Multilayer neural networks: learning models and applications

Sergio Gómez Jiménez

Multilayer neural networks: learning models and applications

Xarxes neuronals multicapa: models d'aprenentatge i aplicacions

Memòria de la Tesi presentada
per en Sergio Gómez Jiménez
per a optar al grau de
Doctor en Ciències Físiques

Departament d'Estructura i Constituents de la Matèria
Universitat de Barcelona
Juliol de 1994

Emili Elizalde i Rius, i *Lluis Garrido i Beltran*, professors titulars del Departament d'Estructura i Constituents de la Matèria de la Universitat de Barcelona,

Certifiquen: que la present memòria, que porta per títol *Multilayer neural networks: learning models and applications*, ha estat realitzada sota la seva direcció i constitueix la Tesi d'en *Sergio Gómez Jiménez* per a optar al grau de Doctor en Ciències Físiques.

Emili Elizalde i Rius   Lluis Garrido i Beltran

A mis padres

A Esther

Acknowledgements

I would like to thank all the people that, during the last four years, have made possible the realization of this work.

Contents

List of Figures

List of Tables

Resum

Els models de *xarxes neuronals artificials* es van introduir per tal d'explicar el funcionament del sistema nerviós dels éssers vius i, en particular, el del cervell humà. Des de finals del segle XIX es sabia que aquestes estructures eren bàsicament enormes col·leccions d'un tipus de cèl·lula, anomenades *neurones*, altament connectades entre sí. Per tant, el coneixement del funcionament d'una sola neurona potser permetria aclarir en part el seu comportament col·lectiu.

L'any 1943 McCulloch i Pitts van proposar el primer model de xarxa neuronal artificial (veure Secc. 1.1). Basant-se en els coneixements existents sobre la morfologia i la fisiologia de les neurones, van definir una *neurona artificial* (també anomenada *unitat*) com a un element de processament l'estat (o *activació*) del qual només pot prendre dos valors, zero o u, i que està connectat amb moltes altres neurones de dues maneres diferents: rebent com a senyal d'entrada l'estat de l'altre unitat multiplicat per un cert pes, o mostrant-li el seu estat a l'altre neurona (senyal de sortida). Si la suma dels senyals d'entrada supera un cert llindar, la neurona pren com a estat el valor u, i en cas contrari l'activació és zero (és a dir, la *funció d'activació* és la funció esglaó). Aquesta dependència entre les activacions individuals de les diferents neurones artificials és la que fa que la xarxa evolucioni, posant de manifest el seu comportament col·lectiu.

Una de les característiques més interessants de les xarxes amb unitats completament interconnectades és la possibilitat d'emmagatzemar patrons en forma de *memòria associativa*: escollint adequadament els pesos i els llindars, la dinàmica fa que la xarxa tendeixi a assolir una configuració estable que correspon al patró emmagatzemat més semblant a l'estat inicial de la xarxa. Existeixen diverses tècniques pel càlcul d'aquests paràmetres com, per exemple, la *regla de Hebb*, la *regla de la pseudo-inversa* o l'algorisme *AdaTron* (veure Secc. 2.1). No obstant, totes elles presenten una sèrie de problemes que dificulten la seva operativitat: la seva capacitat és limitada, existeixen estats estables que no corresponen a cap patró emmagatzemat, i hi poden haver estats oscil·lants dels quals la dinàmica no es pot escapar.

A Secc. 2.2 exposem tres solucions diferents al problema de la memòria associativa que no pateixen dels problemes anteriors, totes elles basades en l'ús de *xarxes neuronals multicapa*. En aquest tipus de xarxes les unitats s'agrupen en capes i, habitualment, només hi ha connexions entre capes consecutives. A més,

dues capes tenen funcions especials: la d'entrada, on la xarxa llegeix els patrons d'entrada, i la de sortida, on la xarxa col·loca el resultat de processar el patró d'entrada (veure Secc. 1.3). Les nostres solucions són òptimes en el sentit que sempre recuperen el patró emmagatzemat més proper a qualsevol patró d'entrada que li introduïm. Això ho aconseguim dividint el problema en tres fases: càlcul de les distàncies entre el patró d'entrada i els emmagatzemats, identificació del patró que està a distància mínima, i recuperació d'aquest patró.

