IL NUOVO CIMENTO Vol. 30 C, N. 3 Maggio-Giugno 2007
DOI 10.1393/ncc/i2007-10244-8

Neural networks: A review

T. Squartini
Dipartimento di Fisica, Università di Siena - via Roma 56, 53100 Siena, Italy

(ricevuto il 2 Maggio 2007; revisionato il 22 Settembre 2007; approvato il 23 Ottobre 2007; pubblicato online il 5 Dicembre 2007)

Summary. — Neural networks can be defined in a variety of ways. Biology, informatics, mathematics, physics: each discipline gave its own definition. We will try to explain what neural networks are and why they are so useful. The paper is divided in two parts: in the first one preliminary definitions are given and general properties are illustrated. In the second part structures and dynamics of a number of networks are analyzed.

PACS 84.35.+i – Neural networks.

1. – Introduction

The human brain is structured as a collection of 10^{11} cells, the neurons; each of them is connected to hundreds of other neurons, building an interconnected structure called neural network. This structure results in a very powerful computational device.

One aim of computer science is to study these potentialities to build machines driven by artificial human brains (that is by artificial neural networks). This approach is based on two principles: connectivism and neuromorphism. Connectivism identifies the peculiar properties of the brain as a consequence of the great number of connections of very simple units (neurons). Neuromorphism establishes an "operative procedure", based on pure imitation of biology: to reproduce the function of an organ it is necessary to reproduce its shape. Application of both principles conduces to project different machines from modern computers. Mimiking biology we encounter the notions of parallel calculus (neurons work independently), learning (algorithms are no more needed since neurons do not possess a code of instructions resuming the whole activity of the brain), distributed and content-addressable memory (neurons in brain "share" information: every day a lot of neuronal cells die but we do not lose memory; moreover we are able to remember "by content" that is, when stimulated with a color, we can remember a perfume). On the contrary, modern computers have an address-based memory.

Physics at last studies neurons because the activity of the brain can be interpreted as a collection of emergent properties of the whole neural net: in this sense, neural networks are complex systems [1, 2]. A first structure-based definition of neural networks

could therefore be the following: neural networks are collections of simple, interconnected elements, called neurons. An alternative (and activity-based) definition could be: neural networks are structures that process "information", reacting to the stimuli of external world by the production of adequate answers [3, 4].

2. – Biology of the neuron

2˙1. *Structure.* – Neurons are the basic units of the human nervous system. They are composed of different parts: dendrites (where synaptic terminals of the other connected neurons end), cellular body (where the nucleus is), axons (that "conduct" electric signals) and synaptic terminals (the terminals of the axon, ending over dendrites of other neurons). Note that "synapses" are not a physical part of the neuron: they are the regions of contact between a dendrite and a synaptic terminal. Dale stated that the synapses of a neuron could be only excitatory or inhibitory, the first ones stimulating the postsynaptic neuron to conduce, the other ones inhibiting it (even if exceptions were found it remains valid for the majority of neurons in the brain).

2˙2. *Activity.* – The neuronal activity is based on the ionic concentrations and electric potential values on both sides of the axonic membrana. For a mole of ions of a single chemical species (with charge F) these effects are accounted by the Nerst formula

$$V_{\text{in}} = \frac{RT}{F} \ln \frac{c_{\text{out}}}{c_{\text{in}}} \tag{1}$$

(where V_{in} is measured with respect to V_{out}, set to zero). Nevertheless, because of the simultaneous presence of three different chemical species (P is the permeability of the membrana to them), the correct relation is the Goldman equation

$$V_{\text{in}} = \frac{RT}{F} \ln \frac{P[\text{K}^+]c_{\text{out}}[\text{K}^+] + P[\text{Na}^+]c_{\text{out}}[\text{Na}^+] + P[\text{Cl}^-]c_{\text{in}}[\text{Cl}^-]}{P[\text{K}^+]c_{\text{in}}[K^+] + P[\text{Na}^+]c_{\text{in}}[\text{N}a^+] + P[\text{Cl}^-]c_{\text{out}}[\text{C}l^-]}. \tag{2}$$

An active mechanism to compensate passive ionic tendencies and to maintain ionic concentrations is also present: the sodium-potassium pump. The value of V_{in} is determined by the equilibrium between these tendencies. When electric impulses are present, P's are modified and the Goldman relation no more holds.

3. – A mathematical model of neuron

McCulloch and Pitts were among the first ones who tried to build a mathematical model of neurons. They did not try to explain biology; they only "imitated" Nature to build an efficient computational device. Their neuron is a very simple element that receives inputs from other neurons, calculates their sum (weighted with appropriate coefficients, representing the strength of the connections between pairs of neurons) and compares it with a threshold value. According to it, the neuron output may change [5, 6].

3˙1. *Synaptic weights.* – Weights model synapses. They are often symbolized with w_{ij}, indicating a connection between neuron i and neuron j. The symbol also indicates that information flows from neuron i to neuron j. Weights are nothing but numerical coefficients whose sign indicates excitatory ($w_{ij} > 0$) or inhibitory ($w_{ij} < 0$) behavior

(if $w_{ij} = 0$ then no connection is present). Connections may be symmetric ($w_{ij} = w_{ji}$) or asymmetric ($w_{ij} \neq w_{ji}$) and loops may or may not be present ($w_{ii} \neq 0$; $w_{ii} = 0$). Weights between pairs of neurons can be represented as a matrix $\mathbf{W} = \{w_{ij}\}$.

