied to neuron j. Hence the function signal appearing at the output of neuron j at iteration n is

$$y_j(n) = \varphi\left(v_j(n)\right) (4.5)$$

Now the algorithm applies a correction $\Delta w_{ji}(n)$ to the synaptic weight $w_{ji}(n)$, which is proportional to the istantaneous gradient $\delta\xi(n)/\delta w_{ji}(n)$. According to the chain rule, we may express this gradient as follows:

$$\frac{\delta\xi(n)}{\delta w_{ji}(n)} = \frac{\delta\xi(n)}{\delta e_j(n)} \frac{\delta e_j(n)}{\delta y_j(n)} \frac{\delta y_j(n)}{\delta v_j(n)} \frac{\delta v_j(n)}{\delta w_{ji}(n)} \; (4.6)$$

This gradient represents a sensivity factor, determining the direction of search in weight space for the synaptic weight wji.

Differentiating both sides of Eq.(4.2) with respect to $e_j(n)$, we get

$$\frac{\delta\xi(n)}{\delta e_j(n)} = e_j(n)(4.7)$$

Differentiating both sides of Eq.(4.1) with respect to $y_j(n)$, we get

$$\frac{\delta e_j(n)}{\delta y_j(n)} = -1 \ (4.8)$$

Next, differentiating both sides of Eq.(4.5) with respect to $v_j(n)$, we get

$$\frac{\delta y_j(n)}{\delta v_j(n)} = \varphi_j'\left(v_j(n)\right) (4.9)$$

Finally, differentiating Eq. (4.4) with respect to $w_{ji}(n)$ yields

$$\frac{\delta v_j(n)}{\delta w_{ji}(n)} = y_i(n)(4.10)$$

Hence, the use of Eqs. (4.7) to (4.10) in (4.6) yields

$$\frac{\delta\xi(n)}{\delta w_{ji}(n)} = -e_j(n)\varphi_j'\left(v_j(n)\right)y_i(n) \ (4.11)$$

The correction $\Delta w_{ji}(n)$ applied to w_{ji} is defined by the delta rule

$$\Delta w_{ji}(n) = -\eta\frac{\delta\xi(n)}{\delta w_{ji}(n)} \ (4.12)$$

where η is a constant that determines the rate of learning: *learning-rate parameter*.
The use of the minus sign in Eq. (4.12) accounts for gradient descent in weight space.
Accordingly, the use of Eq. (4.11) in (4.12) yields

$$\Delta w_{ji}(n) = \eta\delta_j(n)y_i(n) \ (4.13)$$

where the local gradient is itself defined by

$$\delta_j(n) = -\frac{\delta\xi(n)}{\delta e_j(n)}\frac{\delta e_j(n)}{\delta y_j(n)}\frac{\delta y_j(n)}{\delta v_j(n)} = e_j(n)\varphi_j'\left(v_j(n)\right) \ (4.14)$$

From the last two equations we note that a key factor involved in the weight adjustement $\Delta w_{ji}(n)$ is the error $e_j(n)$ at the output of neuron *j*.
We have to distinguish between two cases, depending on where in the network neuron *j* is located.

1. **Neuron *j* is an output node**

   We know that each output node of the network is supplied with a desired response of its own. Hence we may use Eq. (4.1) to compute the error signal ej associated with this neuron.
   Now, by using Eq. (4.14), we can easily compute the local gradient $\delta_j(n)$.

2. **Neuron *j* is a hidden node**

   In this case, we have no specified desired response for that neuron. Accordingly, the error signal for a hidden neuron would have to be determinated recursively in terms of the error signals of all the neurons to which that hidden neuron is directly connected.