Malgrat l'indubtable interès dels sistemes de memòria associativa, que poden servir per a explicar el funcionament de la memòria en els éssers vius, la característica que ha atret més l'atenció últimament és la possibilitat de fer-les anar com a mecanismes d'entrada-sortida que aprenen a partir d'exemples. És a dir, donat un conjunt de parelles formades cadascuna per un patró d'entrada i un de sortida, existeixen diverses maneres d'ajustar els pesos i els llindars de la xarxa (normalment multicapa) per tal d'aconseguir que aquesta interpoli prou bé bona part dels exemples: aquest és l'anomenat *aprenentatge supervisat* amb xarxes neuronals.

Un primer problema que hem tractat és la *codificació* i *decodificació* de patrons amb components a dos valors (veure Secc. 3.1). Més concretament, suposem que tenim N patrons d'entrada i els seus corresponents N patrons de sortida, tots ells del tipus mencionat anteriorment i de longitud N, i volem construir una xarxa multicapa que faci l'assignació desitjada. Demanem, a més, que entre les capes d'entrada i sortida existeixi una altra de tamany el més petit possible. Aquest minim número és, bàsicament, $R \sim \log_2 N$, que s'assoleix quan l'estat d'aquesta capa és, per cada patró d'entrada, un número diferent en base dos. Nosaltres demostrem que, per patrons arbitraris, no existeix cap combinació de pesos i de llindars que facin aquesta feina directament, és a dir, sense cap altra capa intermitja apart de la de R unitats. Pel cas particular de patrons unaris, però, sí que hi ha infinites solucions, i nosaltres en donem una d'especialment senzilla.

La introducció de noves capes intermitges obre les portes a moltes altres solucions completament generals, de les quals nosaltres n'expliquem unes quantes. La idea principal és que, si aconseguim transformar els patrons inicials en patrons unaris mitjantçant una xarxa amb només dues capes, llavors podem aprofitar la solució unària anterior com a part de la solució general.

Si a una xarxa d'aquest tipus se li presenta un patró que no es cap dels utilitzats a l'hora de dissenyar-la, pot passar que el senyal d'entrada d'alguna unitat coincideixi amb el seu llindar. La neurona, doncs, no pot decidir si la seva activació ha de ser zero o u. En aquesta mateixa secció donem quatre solucions diferents, i estudiem en detall els dos cassos més interessants. En particular, calculem el nombre de patrons que generen aquestes indecisions per a la xarxa de codificació de patrons unaris, amb i sense soroll tèrmic, i l'*accessibilitat* dels possibles estats de la capa intermitja.

En les solucions que hem proposat pel problema de la codificació, els pesos i els llindars s'han calculat de forma teòrica i s'han proveït fórmules per a calcular-los.

Quan els patrons d'entrada poden prendre valors continus, però, aquest enfocament ja no serveix, havent-se de cercar nous mètodes. Per exemple, la *perceptron learning rule* (veure Secc. 3.2) permet trobar els paràmetres d'un *perceptró simple* (que no és més que una xarxa amb una capa d'entrada i una única unitat de sortida, sense altres neurones intermitges) que aplica correctament tots els patrons a les seves imatges desitjades, sempre i quan aquests patrons siguin linealment separables (com només hi ha una unitat de sortida, i aquesta només pot prendre dos valors, les úniques possibles imatges desitjades són zero o u; és per això que es parla de separar aquests dos tipus de patrons).

Una possible generalització d'aquesta regla consisteix en adaptar-la per a què funcioni amb perceptrons simples que tinguin una unitat de sortida multiestat. Un dels seus avantatges és que permeten tractar problemes de classificació en més de dues classes de forma natural, sense haver d'utilitzar varies unitats per a representar totes les classes. A Secc. 3.3 exposem la nostre solució i donem una demostració de la convergència del mètode. A més, aconseguim definir l'anomenat *perceptró multiestat de màxima estabilitat* i en donem un esquema de la demostració de la seva existència i unicitat.

Quan un problema no és linealment separable, no existeix cap perceptró simple que pugui aprendre tots els patrons alhora. L'alternativa evident són, doncs, les xarxes multicapa. No obstant, l'aprenentatge d'aquestes xarxes és molt més complicat, degut especialment al caràcter discret de les funcions d'activació. Una manera molt elegant de resoldre un problema amb xarxes multicapa consisteix en començar amb una xarxa petita, i llavors anar afegint neurones durant l'aprenentatge fins que, finalment, tots els patrons estiguin correctament classificats. Aquest és precisament el funcionament del *tiling algorithm* i d'altres mètodes relacionats (veure Secc. 3.4). L'avantatge del tiling algorithm respecte alguns altres és que permet la seva utilització amb unitats multiestat. Així hem pogut aprendre problemes multiestat no separables amb certa facilitat.