3˙2. *Input.* – The input to the i-th neuron can be computed in many ways. The more common is the dot-product

$$P_i = \mathbf{s}_{(i)}^T \cdot \mathbf{W}_i = \sum_j w_{ji} s_j, \tag{3}$$

where the index j labels neurons sending "output signals" to the i-th neuron. Their outputs, s_j's, are the components of the column vector $\mathbf{s}_{(i)}^T$ and $\mathbf{W}_i$ is the i-th column of the weight matrix (here considered as a vector). Alternative formulas describing the i-th input exist: they are used principally in biologic/psychologic models to account for "tendencies" in human behaviors. Our personal inclinations drive our choices and make us unique individuals: our neurons are different from those of the other people and this can be accounted for, mathematically, by adding a sort of "internal" threshold (θ_i), or bias (b_i, that is a "special" weight connecting the i-th neuron to a unit whose output is always set to 1), to the input of our neurons:

$$P_i = \textstyle\sum_j w_{ji} s_j + b_i, \quad P_i = \sum_j w_{ji} s_j + \theta_i.$$

A possible different kind of input is the distance

$$P_i \equiv d_i = ||\mathbf{s}_{(i)}^T - \mathbf{W}_i||\,, \tag{4}$$

P_i is also called the activation of the i-th neuron.

3˙3. *Output.* – The output of a neuron is computed by applying a function f (the activity function) to the activation P_i. It can be the linear function

$$f(P_i) = P_i \tag{5}$$

or a non-linear one. Common discrete-threshold non-linear functions are the sign function

$$f(P_i) = \mathrm{sgn}(P_i) \tag{6}$$

and the step, or Heaviside, function

$$f(P_i) = \Theta(P_i). \tag{7}$$

We can also choose continuous functions as the hyperbolic tangent

$$f(P_i) = \tanh(P_i) \tag{8}$$

or the logistic function

$$f(P_i) = \frac{1}{1 + \exp[-P_i]}\,. \tag{9}$$

Specifically with distance inputs, a (normalized) Bell function can be used

$$f(P_i) = N \exp[-P_i^2]. \tag{10}$$

The value $s_i = f(P_i)$ is also known as the state of the neuron. According to the f we can choose $s_i = \{0, 1\}$, $s_i = \{-1, 1\}$, $s_i \in (0, 1)$, $s_i \in (-1, 1)$: neurons whose state is discrete and whose values are 0 or 1 (0 or -1) are called binary (bipolar) [7].

A neuron, therefore, is a device for numeric calculus that receives numbers as inputs, "combines" them by means of simple algebraic operations (sums and products) and calculates the value of a chosen function, again returning a number. Since this process cannot be inverted, we can say that a neuron is a numeric, feedforward, information-processing device [8, 9].

4. – Neural networks

Neurons can be put together to form a structure whose behavior is determined by the "collective" behavior of the units. These structures are called neural networks.

4˙1. *Vector state*. – To describe the structure of a n-neuron network a vector state

$$\mathbf{S}(t) = (s_1(t), s_2(t)...s_i(t)...s_n(t)) \tag{11}$$

can be defined, that is the collection of the single neurons states. The state of a network is not fixed: it evolves with time, as single neuronal states evolve. A neuron "evolves" when, provided with activation at time t, $P_i(t)$, it calculates its output, updating its state, $s_i(t) = f(P_i(t))$. As an example, for binary neurons the updating rule can be the following:

$$f(P_i) = \begin{cases} 1, & \text{if } \sum_j w_{ji} s_j + b_i \geq 0; \\ 0, & \text{if } \sum_j w_{ji} s_j + b_i < 0. \end{cases} \tag{12}$$

Updating of neuron states can be synchronous or asynchronous. It is synchronous when neurons update their state simultaneously; it is asynchronous when neurons update their state in sequence, one after the other.

4˙2. *Architecture*. – To describe the architecture of a net it is not sufficient to specify the values of the weights: we must also specify the direction of their links. From this point of view, neurons may create very different structures that can be divided into feedback and feedforward. Individually, neurons remain feedforward devices (allowing information to flow in only one direction) but feedforward nets (allowing information to flow only from input units to output units) as well as feedback nets (allowing information to flow "bidirectionally") can be created.

Feedback nets. Feedback nets can be divided into fully-connected and recurrent. In fully-connected nets every neuron is connected to all the others and links are bidirectional.

In recurrent nets interconnections are directed cycles. From graph theory a cycle is a simple, closed path, that is a sequence of nodes and edges (connecting each node to the next one in the sequence), with no repetition of any edge or node and where the

initial and final edges coincide. It is directed when all the edges in the sequence are directed the same way. A neuron in recurrent nets can be or cannot be linked to all the other neurons. For example, some recurrent networks have two layers where neurons are arranged: neurons in layers are not interconnected, but every neuron is connected to all neurons in the next one. These layers are called "input layer" (where all neurons are input units) and "output layer" (where all neurons are output units).