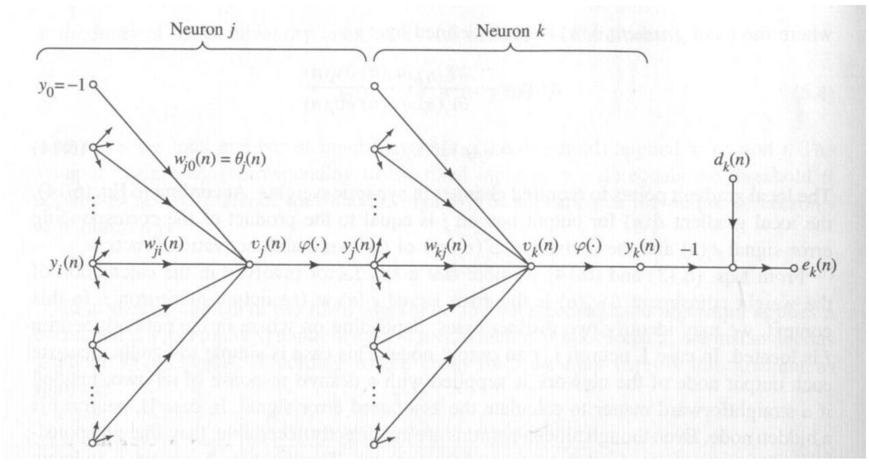


Fig. 4.4*: Details of output neuron k connected to hidden neuron j*

Let us consider the situation of Fig. (4.4): neuron j is a hidden node of the network.
We may define the local gradient delta for hidden neuron j as

$$\delta_j(n) = -\frac{\delta\xi(n)}{\delta y_j(n)}\frac{\delta y_j(n)}{\delta v_j(n)} = -\frac{\delta\xi(n)}{\delta y_j(n)}\varphi_j'\left(v_j(n)\right) \quad (4.15)$$

We have to calculate the partial derivate $\delta\xi(n)/\delta y_j(n)$
First of all we see that

$$\xi(n) = \frac{1}{2}\sum_{k\in C} e_k^2(n)(4.16)$$

In any event, differentiating Eq. (4.16) with respect to the function signal$y_j(n)$, we get

$$\frac{\delta\xi(n)}{\delta y_j(n)} = \sum_k e_k \frac{\delta e_k(n)}{\delta y_j(n)} (4.17)$$

Next, by using the chain rule for the partial derivate$\delta e_k(n)/\delta y_j(n)$ we can rewrite Eq. 17 in the equivalent form

$$\frac{\delta\xi(n)}{\delta y_j(n)} = \sum_k e_k(n) \frac{\delta e_k(n)}{\delta v_k(n)} \frac{\delta v_k(n)}{\delta y_j(n)} (4.18)$$

However, from Fig. 4 we note that

$$e_k(n) = d_k(n) - y_k(n) = d_k(n) - \varphi_k\big(v_k(n)\big) \; (4.19)$$

Hence

$$\frac{\delta e_k(n)}{\delta v_k(n)} = -\varphi_j'\big(v_k(n)\big) \;\; (4.20)$$

We also note that for neuron k, the net internal activity level is

$$v_k(n) = \sum_{j=0}^{q} w_{kj}(n) y_j(n) \;\; (4.21)$$

where q is the total number of inputs applied to neuron k.
Here again, the synaptic weight w_{k0} is equal to the threshold θ applied to the neuron k, and the corrisponding input y_0 is fixed at the value -1.
In any event, differentiating Eq. (4.21) with respect to $y_j(n)$ yields

$$\frac{\delta v_k(n)}{\delta y_j(n)} = w_{kj}(n) \; (4.22)$$

Thus, using Eq. (4.20) and (4.22) in (4.18), we get the desired partial derivate

$$\frac{\delta\xi(n)}{\delta y_j(n)} = -\sum_k e_k(n)\varphi_k'\big(v_k(n)\big) w_{kj}(n) = -\sum_k \delta_k(n) w_{kj}(n) \;\; (4.23)$$

where we have used Eq. (4.14).

Finally, using Eq. (4.23) in (4.15), we get the local gradient $\delta_j(n)$ for hidden neuron *j*, after rearranging terms, as follows:

$$\delta_j(n) = \varphi_j'\left(v_j(n)\right)\sum_k \delta_k(n)w_{kj}(n) \ (4.24)$$

The factor $\varphi_j'\left(v_j(n)\right)$ involved in the computation of the local gradient $\delta_j(n)$ in Eq. (4.24) depends only on the activation function associated with hidden neuron j.
The first set of terms in the summation over k, the $\delta_k(n)$, requires knowledge of the error signals $e_k(n)$, for all those neurons that lie in the layer to the immediate right of hidden neuron j.
The second set of terms, the $w_{kj}(n)$, consists of the synaptic weights associated with these connections.