Una propietat molt remarcable de les xarxes multicapa amb funcions d'activació discretes és el fet que la primera capa intermitja juga un paper diferent a la resta, ja que defineix una partició a l'espai de patrons d'entrada. Tots els patrons que pertanyen a una mateixa unitat d'aquesta partició, quan són introduïts a la xarxa, produeixen la mateixa sortida. Llavors, si es vol que la xarxa neuronal generalitzi correctament, caldrà posar, durant l'aprenentatge, tot l'èmfasi en aconseguir que aquesta partició sigui la millor possible.

Si es substitueixen les funcions d'activació discretes per unes de contínues i diferenciables (per exemple sigmoides, que s'assemblen a la funció esglaó habitual), la xarxa multicapa es converteix en una funció contínua i diferenciable entre l'espai dels patrons d'entrada i els de sortida. Llavors, una manera convenient de fer l'aprenentatge consisteix en definir una funció error entre les sortides de la xarxa i les sortides desitjades, i tractar de minimitzar-la. Un mètode senzill de fer aquesta minimització consisteix en utilitzar el mètode de descens pel màxim pendent. Calculant el gradient de l'error respecte dels pesos i dels llindars s'obté així el conegut mètode de la *back-propagation* (veure Secc. 4.1).

A diferència del tiling algorithm, que sempre acaba aprenent tots els patrons que se li ensenyan (suposant que no hi hagi patrons contradictoris), la back-propagation difícilment redueix l'error fins a zero. Les principals raons són que el mètode del gradient tendeix a trobar mínims de la funció error, però no pot assegurar que els mínims trobats siguin globals, i que freqüentment es treballa amb patrons amb soroll. No obstant, aquesta habilitat per a tractar el soroll juntament amb la facilitat de les xarxes per a aprendre funcions complicades són les principals causes de l'èxit de la back-propagation en tota mena d'aplicacions.

El fet que la back-propagation es basi en la minimització d'una funció d'error fa possible un estudi més teòric sobre quin és aquest mínim, i quina influència té sobre els resultats la manera d'escollir la representació dels patrons de sortida. A Secc. 4.2 trobem el mínim suposant que la xarxa pot aprendre qualsevol funció arbitrària, i demostrem, entre altres coses, que en problemes de classificació l'ús de representacions de sortida binàries és incorrecte ja que impedeixen fer les òptimes decisions Bayesianes. A més, les nostres simulacions mostren una perfecta correspondència amb les prediccions teòriques, posant de manifest la seva validesa.

A Secc. 4.3 apareixen algunes de les aplicacions que hem desenvolupat utilitzant xarxes neuronals. La primera d'elles consisteix en la *reconstrucció d'imatges* a partir de dades amb soroll. El que fem és ensenyar a una xarxa multicapa a treure aquest soroll, utilitzant com a base una imatge reconstruïda per altres mètodes. Un cop fet l'aprenentatge la xarxa ja es pot aplicar a noves imatges, i observem que la nostra reconstrucció té una qualitat molt semblant a l'obtinguda pels altres mètodes, però amb molt menys esforç i molta més velocitat.

Una segona aplicació és la *compressió d'imatges*. Com les imatges digitalitzades solen ocupar molt d'espai en disc, és necessari comprimir-les de manera que ocupin el mínim possible, però habitualment no hi ha prou amb els mètodes de compressió reversible corrents. Llavors, pot ser preferible perdre una mica de qualitat si amb això s'aconsegueixen factors de compressió més elevats. Utilitzant una variant de la back-propagation, anomenada *self-supervised back-propagation*, es poden comprimir imatges d'aquesta manera. No obstant, aquest mètode presenta alguns problemes que afecten a la qualitat de la imatge comprimida. Nosaltres proposem una sèrie de modificacions que milloren sensiblement el mètode, com són la utilització dels valors de les cel·les veines, o la introducció d'un terme repulsiu per tal de minimitzar la pèrdua d'informació entre les capes de la xarxa.

Finalment, hem aplicat *xarxes recurrents* per l'aprenentatge de *sèries temporals*, en particular de la coneguda sèrie de taques solars. Es demostra que aquestes xarxes donen resultats més bons que les multicapa corrents, sobretot per a prediccions a llarg termini. Per a aconseguir-ho, però, cal escollir correctament el tipus de funció d'activació de cada capa, ja que amb les sigmoides no es pot aprendre aquesta tasca.