Actually we can individuate a third kind of feedback nets, resonance networks. As an example consider two layers of units, where neurons in each layer are not interconnected but every neuron in the first one is connected to every neuron in the second one by means of bidirectional links: we cannot find directed cycles, but information is not forced to flow unidirectionally.

Feedforward nets. In feedforward nets all links are directed unidirectionally and directed cycles do not exist. Constitutive units of these nets are layers: an input layer, an output layer and one or more hidden layers (slabs not in direct contact with external world).

4˙3. *Learning rules*. – To solve a particular problem a rule to set weights must be also chosen. The phase of setting weights is called training; after training, a testing phase follows. Initially the net is fed with a sequence of appropriately chosen vectors (patterns) and weights are set by means of some rule (training). Afterwards, the net is fed with other vectors (different ones from the learning phase) and the output of the net is controlled (testing). Learning can be supervised, when the net is shown both input vectors and output vectors or unsupervised, when the net is shown only input vectors.

Hebb rule. Setting of weights, in the training phase, is an iterative procedure whose generic step can be written as

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}. \tag{13}$$

A learning rule prescribes how to compute the term Δw_{ij}. Hebb rule (for bipolar neurons) prescribes that

$$\Delta w_{ij} = \eta(s_i(t)s_j(t)). \tag{14}$$

This rule imitates biology: a connection is strengthened when coupled neurons are "active" or "not active" simultaneously. Hebb rule for binary units is

$$\Delta w_{ij} = \eta(2s_i(t) - 1)(2s_j(t) - 1). \tag{15}$$

Delta rule. Delta rule, instead, prescribes that a sort of distance E between the output computed by the network or by a layer of neurons (the vector $\mathbf{o}_p$) and the desired output (the vector $\mathbf{t}_p$) must be minimized and then used to update weights. Delta rule is very general and its prescription can be adapted to a number of different situations even if it is used extensively with feedforward nets. Here, only an example is mentioned:

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}\,, \tag{16}$$

where

$$E = \sum_p E_p = \sum_p \sum_j \frac{1}{2}(o_j^p - t_j^p)^2. \tag{17}$$

Index p labels vectors in the training set. Suppose we have a two-layer, feedforward net and consider the j-th output component (the activation of j-th output neuron, after p-th vector from training set is provided, is P_j^p):

$$o_j^p = f(P_j^p) = f\left(\sum_i w_{ij} s_i^p\right). \tag{18}$$

So

$$\begin{aligned}
\Delta w_{ij} &= -\eta \frac{\partial E}{\partial w_{ij}} \\
&= -\eta \sum_p \frac{\partial E_p}{\partial w_{ij}} \\
&= -\eta \sum_p \frac{\partial E_p}{\partial o_j^p} \ \frac{\partial o_j^p}{\partial P_j^p} \ \frac{\partial P_j^p}{\partial w_{ij}} \\
&= -\eta \sum_p (o_j^p - t_j^p) \ f'(P_j^p) \ s_i^p.
\end{aligned}$$

Competitive rule. This rule is based on competition among neurons. Weights are updated by

$$\Delta w_{ij} = \eta ||\mathbf{s}_{(j)}^T - \mathbf{W}_i|| \,, \tag{19}$$

where $\mathbf{W}_i$ is the i-th column of the matrix. In nets trained with this rule neurons "compete": the "winner" gains the possibility to update its weights vector to become more alike that specific input pattern.

In all these examples η is a coefficient called learning rate; it can be updated as learning procedure advances. In addition fixed-weight networks must be mentioned, where no training procedure is used: analysis is used to set their weights [7].

5. – Examples of networks

A typical use of neural networks are their mapping, either as pattern classification or pattern association: a vector is presented as input to the network and an output is expected. In these cases networks are called associative memories. We distinguish two kinds of memories. Autoassociative memories are used to associate an input with itself (to correct noisy or incomplete vectors, for example): the input vector and the corresponding output vector coincide. Eteroassociative nets are used, on the contrary, to associate different vectors.

Associative memories work in a such a way that, if $\mathbf{x}$ is associated to $\mathbf{y}$, then the basin of attraction of vector $\mathbf{x}$, $\mathrm{B}(\mathbf{x})$ (that is the maximum number of vectors "similar"

to $\mathbf{x}$ that, once stored, are correctly recalled), is also associated to $\mathbf{y}$. Great attention must be paid in choosing the learning rule for memories, according to the correlation of vectors to be stored: the basin of these vectors depends on it. Examples concerning specific architectures will be given.

5˙1. *Linear associators.* – They are two-layer, feedforward nets with a n-neuron input layer and a k-neuron output layer: the activity function of all neurons is the linear one. If m input patterns are to be associated with m output patterns we can arrange them into matrix form,

$$\mathbf{X} = \{x_{ij}\}_{m\times n}, \quad \mathbf{Y} = \{y_{ij}\}_{m\times k}, \quad \mathbf{W} = \{w_{ij}\}_{n\times k}$$

and $\mathbf{XW} = \mathbf{Y}$. Now, if $m = n$ and input patterns are linearly independent, for a linear eteroassociator $\mathbf{W} = \mathbf{X}^{-1}\mathbf{Y}$. For a linear autoassociator, the same reasoning leads to $\mathbf{W} = \mathbf{X}^{-1}\mathbf{X} = \mathbf{I}$. An autoassociator, however, should even correct wrong patterns and not only return them. For this reason the Hebb rule can be used to find $\mathbf{W} = \mathbf{X}^T\mathbf{X}$ and for eteroassociators, when the $\mathbf{X}$ matrix is not invertible, $\mathbf{W} = \mathbf{X}^T\mathbf{Y}$. The Hebb rule, however, works better when input vectors are pairwise orthogonal or very nearly so. When this is not the case $\mathbf{X}^+$, the pseudo-inverse matrix, can be used to find $\mathbf{W} = \mathbf{X}^+\mathbf{Y}$ or $\mathbf{W} = \mathbf{X}^+\mathbf{X}$ [10-13].