The main relationship descending from the back-propagation algorithm are:

- The correction $\Delta w_{ji}(n)$ applied to the synaptic weight connecting neuron i to neuron j is defined by the delta rule

$$\begin{pmatrix} Weight \\ correction \\ \Delta w_{ji}(n) \end{pmatrix} = \begin{pmatrix} learning \\ rate\ parameter \\ \eta \end{pmatrix}\begin{pmatrix} local \\ gradient \\ \delta_j(n) \end{pmatrix}\begin{pmatrix} input\ signal \\ of\ neuron\ j \\ y_i(n) \end{pmatrix}$$

- The local gradient delta depends on whether neuron j is an output or an hidden node:
  - If neuron j is an output node, $\delta_j(n)$ equals the product of the derivative φ and the error signal $e_j(n)$.
  - If neuron j is a hidden node, $\delta_j(n)$ equals the product of the associated derivative di and the weighted sum of the δ 's computed for the neurons in the next hidden or output layer that are connected to neuron j.

4.1.2 Momentum

Let us consider the learning-rate parameter eta and the trajectory in wieght space computed by the method of steepest descendet, approximated by the back propagation algorithm.
The smaller we make eta, the smaller will the changes to the synaptic weights be and the smoother will be the trajectory in weight space. In this case, however, we will have to pay the cost of a slower learning rate.
On the other hand, if we increase the value of eta, the changes on the weights may bring the network to an instability status.
To avoid the danger of instability we may modify the delta rule by introducing a momentum term:

$$\Delta w_{ji}(n) = \alpha\Delta w_{ji}(n-1) + \eta\delta_j(n)y_j(n) \ \ (4.35)$$

where a is number that takes values in the range [0,1] called the momentum constant.

Solving(4.35) for $\varDelta w_{ji}(n)$ and substituting (4.11) and (4.14) we may rewrite it in the equivalent form:

$$\varDelta w_{ji}(n) = -\eta \sum_{t=0}^{n} \alpha^{n-t} \frac{\delta \xi(t)}{\delta w_{ji}(t)} \quad (4.36)$$

Considering the Eq. (4.36), we may make the following considerations:

1. When the partial derivate de/dw has the same algebraic sign on consecutive iterations, Dw grows in magnitude, and so the weight w is adjusted by a large amount. Hence the inclusion of momentum in the back-propagation algorithm tends to accelerate descent in steady downhill directions.
2. When the partial derivate de/dw has opposite signs on consecutive iterations, Dw shrinks in magnitude, and so the weight w is adjusted by a small amount. Hence the inclusion of momentum has a stabilizing effect in directions that oscillates in sign.

The introduction of the momentum terms, enable the algorithm to get in gradient descent direction faster, avoiding oscillations due to the change of the sign.

4.2 MLP functional capabilities

The capabilities of the multilayer perceptron can be wiewed from three different perspectives:

- The possibility to implement Boolean logic functions
- The ability to implement non linear transformations for functional approximations problems
- The capability to partition the pattern space for classification problems

We will focus mainly on the last point giving an example of an application for medical decision.

5. Neural networks as medical decision support

Medical decision support with neural networks is a field exposed to a constant accelerating evolution and this fact is proved by the growing interest in the medical literature, during the last years.

Neural networks can play a key role in this field as they are effective at multifactorial analysis.
To be more specific, many medical decisions are made in situations in which multiple factors must be weighted, and these mathematical tools provide an efficent manner to manage them.

The rapid development of this kind of "smart computing" is mainly due to three factors

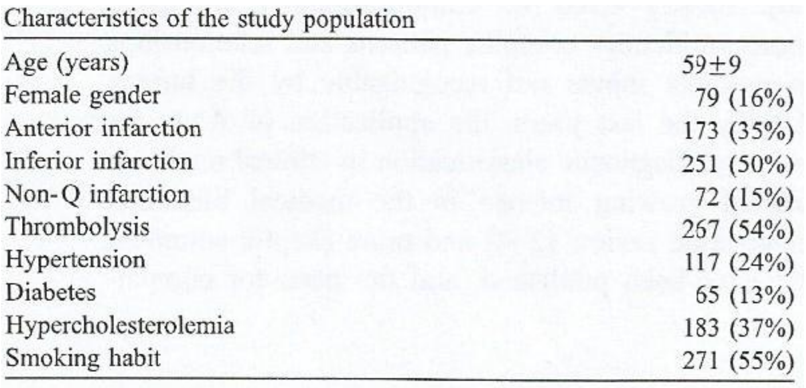
- ever-increasing mediacal databases
- the proliferation of new medical markers
- the accumulation of experience by responsible neural network experts

An important application of neural netwoks as medical decision support is their use for prognostic and diagnostic classification, since these computational supports can foresee the exit of a medical operation or the progress of a diseas in an automatic and quite accurate manner.