Chapter 1

Introduction

1.1 From biology to artificial neural networks

The structure of biological nervous systems started to be understood in 1888, when Dr. Santiago Ramón y Cajal succeeded to see the *sysnapses* between individual nervous cells, the *neurons*. This discovery was quite impressive since it proved that all the capabilities of the human brain rest not so much in the complexity of its constituents as in the enormous number of neurons and connections between them. To give an idea of these magnitudes, the usual estimate of the total number of neurons in the human central nervous system is 10^{11}, with an average of 10 000 synapses per neuron. The combination of both numbers yields a total of 10^{15} synaptic connections in a single human brain!

All the neurons share the common structure schematized in Fig. 1.1. There is a *cell body* or *soma* where the *cell nucleus* is located, a tree-like set of fibres called *dendrites* and a single long tubular fibre called the *axon* which arborizes at its end. Neurons establish connections either to sensory organs (input signals), to muscle fibres (output signals) or to other neurons (both input and output). The output junctions are called *synapses*. The interneuron synapses are placed between the axon of a neuron and the soma or the dendrites of the next one.

The way a neuron works is basically the following: a potential difference of chemical nature appears in the dendrites or soma of the neuron, and if its value reaches a certain threshold then an electrical signal is created in the cell body, which immediately propagates through the axon without decaying in intensity. When it reaches its end, this signal is able to induce a new potential difference in the postsynaptic cells, whose answer may or may not be another *firing* of a neuron or a contraction of a muscle fibre, and so on. Of course, a much more detailed overview could be given, but this suffices for our purposes.

In 1943 W.S. McCulloch and W. Pitts [48] proposed a mathematical model for capturing some of the above described characteristics of the brain. First, an *artificial neuron* (or *unit* or simply *neuron*) is defined as a processing element

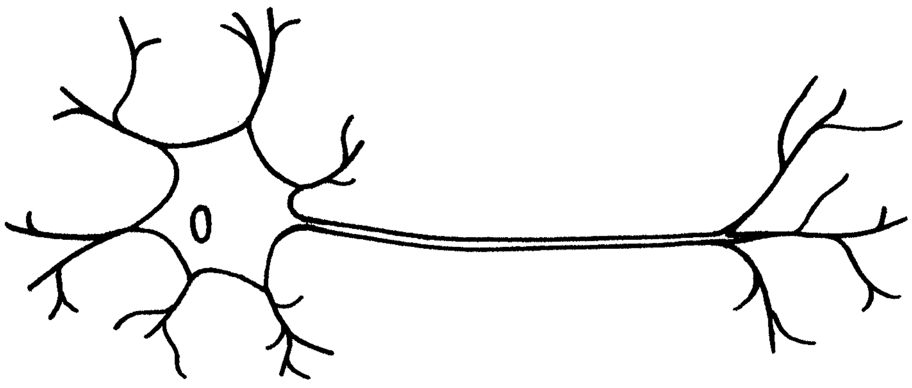

Figure 1.1: Schematic drawing of a neuron.

whose state ξ at time t can take two different values only: $\xi(t) = 1$, if it is firing, or $\xi(t) = 0$, if it is at rest. The state of, say, the i-th unit, $\xi_i(t)$, depends on the inputs from the rest of the N neurons through the discrete time dynamical equation

$$\xi_i(t) = \Theta\left(\sum_{j=1}^{N} \omega_{ij}\xi_j(t-1) - \theta_i\right) , \tag{1.1}$$

where the *weight* ω_{ij} represents the strength of the synaptic coupling between the j-th and the i-th neurons, θ_i is the *threshold* which points out the limit between firing and rest, and Θ is the unit step *activation function* defined as

$$\Theta(h) \equiv \begin{cases} 0 & \text{if } h \leq 0\,, \\ 1 & \text{if } h > 0\,. \end{cases} \tag{1.2}$$

Then, a set of mutually connected McCulloch-Pitts units is what is called an *artificial neural network*.

In spite of the simplicity of their model, McCulloch and Pitts where able to prove that artificial neural networks could be used to do any desired computation, provided the weights ω_{ij} and the thresholds θ_i were chosen properly. This fact made that the interest towards artificial neural networks was not limited to the description of the collective behaviour of the brain, but also as a new paradigm of computing opposed to that of serial computers. However, there was a big problem which had to be solved: how can one determine the weights and thresholds in order to solve any given task?

1.2 A historical overview of artificial neural networks

As a consequence of their origins, the use of artificial neural networks was seen as a very promising method of dealing with cognitive tasks, such as pattern recognition or associative memory. From such a point of view, E. Caianiello designed in 1961 a first *learning* algorithm to adjust the connections [5], inspired in the ideas of D.O. Hebb [31].