5˙2. *Non-linear associators.* – Hypotheses for the linear and the non-linear associators are the same. The only difference is in the non-linear activity function, say sign function, of neurons: our goal is to verify that

$$\operatorname{sgn}(\mathbf{XW}) = \mathbf{Y}, \tag{20}$$

where the sign function acts over each component of the vector $\mathbf{XW}$. If we are given m pairs of vectors to be associated, matrix $\mathbf{W}$ can be built by means of the Hebb rule

$$\mathbf{W} = \mathbf{W}^1 + \mathbf{W}^2 + ... + \mathbf{W}^m, \tag{21}$$

where $\mathbf{W}^p = \{x_i^p y_j^p\}$ is the matrix we should use if only the p-th vector pair were to be associated. So

$$\mathbf{x}_p\mathbf{W} = \mathbf{y}_p(\mathbf{x}_p \cdot \mathbf{x}_p) + \sum_{l\neq p} \mathbf{y}_l(\mathbf{x}_l \cdot \mathbf{x}_p). \tag{22}$$

Is that true that $\operatorname{sgn}(\mathbf{x}_p\mathbf{W}) = \mathbf{y}_p$? It is, if the crosstalk term $\sum_{l\neq p} \mathbf{y}_l(\mathbf{x}_l \cdot \mathbf{x}_p)$ is negligible, that is input vectors are pairwise orthogonal or very nearly so. For two-layer, feedforward, autoassociative nets the weight-matrix is $\mathbf{W} = \mathbf{X}^T\mathbf{X} = \mathbf{x}_1^T\mathbf{x}_1 + ... + \mathbf{x}_m^T\mathbf{x}_m$ and $\operatorname{sgn}(\mathbf{XW}) = \operatorname{sgn}(\mathbf{XX}^T\mathbf{X})$. Is that true that $\operatorname{sgn}(\mathbf{XX}^T\mathbf{X}) = \mathbf{X}$? It is, if input vectors are pairwise orthogonal or very nearly so.

5˙3. *Perceptron.* – A perceptron is a feedforward net used for pattern classification. It has an input layer and a single output unit. The learning rule prescribes that $\Delta w_i = \alpha t s_i$, where t is the required output value. Weights are changed only when the output value, o, is different from the expected one, t: this learning rule is iterative, ending when weights change no more. Correct weights are reached in a finite number of steps: if $\mathbf{x}_p$ are

the vectors in the training set, the number of required iterations is $k \leq \frac{M||\mathbf{w}||^2}{m^2}$, where $M = \max\{||\mathbf{x}_p||^2\}$ and $m = \min\{\mathbf{x}_p \cdot \mathbf{w}\}$. There exist perceptrons returning a vector; in that case $\Delta w_{ij} = \alpha t_j s_i$ [10, 14].

5˙4. *BAM*. – This is a two-layer, resonance net being used for association problems where information flows bidirectionally: to realize the correct association, the output vector is fed back to the input units a number of times (the index i labels a "crossing" of the network):

$$\begin{cases} \mathbf{y}_{(i)} = \mathrm{sgn}(\mathbf{x}_{(i)}\mathbf{W}), \\ \mathbf{x}^T_{(i+1)} = \mathrm{sgn}(\mathbf{W}\mathbf{y}^T_{(i)}); \end{cases} \tag{23}$$

a fixed point of the net is a pair of vectors $(\mathbf{x}, \mathbf{y})$ whose values become independent of the number of iterations:

$$\begin{cases} \mathbf{y} = \mathrm{sgn}(\mathbf{x}\mathbf{W}), \\ \mathbf{x}^T = \mathrm{sgn}(\mathbf{W}\mathbf{y}^T). \end{cases} \tag{24}$$

By means of the Hebb rule, if $\mathbf{W} = \mathbf{x}_1^T\mathbf{y}_1 + \mathbf{x}_2^T\mathbf{y}_2 + ... + \mathbf{x}_m^T\mathbf{y}_m$ a fixed point is found for eteroassociative nets; if $\mathbf{W} = \mathbf{x}_1^T\mathbf{x}_1 + \mathbf{x}_2^T\mathbf{x}_2 + ... + \mathbf{x}_m^T\mathbf{x}_m$ a fixed point is found for autoassociative nets. Evolution of BAM can also be analyzed introducing an energy function

$$E = -\frac{1}{2}\mathbf{x}_{(i+1)}\mathbf{W}\mathbf{y}^T_{(i)}. \tag{25}$$

We can rewrite the energy this way $E = -(1/2)\sum_j P_j x_j$, where P_j is the product of the j-th line of $\mathbf{W}$ with the column vector $\mathbf{y}^T$, that is the activation of j-th unit in the input layer. Suppose that j-th input unit changes its state: if we calculate the value of E with the unit updated and subtract it from the value of E without the unit updated, we have

$$E(x_{j;\,(i+1)}, \mathbf{y}_{(i)}) - E(x'_{j;\,(i+1)}, \mathbf{y}_{(i)}) = -\frac{1}{2}P_j(x_{j;\,(i+1)} - x'_{j;\,(i+1)}) > 0. \tag{26}$$