5.1 Neural networks for risk stratification following uncomplicated myocardial infarction

We want now to focus on a practical application of neural networks in medical decision support systems, by presenting an example of artificial neural network for risk stratification following uncomplicated myocardial infarction

We refer to a study about clinical risk stratification on a population of 496 patients recovering from acute myocardial infarction.



Characteristics of the study population	
Age (years)	59±9
Female gender	79 (16%)
Anterior infarction	173 (35%)
Inferior infarction	251 (50%)
Non-Q infarction	72 (15%)
Thrombolysis	267 (54%)
Hypertension	117 (24%)
Diabetes	65 (13%)
Hypercholesterolemia	183 (37%)
Smoking habit	271 (55%)

Fig. 5.1: *Characteristics of the study population*

This population, whose clinical characteristics are reported in Table (5.1), had been selected on the basis of different parameters and all patients had undergone exercise elecrtocardiography and pharmacological stress echocardiography in random order within 14 days of the acute event off therapy and were prospectively followed-up for a mean of 24 months.

5.1.1 Decisional process

Collected data are given as input to a feedforward neural network, made up of up to two hidden layers composed of five to forty neurons for layer, and an output layer consisting of a sigle neuron. To train the network it was used a function implementing the back propagation algorithm and as activation function the logistic one had been used.
Accordingly, the output may assume any value in the range [0,1], and a validation criterion is therefore needed to choice the continuous value to assign to one out of the N discrete values that represent the output classes.

5.1.2 Classification

Pattern Classification is a learning task characterized by a fixed number of categories or classes, into which input stimuli (activations) are to be classified.
To perform it, the neural network first udergoes a training session, during which the network is repeatedly presented a set of input patterns along with the category to which each particular pattern belongs. Then, a new pattern is presented to the network, which is part of the same population of patterns but that has not been seen before. The problem is now for the network to classify this new pattern correctly.
The main advantage of using a neural network consists in the fact that it can construct nonlinear decision boundaries between the different classes in a non parametric fashion, and thereby offer a practical method for solving highly complex pattern classification problems.
Note that pattern classification as described above is a supervised learning problem.

In the case of a network with a single output neuron, and then two output classes, an "high" value is assigned to the elements belonging to a class, whereas a "low" value is assigned to the elements of the other.

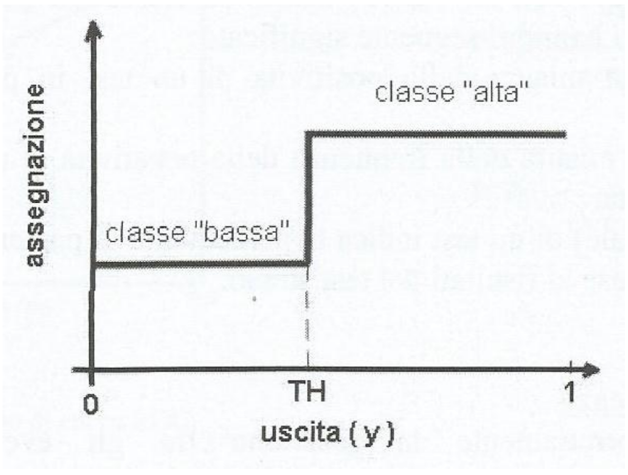


Fig. 5.2: *Assignement criterion*

The criterion consists thus, in setting a decision threshold in the range (0,1), so as to assign output values greater than the threshold to the "high" class, and the other values to the "low" class.

Let us introduce some indexes to evaluate the quality of the classifier:

$$Sensibility = \frac{TP}{TP + FN}$$

$$Specificity = \frac{TN}{TN + FP}$$

$$Accuracy = \frac{TP + TN}{TOTAL}$$

where the terms:

TP	(true positives) outlines the number of patients belonging to the "high" class, that are correctly classified
FP	(false positives) outlines the number of patients belonging to the "low" class, that are wrongly classified as "high" class elements
TN	(true negatives) outlines the number of patients belonging to the "low" class, that are correctly classified
FN	(false negatives) outlines the number of patients belonging to the "high" class, that are wrongly classified as "low" class elements

If we consider the class of patients with a particular disease as the "high" class, and the class of the healthy patients as the "low" class:

Sensibility is the measure of the positiveness of the test, in presence of the considered desease;

Specificity is the measure of the frequency of the negativeness of the test, in absence of the considered desease;

Accuracy of a test, outlines the rate of patients correctly classified, according to test's results.