In order to simplify the problem, F. Rosenblatt and his collaborators directed their efforts to the study of a particular type of neural networks: the *perceptrons* [63]. They believed that the perceptrons, for which the neurons are distributed in layers with feed-forward connections, could describe some of the principal characteristics of the perception mechanisms. Their most interesting result was the discovery of a perceptron learning rule, together with its corresponding convergence theorem, which could be used for the training of two-layer perceptrons. This discovery seemed to open the doors of artificial intelligence. However, in 1969 M. Minsky and S. Papert published a book [51] which stated some of the limitations of the simple perceptrons. In particular, they proved the existence of very simple tasks, such as the XOR problem, which simple perceptrons cannot learn. The effect was that this line of research was completely aborted for the next twenty years.

The discovery of the close relationship existent between neural networks and *spin glasses* oriented the investigations towards *stochastic neural networks*, specially as *content-addressable associative memory* machines. For them, the updating rule of eq. (1.1) is replaced by a similar but probabilistic law, which at low temperature recovers its original deterministic form. For instance, W.A. Little was concerned with synchronous and parallel dynamics [46], while J.J. Hopfield studied the sequential dynamic case [35, 36]. Application of statistical mechanics tools to this sort of problems continues to be very useful nowadays (see e.g. [3, 12, 33, 53, 59]).

The existence of an energy function which governs the evolution of the network towards the retrieval states was one of the basic ideas over which the use of neural networks as associative memory devices rested. Hopfield and Tank realized that in *combinatorial optimization problems* there exists also a cost function analog to the energy. Thus, interpreting its coefficients as weights of a network it is possible to achieve good solutions to them with a minimum effort [38].

From the point of view of understanding how the brain works, the discovery that some neurons of the visual cortex of the cat were specialized to the recognition of certain orientations of the optical patterns, and that adjacent neurons detected patterns with a slightly different angle [40], demonstrated that at least part of the information is stored in the brain in a topological way. Several mechanisms were proposed to explain this topological-structure formation, giving rise to some *unsupervised learning* algorithms, standing out the *winner-takes-all* method [69], the *feature maps* of T. Kohonen [43], and the *ART1* and *ART2* networks of Carpenter and Grossberg [6, 7].

Nevertheless, the reason why neural nets have become so popular in the last few years is the existence of new learning algorithms to adjust the weights and thresholds of multilayer perceptrons. The famous *error back-propagation* method was initially introduced in 1974 by P. Werbos [80], but it remained forgotten until 1985 when D.B. Parker [60] and Rumelhart, Hinton and Williams [67, 68] rediscovered and applied it to solve many problems. With back-propagation it is possible to train a neural network to learn any task from examples. Whether this is the mechanism used by the brain or not seems to be not so important, since it has given pass to new classification and interpolation tools which have proved to work better than the traditional methods.

Some interesting applications of the previous and many other models of artificial neural networks could be event reconstruction in particle accelerators [9, 10, 77] and radar signal analysis [72, 73].

In the present times artificial neural networks constitute one of the most successful and multidisciplinary subjects. People with very different formation, ranging from physicists to psychologists and from biologists to engineers, are trying to understand why they do work well, how can the learning algorithms be improved, how can they be implemented in hardware, which architectures are the best ones for each problem, how could they be modified in order to incorporate as many characteristics of the brain as possible, which sort of patterns can be stored, which are the representation of the patterns with better generalization properties, which is the maximum capacity of the net, which are the main characteristics of their dynamics, etc. This is just a small sample of the work which is taking place all over the world (see e.g. [34, 45, 52]), and to which we have tried to give a necessarily small contribution.

1.3 Multilayer neural networks

Among the different types of neural networks, those in which we have concentrated most of our research are Rosenblatt's *perceptrons*, also known as *multilayer feed-forward neural networks* [63]. In these nets there is a *layer* of *input units* whose only role is to feed input patterns into the rest of the network. Next, there are one or more intermediate or *hidden* layers of neurons evaluating the same kind of function of the weighted sum of inputs, which, in turn, send it forward to units in the following layer. This process goes on until the final or *output* level is reached, thus making it possible to read off the computation.

In the class of networks one usually deals with, there are no connections leading from a neuron to units in previous layers, nor to neurons further than the next contiguous level, i.e. every unit feeds only the ones contained in the next layer. Once we have updated all the neurons in the right order, they will not change their states, meaning that for these architectures time plays no role.