As an example, when a bipolar neuron with zero threshold is updated, $x_{j;\,(i+1)}$ and $-x'_{j;\,(i+1)}$ have the same sign and it differs from the sign of P_j: that is why the last inequality holds. The same result holds if we update the j-th output unit or performing a synchronous updating. BAM always reaches a state of (local) minimum energy (the number of states "visited" in the process is finite) whose value is no more changed by the updating process: the fixed point computed by means of $\mathbf{W}$ is reached.

5˙5. *Discrete Hopfield networks*. – Hopfield nets are fully-connected nets of bivalent (bipolar o binary) neurons, whose activation is of the form $P_i = \sum_j w_{ji}s_j - \theta_j$. Weights are symmetric and no loops are present ($w_{ij} = w_{ji}$, $w_{ii} = 0$). The net is trained by the Hebb rule and it is evolved by asynchronous updating. An energy function can be introduced:

$$E = -\frac{1}{2}\sum_{ij} s_i s_j w_{ij} + \sum_i s_i\theta_i. \tag{27}$$

Hopfield networks "evolve" by means of asynchronous updating exploring the configurations space of the net, that is a discrete space where each state is represented as a 2^n bits word, n being the number of neurons in the net. As the net evolves

$$\mathbf{S} = (s_1, s_2 ... s_k ... s_n) \longrightarrow \mathbf{S}' = (s_1, s_2 ... s'_k ... s_n).$$

Let us calculate

$$E(\mathbf{S}) - E(\mathbf{S}') = \Big(- \sum_j s_k s_j w_{kj} + s_k \theta_k \Big) - \Big(- \sum_j s'_k s_j w_{kj} + s'_k \theta_k \Big). \tag{28}$$

Hypotheses of symmetric weights and absence of loops allow us to re-write this expression in the following way:

$$\begin{aligned} E(\mathbf{S}) - E(\mathbf{S}') &= -(s_k - s'_k) \sum_j s_j w_{kj} + \theta_k (s_k - s'_k) \\ &= -(s_k - s'_k) \Big(\sum_j s_j w_{kj} - \theta_k \Big). \end{aligned}$$

From this there follows, for binary (bipolar) neurons

$$s_k = 1 \longrightarrow s'_k = 0\,(-1) \Longrightarrow s_k - s'_k > 0, \ \left(\sum_j s_j w_{kj} - \theta_k \right) < 0, \tag{29}$$

$$s_k = 0\,(-1) \longrightarrow s'_k = 1 \Longrightarrow s_k - s'_k < 0, \ \left(\sum_j s_j w_{kj} - \theta_k \right) > 0. \tag{30}$$

In both cases $E(\mathbf{S}) - E(\mathbf{S}') > 0$. Nevertheless, without the conditions of symmetric weights and absence of loops, convergence of Hopfield nets is no more guaranteed: oscillations or limit cycles can be encountered. Evolution of Hopfield nets guarantees that a state with a (local) minimum energy exists and the net converges to it. As an example, to store m orthogonal vectors (or very nearly so) of bipolar data in Hopfield nets used as autossociative memories, we can choose $\mathbf{W} = \big(\mathbf{x}_1^T \mathbf{x}_1 - \mathbf{I}\big) + \big(\mathbf{x}_2^T \mathbf{x}_2 - \mathbf{I}\big) + ... + \big(\mathbf{x}_m^T \mathbf{x}_m - \mathbf{I}\big)$; the energy hypersurface will have local minima in correspondence of these stored vectors, since $E(\mathbf{S}) = \sum_i -(1/2)||\mathbf{S}\mathbf{x}_i^T||^2 + \frac{mn}{2}$. If vectors to be stored are in general correlated, then the capacity of Hopfield nets is $m = 0.15n$, for binary patterns, and $m = n/(2 \log_2 n)$ for bipolar patterns [15].

5˙6. *Continue Hopfield nets.* – Continuity is introduced at three different levels: activity functions are continuous in activation, activation is continuous in time and updating is continuous. So a new relation holds,

$$\frac{\mathrm{d}P_i(t)}{\mathrm{d}t} = \eta \left(- P_i(t) + \sum_j w_{ji} s_j(t) \right) \Longrightarrow P_i(t) + \frac{1}{\eta} \frac{\mathrm{d}P_i(t)}{\mathrm{d}t} = \sum_j w_{ji} s_j(t), \tag{31}$$

where, for example, $s_i(t) = \tanh(P_i(t))$. Energy can be defined as

$$E = -\frac{1}{2}\sum_{ij} s_i s_j w_{ij} + \sum_i \int_0^{s_i} f^{-1}(s)\mathrm{d}s. \tag{32}$$