5.1.3 Roc plots

To estabilish the value of the threshold that maximizes the probability of correct classifications we can use the ROC plots: a ROC plot is typically graphed as a plot of sensivity on the ordinate and the false-positive fraction on the abscissa.

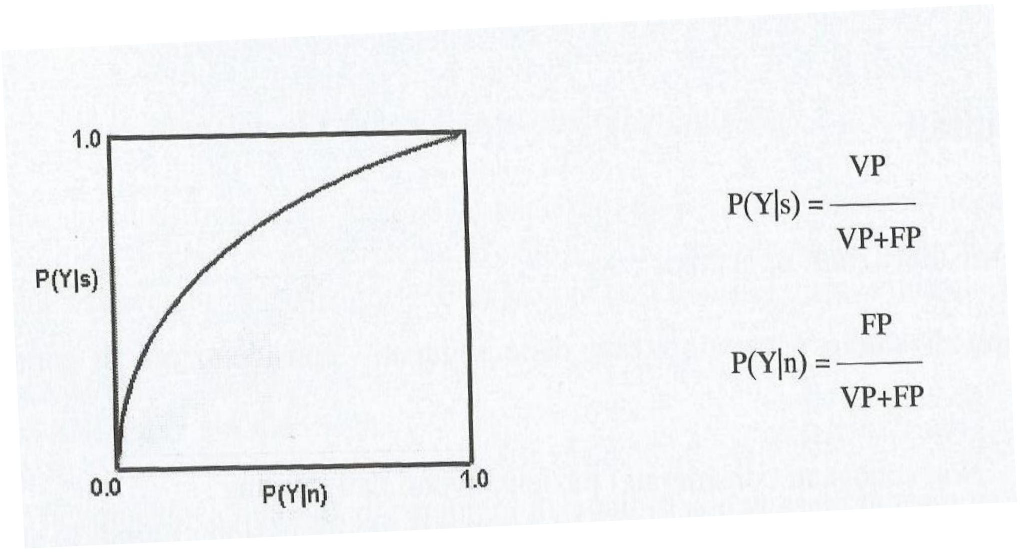
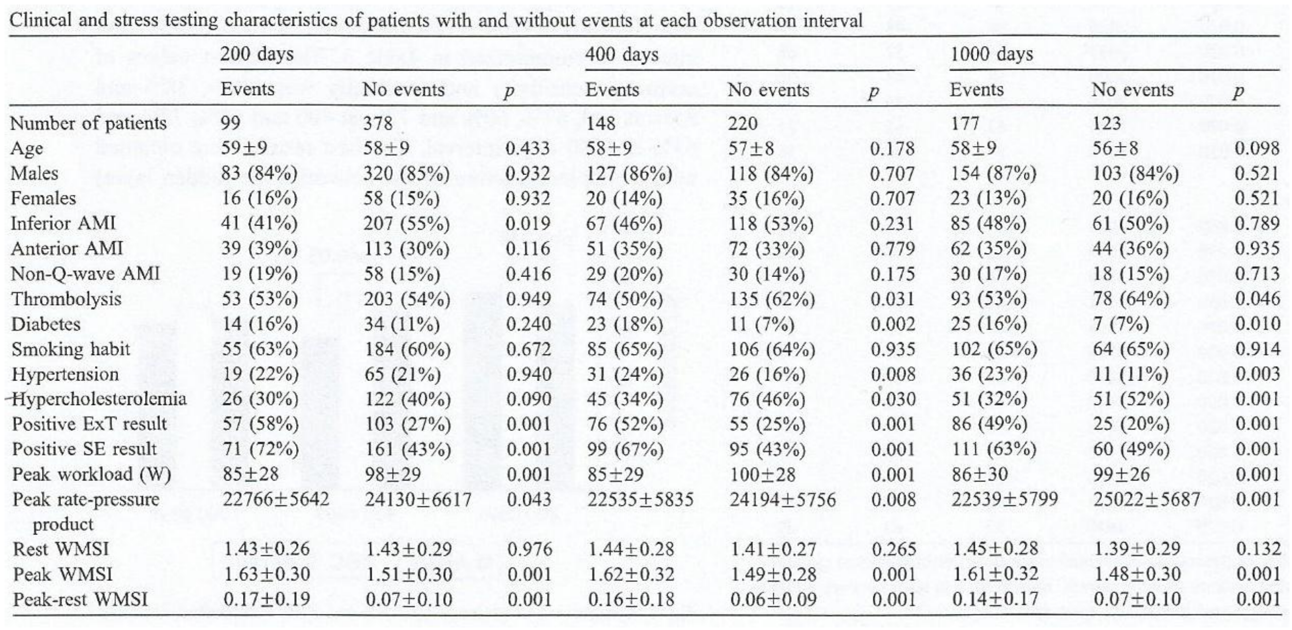