In Fig. 1.2 we have represented a three-layer perceptron with n_1 input units, n_2 hidden units in a single hidden layer, and n_3 outputs. When an input vector $\boldsymbol{\xi}$ is introduced to the net, the states of the hidden neurons acquire the values

$$\sigma_j = g\left(\sum_{k=1}^{n_1} \omega_{jk}^{(2)} \xi_k - \theta_j^{(2)}\right) \,, \; j = 1, \ldots, n_2 \,, \tag{1.3}$$

and the output of the net is the vector $\boldsymbol{\zeta}$ whose components are given by

$$\zeta_i = g\left(\sum_{j=1}^{n_2} \omega_{ij}^{(3)} \sigma_j - \theta_i^{(3)}\right) \,, \; i = 1, \ldots, n_3 \,. \tag{1.4}$$

Here we have supposed that the *activation function* can be any arbitrary function g, though it is customary to work only with bounded ones either in the interval $[0, 1]$ or in $[-1, 1]$. If this transfer function is of the form of the Θ step function of eq. (1.2) it is said that the activation is *discrete*, since the states of the neurons are forced to be in one of a finite number of different possible values. Otherwise, commonly used *continuous* activation functions are the *sigmoids* or *Fermi functions*

$$g(h) \equiv \frac{1}{1 + e^{-\beta h}} \,, \tag{1.5}$$

which satisfy

$$\lim_{\beta \to \infty} g(h) = \Theta(h) \,. \tag{1.6}$$

In the terminology of statistical mechanics the parameter β is regarded as the inverse of a temperature. However, for practical applications we will set $\beta = 1$.

In general, if we have L layers with $n_1, \ldots, n_L$ units respectively, the state of the multilayer perceptron is established by the recursive relations

$$\xi_i^{(\ell)} = g\left(\sum_{j=1}^{n_{\ell-1}} \omega_{ij}^{(\ell)} \xi_j^{(\ell-1)} - \theta_i^{(\ell)}\right) \,, \; i = 1, \ldots, n_\ell \,, \; \ell = 2, \ldots, L \,, \tag{1.7}$$

where $\boldsymbol{\xi}^{(\ell)}$ represents the state of the neurons in the ℓ-th layer, $\{\omega_{ij}^{(\ell)}\}$ the weights between units in the $(\ell - 1)$-th and the ℓ-th layers, and $\theta_i^{(\ell)}$ the threshold of the i-th unit in the ℓ-th layer. Then, the input is the vector $\boldsymbol{\xi}^{(1)}$ and the output $\boldsymbol{\xi}^{(L)}$ (see Fig. 1.3).

By *simple perceptron* one refers to networks with just two layers, the input one and the output one, without internal units, in the sense that there are no intermediate layers, and with the step activation function (1.2). These devices have been seen to have limitations, such as the XOR problem, which do not show up in feed-forward networks with hidden layers present. Actually, it has been proved that a network with just one hidden layer can represent any boolean function [11].

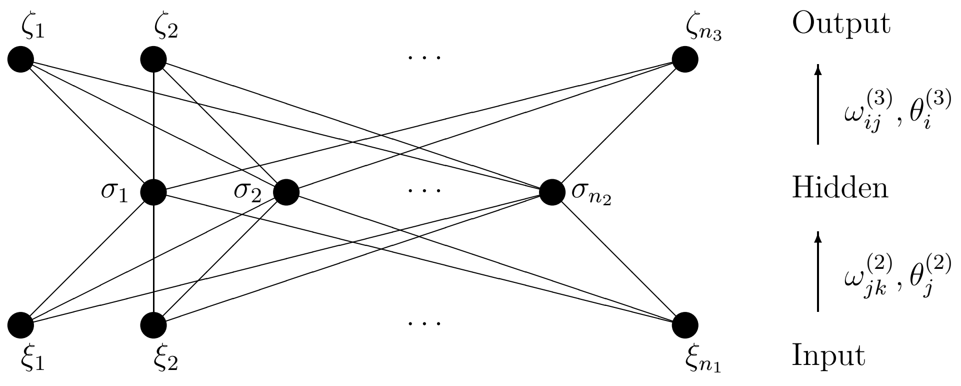

Figure 1.2: A three-layer perceptron consisting of input, hidden and output layers.

$$\boldsymbol{\xi}^{(1)} \xrightarrow[\substack{\omega_{ij}^{(2)} \\ \theta_i^{(2)}}]{} \cdots \longrightarrow \boldsymbol{\xi}^{(\ell-1)} \xrightarrow[\substack{\omega_{ij}^{(\ell)} \\ \theta_i^{(\ell)}}]{} \boldsymbol{\xi}^{(\ell)} \longrightarrow \cdots \xrightarrow[\substack{\omega_{ij}^{(L)} \\ \theta_i^{(L)}}]{} \boldsymbol{\xi}^{(L)}$$

Figure 1.3: Schematic display of the states, weights and thresholds of a multilayer neural network.