Asynchronously updating continuous Hopfield nets (with symmetric weights and no loops) means computing

$$\begin{aligned}\frac{\mathrm{d}E}{\mathrm{d}t} &= -\sum_{ij}\frac{\mathrm{d}s_i}{\mathrm{d}t}s_j w_{ij} + \sum_i f^{-1}(s_i)\frac{\mathrm{d}s_i}{\mathrm{d}t} = -\sum_i \frac{\mathrm{d}s_i}{\mathrm{d}t}\left(\sum_j s_j w_{ij} - P_i\right)\\ &= -\frac{1}{\eta}\sum_i \frac{\mathrm{d}s_i}{\mathrm{d}t}\ \frac{\mathrm{d}P_i}{\mathrm{d}t} = -\frac{1}{\eta}\sum_i f^{'}(P_i)\Big(\frac{\mathrm{d}P_i}{\mathrm{d}t}\Big)^2.\end{aligned}$$

$\mathrm{d}E/\mathrm{d}t \leq 0$ follows since the activity function is monotonically increasing ($f^{'}(P_i) > 0$): a stable state for which $\mathrm{d}E/\mathrm{d}t = 0$ is reached when $\mathrm{d}P_i/\mathrm{d}t = 0$. In Hopfield nets all neurons are visible; this hypothesis can be abandoned as we will see.

5˙7. *Stochastic networks*. – Networks are called stochastic when probability is introduced at some level. Here, two examples are given.

Simulated annealing. This is a technique to allow an Hopfield net to escape from local minima of energy and to reach a global minimum. Evolution of an Hopfield net is seen as a trajectory in configurations space: a state whose energy is the lowest among those visited is reached. A greater portion of space can be visited allowing the energy to increase. Annealing prescribes that the net is left "free" to evolve by random changes of its actual state: energy changes in turn randomly, also increasing. Simulating annealing prescribes probabilities of accepting a change:

$$\begin{cases} \Delta E \leq 0, & p = 1;\\ \Delta E > 0, & p = \dfrac{1}{1+\exp[\Delta E/T]}. \end{cases} \tag{33}$$

Since simulated annealing starts at high temperatures so, in the limit of $T \longrightarrow +\infty$, probability that energy increases $p \longrightarrow 1/2$; as $T \longrightarrow 0$, $p \longrightarrow 0$. Evolving the net by annealing means warming it up, first, and then cooling it down slowly [16-19].

Boltzmann machines. These nets are stochastic discrete Hopfield networks. Updating is asynchronous and states (for binary neurons, say) evolve with a probability proportional to the activation of units:

$$s_i(t+1) = \begin{cases} 1, & p_i = \frac{1}{1+\exp[-(\sum_j w_{ji}s_j - \theta_i)/T]};\\ 0, & 1-p_i. \end{cases} \tag{34}$$

Note that, despite the sign of the activation, neuron can "flip" both ways and convergence to a minimum-energy state is no more guaranteed. Boltzmann machines reduce to

deterministic Hopfield nets when $T \longrightarrow 0 : p_i \longrightarrow 1$ if $P_i > 0$ and $p_i \longrightarrow 0$ if $P_i < 0$:

$$p_i = \frac{1}{1 + \exp\Big[-\big(\sum_j w_{ji} s_j - \theta_i\big)/T\Big]} \longrightarrow \Theta(P_i). \tag{35}$$

When $T > 0$ Boltzmann nets do not reach a stable state: it keeps "flipping" as shown by the $2^n \times 2^n$ transition matrix (n is the number of neurons), whose element p_{ij} (p_{ii}) represents the probability that the net goes from state i to state j (remains in current state)

$$\{p_{ij}\} = \begin{pmatrix} 1 - \sum_{i=2}^{m} \frac{1}{1+\exp[(E_i - E_1)/T]} & \cdots & \frac{1}{1+\exp[(E_m - E_1)/T]} \\ \vdots & \ddots & \vdots \\ \frac{1}{1+\exp[(E_1 - E_m)/T]} & \cdots & 1 - \sum_{i=2}^{m} \frac{1}{1+\exp[(E_i - E_m)/T]} \end{pmatrix}. \tag{36}$$

To understand the meaning of this matrix consider its equilibrium eigenvector (with eigenvalue 1) ($\frac{\exp[-E_1/T]}{Z}, \frac{\exp[-E_2/T]}{Z}, ..., \frac{\exp[-E_m/T]}{Z}$). Therefore, for a given T we do not reach a stable state but a stable probability distribution of states. The state of the net will continue to change from the initial one but as time goes on, each state will be encountered a number of times more and more close to the value given by the Boltzmann distribution [17, 20, 1].