***Fig. 5.3**: ROC plot*

The optimum value of the threshold is the one associated to the point that gets closer to the extremity (0,1) of the graph.
According to the plot, when the extremity (0,1) is reached, that means the network has correctly classified all the patterns, since in this case we have P(TP)=1 and P(FP)=0.

5.1.4 Analysis and results of the test

Fifteen clinical (see table 5.1), exercise ECG and stress echocardiography variables were selected as predictors of the cumulative endpoint of cardiac death, nonfatal infarction and unstable angina. Shorts (200) days, medium (400 days) and long (1000 days) term observation intervals, including 50%, 75% and 90% of the events, respectively were considered. At each interval, any patient was assigned to the "event" or "no event" class.



Clinical and stress testing characteristics of patients with and without events at each observation interval

	200 days			400 days			1000 days		
	Events	No events	*p*	Events	No events	*p*	Events	No events	*p*
Number of patients	99	378		148	220		177	123	
Age	59±9	58±9	0.433	58±9	57±8	0.178	58±9	56±8	0.098
Males	83 (84%)	320 (85%)	0.932	127 (86%)	118 (84%)	0.707	154 (87%)	103 (84%)	0.521
Females	16 (16%)	58 (15%)	0.932	20 (14%)	35 (16%)	0.707	23 (13%)	20 (16%)	0.521
Inferior AMI	41 (41%)	207 (55%)	0.019	67 (46%)	118 (53%)	0.231	85 (48%)	61 (50%)	0.789
Anterior AMI	39 (39%)	113 (30%)	0.116	51 (35%)	72 (33%)	0.779	62 (35%)	44 (36%)	0.935
Non-Q-wave AMI	19 (19%)	58 (15%)	0.416	29 (20%)	30 (14%)	0.175	30 (17%)	18 (15%)	0.713
Thrombolysis	53 (53%)	203 (54%)	0.949	74 (50%)	135 (62%)	0.031	93 (53%)	78 (64%)	0.046
Diabetes	14 (16%)	34 (11%)	0.240	23 (18%)	11 (7%)	0.002	25 (16%)	7 (7%)	0.010
Smoking habit	55 (63%)	184 (60%)	0.672	85 (65%)	106 (64%)	0.935	102 (65%)	64 (65%)	0.914
Hypertension	19 (22%)	65 (21%)	0.940	31 (24%)	26 (16%)	0.008	36 (23%)	11 (11%)	0.003
Hypercholesterolemia	26 (30%)	122 (40%)	0.090	45 (34%)	76 (46%)	0.030	51 (32%)	51 (52%)	0.001
Positive ExT result	57 (58%)	103 (27%)	0.001	76 (52%)	55 (25%)	0.001	86 (49%)	25 (20%)	0.001
Positive SE result	71 (72%)	161 (43%)	0.001	99 (67%)	95 (43%)	0.001	111 (63%)	60 (49%)	0.001
Peak workload (W)	85±28	98±29	0.001	85±29	100±28	0.001	86±30	99±26	0.001
Peak rate-pressure product	22766±5642	24130±6617	0.043	22535±5835	24194±5756	0.008	22539±5799	25022±5687	0.001
Rest WMSI	1.43±0.26	1.43±0.29	0.976	1.44±0.28	1.41±0.27	0.265	1.45±0.28	1.39±0.29	0.132
Peak WMSI	1.63±0.30	1.51±0.30	0.001	1.62±0.32	1.49±0.28	0.001	1.61±0.32	1.48±0.30	0.001
Peak-rest WMSI	0.17±0.19	0.07±0.10	0.001	0.16±0.18	0.06±0.09	0.001	0.14±0.17	0.07±0.10	0.001

Fig. 5.4: *Clinical and stress testing characteristics of patients with and without events at each observation interval*

By analizing the results, we observe that the best performances are achieved when considering a single hidden layer network and a lighter training.
Moreover, if we compare the prognostic accuracy provided by neural network (70% at 200, 67 at 400%, 64% at 1000 days respectively) with the default accuracy which was 74% at 200, 57% at 400 and 61% at 1000 days interval, we notice that 200-days classification gets the worst results, whereas we have an improvement in the 400-days classification.