Chapter 2

Associative memory

The basic problem of *associative memory* is the storage of a set $\{\boldsymbol{\xi}^\mu, \mu = 1, \ldots, p\}$ of binary patterns, of N bits each, in such a way that, when any other pattern $\boldsymbol{\xi}$ is presented, the 'memorized' one which is closest to it is retrieved. Among the different solutions, the ones with bigger capacity and larger basins of attraction are preferred. The *capacity* is defined as the maximum number of patterns that can be stored, and the *basins of attraction* are the regions of the input space around the patterns in which the memory recall is perfect. In this chapter we will see several possible solutions based in the use of artificial neural networks.

2.1 Hopfield networks

2.1.1 Formulation of the associative memory problem

Let us suppose that we have a fully connected neural net whose dynamics is governed by eq. (1.1). The N units are updated in random or sequential order from an initial state $\boldsymbol{\xi}$. If proper connections are taken, we will prove that the evolution of this net will approach its nearest stored pattern $\boldsymbol{\xi}^\alpha$, provided the difference between them is small enough.

When dealing with this sort of networks it is convenient to modify the mathematical definition of the states in the new terms of the Ising spin-glass theory. Thus, the firing and non-firing 1 and 0 values are replaced by the *up* and *down* spin states, with numerical values +1 and −1 respectively. Moreover, the Θ activation function has to be substituted by the sign function. The new evolution equations are, then,

$$\xi_i(t+1) = \operatorname{sign}\left(h_i(t)\right) \,, \tag{2.1}$$

where

$$\operatorname{sign}(h) \equiv \begin{cases} -1 & \text{if } h \leq 0 \,, \\ +1 & \text{if } h > 0 \,, \end{cases} \tag{2.2}$$

and h_i is a commonly used quantity called the *field* of the i-th neuron, defined as

$$h_i(t) \equiv \sum_{j=1}^{N} \omega_{ij}\xi_j(t) - \theta_i \,. \tag{2.3}$$

Nevertheless, in the rest of this section we will take null values for the thresholds:

$$\theta_i = 0\,,\ i = 1, \ldots, N\,. \tag{2.4}$$

From now on we will switch from one formulation to the other whenever necessary, choosing always the most appropiate one for each particular problem.

Some lines above it has been said that the correct retrieval is achieved if the two patterns $\boldsymbol{\xi}$ and $\boldsymbol{\xi}^\alpha$ are close enough. The concept of nearness is measured with the aid of the so-called *Hamming distance*, which is defined as the number of bits in which they differ. Some equivalent expressions for the calculation of the Hamming distance $\mathcal{H}$ between $\boldsymbol{\xi}$ and $\boldsymbol{\xi}^\mu$ could be

$$\begin{aligned}
\mathcal{H}^\mu(\boldsymbol{\xi}) &= \frac{1}{4}\sum_{j=1}^{N}(\xi_j - \xi_j^\mu)^2 \\
&= \frac{1}{2}\sum_{j=1}^{N}(1 - \xi_j\xi_j^\mu) \\
&= \frac{N}{2} - \frac{1}{2}\sum_{j=1}^{N}\xi_j\xi_j^\mu \,,
\end{aligned} \tag{2.5}$$

where we have made use of the fact that the ξ_j take spin-like values ± 1. Another equivalent (but opposite) measure is the *overlap* $\mathcal{O}$, defined as the number of bits equal to both $\boldsymbol{\xi}$ and $\boldsymbol{\xi}^\mu$:

$$\begin{aligned}
\mathcal{O}^\mu(\boldsymbol{\xi}) &= N - \mathcal{H}^\mu(\boldsymbol{\xi}) \\
&= \frac{1}{4}\sum_{j=1}^{N}(\xi_j + \xi_j^\mu)^2 \\
&= \frac{1}{2}\sum_{j=1}^{N}(1 + \xi_j\xi_j^\mu) \\
&= \frac{N}{2} + \frac{1}{2}\sum_{j=1}^{N}\xi_j\xi_j^\mu \,.
\end{aligned} \tag{2.6}$$