Boltzmann machines cannot "memorize" specific vectors but can learn to reproduce probability distributions. After specifying visible units (input and output ones, labelled with index α) and hidden units (labelled with index β) let us write the probability distribution of visible units $P_\alpha = \sum_\beta P_{\alpha\beta} = \frac{1}{Z} \sum_\beta \exp[E_{\alpha\beta}/T]$, where $Z = \sum_{\alpha\beta} \exp[E_{\alpha\beta}/T]$ and $E_{\alpha\beta} = -\frac{1}{2} \sum_{ij} w_{ij} s_i^{\alpha\beta} s_j^{\alpha\beta}$. By means of delta rule let us minimize the "distance" between P_α and the distribution we want the net to learn, P'_α, that is Shannon's cross entropy [21, 22, 6]

$$D = \sum_\alpha P'_\alpha \log \frac{P'_\alpha}{P_\alpha}. \tag{37}$$

Then, as usual,

$$\Delta w_{ij} = \\ = -\eta \frac{\partial D}{\partial w_{ij}} = \eta \sum_\alpha \frac{P'_\alpha}{P_\alpha} \frac{\partial P_\alpha}{\partial w_{ij}} = \frac{\eta}{T} \Big(\sum_\alpha \frac{P'_\alpha}{P_\alpha} \sum_\beta s_i^{\alpha\beta} s_j^{\alpha\beta} P_{\alpha\beta} - \sum_\alpha P'_\alpha \langle s_i s_j \rangle_{\text{free}} \Big).$$

Let us we write $P_{\alpha\beta} = P_{\beta|\alpha} P_\alpha$ and $\langle s_i s_j \rangle_{\text{fixed}} = \sum_{\alpha,\,\beta} P'_\alpha P_{\beta|\alpha}\, s_i^{\alpha\beta} s_j^{\alpha\beta}$ (that is the expectation value computed with a conditional probability density, where $\langle s_i s_j \rangle_{\text{free}}$ was computed with the usual probability density), then

$$\Delta w_{ij} = \frac{\eta}{T} \big(\langle s_i s_j \rangle_{\text{fixed}} - \langle s_i s_j \rangle_{\text{free}} \big), \tag{38}$$

returning an expression very similar to the Hebb rule. Subscript "fixed" ("free") indicates a net where input units are (not) clamped by means of P'_α. During the fixed phase, visible

units are clamped to the value of a pattern and the network is allowed to reach a condition of low-T equilibrium by annealing; then we compute $\langle s_i s_j \rangle_{\text{fixed}}$. This phase is repeated a number of times, with each pattern begin clamped with a frequency corresponding to the P'_α. In free phase, the network is let running freely. Once the low-T equilibrium is reached we compute $\langle s_i s_j \rangle_{\text{free}}$. The two phases are alternated. At the end of the learning procedure for a single weight, we sum the two terms. This procedure must be followed a number of times corresponding to the frequency of patterns we want the net to reproduce [5, 6, 23].

6. – Applications

Neural networks were born to build more efficient computational devices, imitating the characteristics of the nervous system; they can be used in a number of different ways: here, only three examples concerning pattern recognition and pattern classification are quoted. This finds application in medicine (diagnosis and screening), finance, engineering (product inspection, signature verification), security (fingerprints verification), etc.

6˙1. *Speech recognition.* – In speech recognition problems the net has to cluster similar inputs (fragments of speech) returning a phoneme coded as a two-dimensional array. This can find application for vocal typewriters or for the design of machines to help people with problems of pronunciation.

First, the waveform of the speech to be analyzed is digitalized and converted into a spectral representation.

Second, the spectral domain of the speech is sampled at intervals of time. For each fragment of sampled speech we collect the so-called "features", that is numbers identifying at best the portion of speech as energy, spectral change, etc. (the number of them can vary depending on circumstances). Acoustic content is also analyzed comparing features of different portions of speech, to account for the global dynamic of the speech.

Third, we send the collected features for each fragment of speech to a neural network: the output of the net (usually a multi-layer perceptron) is a classification of each input in terms of phoneme-based categories. One could expect one category for each phoneme; nevertheless phonemes have a great influence on neighboring ones and it is preferred to split each phoneme in a number of "parts" depending on vocabulary and phonetics of that particular language. Each part of a phoneme is not classified according to all possible combinations with other phoneme-parts but according to one of eight bigger categories (to better account for neighboring phonemes pronounce) as "front vowel", "fricative", etc. The outputs of the neural network (whose number is the same of the categories of phonemes) are used to estimate the probability of a category: neural networks can classify in this way, if they are given enough training data and hidden nodes.

Sending inputs to the net continuously, it is possible to build a matrix where the evolution of the phoneme classification *vs.* time is shown.

Fourth, a technique of searching is used to match the neural-network outputs with the words that are assumed to be in the fragment of speech, by 1) expanding and ordering their pronunciations into strings of phonetic-based categories and 2) moving inside the evolution matrix, changing category with time if the probability of the new category is greater than the probability of the current one. At the end of the search, we have a path through the categories: from it we can easily determine the corresponding word [8,24,25].

6˙2. *Handwriting recognition*. – Handwriting recognition is not only one of the most important subprocesses in the analysis of an image containing text, it is also the starting point for a new kind of easy-to-use technology, whose direct application would start with a new generation of palm-pilots.

Firstly, an image is pre-processed: specific algorithms detect where text exists within the scanned image, removing noise and extraneous strokes as symbols, drawings, etc.

The text is then subjected to the "segmentation" phase. For word, two strategies exist: the analytical approach attempts to recognize every single component characters, guessing the boundaries of characters by means of variations in the pen-stroke; the top-down approach attempts to extract features directly from the word as a whole. In recognizing words, segmentation phase define the primitive strokes and the way they may be combined to form "segments" that is the possible, future characters: at the end of the phase a series of segments is obtained.