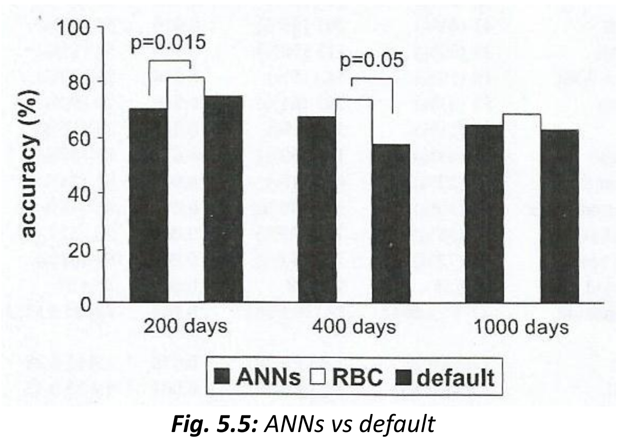
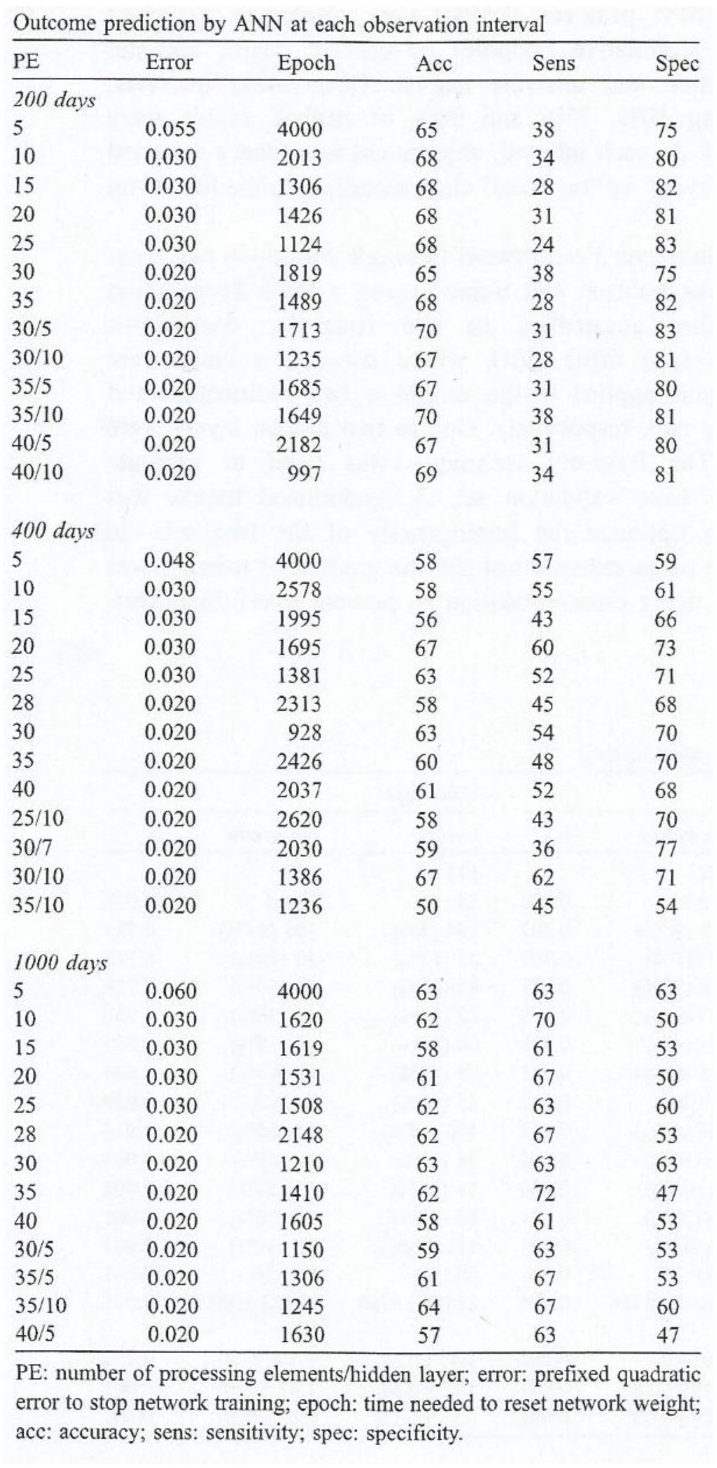