In the next step, the classification phase, each segment is evaluated using a neural network, whose output is a vector of letter-class probabilities (as in the speech recognition procedure): the classifier is a multi-layer perceptron trained with the back-propagation (a sort of delta rule). It is worth to note that the grey-scale input representation is the best for the performance of the net, even if solutions in which independent classifiers, fed with different input representations, are compared. So the architecture of the classifier may consist in multiple inputs, a hidden layer (separate for each input representation), fully connected hidden layers (again separate for each representation) and a shared, fully-connected output layer.

Better performances are obtained negative-training the net, that is training the net to classify invalid segments (it is seen that multi-stroke characters are better represented) or feeding the net with random-variation of data (to account for calligraphy). The output of the classifier is an ensemble of probability vectors, one vector for each segment: only few of its components are passed to the search engine, which then looks for a minimum-cost path through this ensemble trying to guess the most likely word, on the basis of the dictionary "memorized" by the machine.

A general consideration about this procedure is the following: letters have to be "disconnected", that is overlap between characters must be limited: occasional connections between characters are the largest class of errors [8, 24-26].

Image recognition and classification. The same approach can be followed in image recognition as well as in image classification. In this case, the starting point is "to pixel" the image and to normalize each pixel into a grey-scale color value in the range [0,1]: assuming RGB pixels, where each color-component has values in the range [0, 255] (mapping pixels to ASCII code characters) this can be done computing the pixel-input to the net (one pixel, one input neuron) as $\left[1-\left(\frac{\mathrm{R}+\mathrm{G}+\mathrm{B}}{3}\right)\frac{1}{255}\right]$; "0" value means "white pixel" and "1" value means "black pixel". The architecture of the net is the same as before; the number of hidden neurons can vary to find correlations in the input data: as an example, information about shape simply providing information about color. Output neurons can return a value between 0 and 1 (using, for example a sigmoid-shaped activity function) to reduce the number of output neurons needed to express a particular result, in terms of binary coding.

For image classification a backpropagation, multi-layer perceptron can be used. Phases of "pixeling" and color-normalizing are the same as those for image recognition. By means

of a great number of hidden neurons, nets can be fed with only color information, as in a recent study for the classification of different vegetal species: in a similar way, a greater capability in grading colors can be reached, to improve the classification procedure to distinguish similar individuals belonging to the same species [27, 26].

REFERENCES

[1] Turchetti G., in *Dinamica classica dei sistemi fisici* (Zanichelli, Bologna) 2002.
[2] Weisburg G., in *Complex Systems Dynamic* (Addison-Wesley, Reading) 1991.
[3] Amendolia S. R. and Bottigli U., *Tutorial on neural networks*, unpublished.
[4] Rosenblatt F., *Psychol. Rev.*, **65** (1958) 386.
[5] Abdi H., Valentin D. and Edelman B., in *Neural Networks* (Sage Publications, Inc.) 1999.
[6] Mackay D. J. C., in *Information Theory, Inference and Learning Algorithms* (Cambridge University Press) 2003.
[7] Duch W. and Jankowski N., *Neural Computing Surveys*, **2** (1999) 163.
[8] Bishop G. M., in *Neural Networks for Pattern Recognition* (Oxford University Press) 1995.
[9] Specht D. F., *Neural Networks*, **3** (1990) 109.
[10] Abdi H., *J. Biol. Syst.*, **2** (1994) 247.
[11] Abdi H., Valentin D., Edelman B. and O'Toole A. J., *J. Math. Psychol.*, **40** (1996) 175.
[12] Ackley D. H. and Hinton G. E., *Cognitive Science*, **9** (1985) 147.
[13] Block H. D., *Rev. Mod. Phys.*, **34** (1962) 123.
[14] Widrow B. and Lehr M. A., *Proc. IEEE*, **78** (1990) 1415.
[15] Kanter I. and Sompolinsky H., *Phys. Rev. A*, **35** (1987) 380.
[16] Greiner W., Neise L. and Stoker H., in *Thermodynamics and Statistical Mechanics* (Springer-Verlag New York, Inc.) 1995.
[17] Grimmet G. R. and Stirzaker D. R., in *Probability and Random Processes* (Oxford University Press) 1982.
[18] Gurney R. W., in *Introduction to Statistical Mechanics* (Mcgraw Book Company, Inc.) 1949.
[19] Szu H. H. and Maren A. J., *Phys. Lett. A*, **122** (1987) 157.
[20] Kirkpatrick S., Gelatt C. D. and Vecchi M. P., *Science*, **220** (1983) 671.
[21] Cover T. M. and Thomas J. A., in *Elements of Information Theory* (John Wiley & Sons, Inc., New York) 1991.
[22] Jaynes E. T., *Phys. Rev.*, **106** (1957) 620.
[23] Sejnowsky T. J., in *Neural Networks for Computing*, Snowbird, Utah, 1986 (American Institute of Physics) 1986.
[24] Ripley B. D., in *Pattern Recognition and Neural Networks* (Cambridge University Press) 1996.
[25] Vapnik V. N., in *Pattern Recognition and Neural Networks* (Wiley, New York) 1999.
[26] Zeng Q., Milthorpe B. K. and Jones A. S., in *Cytometry,* Part A, **57**A (2004) 1-9.
[27] Yang C. C., Prasher S. O., Landry J. A., Ramaswamy H. S. and Ditommaso A., in *Can. Agric. Eng.*, **42** (2000) 3.