Fig. 5.5: *ANNs vs default*



Outcome prediction by ANN at each observation interval

PE	Error	Epoch	Acc	Sens	Spec
200 days					
5	0.055	4000	65	38	75
10	0.030	2013	68	34	80
15	0.030	1306	68	28	82
20	0.030	1426	68	31	81
25	0.030	1124	68	24	83
30	0.020	1819	65	38	75
35	0.020	1489	68	28	82
30/5	0.020	1713	70	31	83
30/10	0.020	1235	67	28	81
35/5	0.020	1685	67	31	80
35/10	0.020	1649	70	38	81
40/5	0.020	2182	67	31	80
40/10	0.020	997	69	34	81
400 days					
5	0.048	4000	58	57	59
10	0.030	2578	58	55	61
15	0.030	1995	56	43	66
20	0.030	1695	67	60	73
25	0.030	1381	63	52	71
28	0.020	2313	58	45	68
30	0.020	928	63	54	70
35	0.020	2426	60	48	70
40	0.020	2037	61	52	68
25/10	0.020	2620	58	43	70
30/7	0.020	2030	59	36	77
30/10	0.020	1386	67	62	71
35/10	0.020	1236	50	45	54
1000 days					
5	0.060	4000	63	63	63
10	0.030	1620	62	70	50
15	0.030	1619	58	61	53
20	0.030	1531	61	67	50
25	0.030	1508	62	63	60
28	0.020	2148	62	67	53
30	0.020	1210	63	63	63
35	0.020	1410	62	72	47
40	0.020	1605	58	61	53
30/5	0.020	1150	59	63	53
35/5	0.020	1306	61	67	53
35/10	0.020	1245	64	67	60
40/5	0.020	1630	57	63	47

PE: number of processing elements/hidden layer; error: prefixed quadratic error to stop network training; epoch: time needed to reset network weight; acc: accuracy; sens: sensitivity; spec: specificity.

Fig. 5.6: *Outcome prediction by neural network*

6. Conclusions

In this work we evaluated the computational technique of neural networks.
Based on the structure of the brain, these mathematical models can provide an efficient way to handle multifactorial analysis, as they can employ multiple factors in solving several problems such as prediction, classification and pattern recognition.
The main feature of the neural networks is their capability of changing the values of their internal parameters by rearranging weights and layers, and being in this sense adaptive devices.
These changes are made according to learning rules, and we discussed a supervised learning rule: the back-propagation algoritmh.
As far as practital aspects of neural networks is concerned, we focused on a medical application for prognostic classification.
We studied a multilayered neural network, trained by the back-propagation algorithm, that quantified the risk stratification for patients after uncomplicated myocardial infarction.
By changing network parameters, we may observe the different response of the network in terms of performance measures (accuracy, sensivity and specificity).
In our example emerged an insufficent accuracy as far as short-term prognostic is concerned, whereas a quite better performance is reached for longer-term prognostic.
We can conclude that neural network are nowadays beginning to play an important and hopeful emerging role in medical decision system, since they can provide a methodology able to manage the great availability of medical data, with databases rapidly expanding.

7. References

[1] Haykin S, "Neural Networks A Comprehensive foundation", MacMillan 1994.

[2] Bigi R, Gregori D, Cortigiani L, Desideri A, Chiarotto F, Toffolo G, "Artificial neural networks and robust bayesian classifiers for risk stratification following uncomplicated myocardial infarction", International Journal of Cardiology, Elsevier 2004; pp.481-487.

[3] Dayhoff J, DeLeo J, "Artificial neural networks: Opening the Black Box", CANCER supplement, 2001, volume 91, Number 8; pp.1615-1633.

[4] Hush D, Horne B, "Progress in Supervised neural networks", IEEE signal processing magazine, 1993; pp.8-33