2.1.2 The Hebb rule

The easiest and more extensively studied choice of weights which transform a neural net into an associative memory device is

$$\omega_{ij} = \frac{1}{N} \sum_{\nu=1}^{p} \xi_i^\nu \xi_j^\nu \,, \tag{2.7}$$

known as the *Hebb rule*. A remarkable property is the symmetry of the weights, $\omega_{ij} = \omega_{ji}$, which allows the introduction of the energy functional

$$E[\boldsymbol{\xi}] = -\frac{1}{2} \sum_{i,j} \omega_{ij} \xi_i \xi_j \,. \tag{2.8}$$

It is possible to show that the dynamics defined by eqs. (2.1) and (2.3) never increases this energy, meaning that the time evolution of the network tends to bring the system to a local minimum of E.

If we substitue (2.7) into (2.3) with the initial condition $\boldsymbol{\xi}(0) = \boldsymbol{\xi}^\mu$, we obtain

$$h_i^\mu(0) = \xi_i^\mu + \frac{1}{N} \sum_j \sum_{\nu \neq \mu} \xi_i^\nu \xi_j^\nu \xi_j^\mu \,. \tag{2.9}$$

Supposing that the patterns to be stored are *random*, the second term in the r.h.s. turns out to be a sum of $N \times (p-1)$ discrete random variables, each one with equally probable values $+\frac{1}{N}$ and $-\frac{1}{N}$. For large N, and due to the Central Limit Theorem, the distribution of this 'crosstalk' term is gaussian with zero mean and $\sqrt{\frac{p-1}{N}}$ standard deviation. Therefore, if the number of patterns p is small compared to the number of units N, the absolute value of the crosstalk term will have a high probability of being below 1. The consequence is that all the patterns $\boldsymbol{\xi}^\mu$ are stable configurations of the net, since the signs of the h_i^μ are the same as those of the ξ_i^μ.

A similar calculation shows that, if the initial state has n bits different to those of the pattern $\boldsymbol{\xi}^\alpha$, then

$$h_i(0) = \left(1 - \frac{2n}{N}\right) \xi_i^\alpha + \frac{1}{N} \sum_j \sum_{\nu \neq \alpha} \xi_i^\nu \xi_j^\nu \xi_j^\alpha \,. \tag{2.10}$$

Once again, if $n, p \ll N$, the network configuration will be, after one update of all the neurons, the desired $\boldsymbol{\xi}^\alpha$.

Application of the powerful mathematical tools of statistical mechanics shows that $p = 0.14N$ is the maximum number of retrievable patterns that can be stored through this method (see e.g. [1]). This limit is far beyond the theoretical bound of $p = 2N$ for random patterns [24].

2.1.3 The projection or pseudo-inverse solution

When the patterns are correlated the Hebb rule can no longer be applied. Nevertheless, there exists a very simple though non-local solution for the storage of up to N linearly independent patterns [61]. The model consists in the set of weights

$$\omega_{ij} = \frac{1}{N} \sum_{\nu,\sigma=1}^{p} \xi_i^\nu (Q^{-1})_{\nu\sigma} \xi_j^\sigma \, , \tag{2.11}$$

where Q^{-1} is the inverse of the *overlap matrix* Q formed by

$$Q_{\nu\sigma} \equiv \frac{1}{N} \sum_{k=1}^{N} \xi_k^\nu \xi_k^\sigma \, . \tag{2.12}$$

It is straightforward to see that

$$h_i^\mu(0) = \sum_{j=1}^{N} \omega_{ij} \xi_j^\mu = \xi_i^\mu \, , \tag{2.13}$$

so we conclude that every stored pattern is a stable configuration of the neural network. Now both names given to this rule are justified: the *pseudo-inverse* comes from eq. (2.11) and the *projection* from eq. (2.13).

The advantages of this method in front of the Hebb rule are clear: it can deal with correlated though linearly independent patterns, and the capacity of the network has been enlarged up to $\alpha \equiv \frac{p}{N} = 1$. Of course, the limitation to the number of patterns comes from the necessity of inverting the overlap matrix.

However, in 1987 Kanter and Sompolinsky [41] showed that the previous model does not retrieve the stored patterns if $\alpha > \frac{1}{2}$, since even an initial configuration which differs from a memorized pattern by only one bit does not evolve to the full memory. In this case it is said that the *radius of attraction* of the stored patterns are zero. The solution they proposed is the elimination of the self-coupling terms, i.e. $\omega_{ii} = 0 \, , \ \forall i$, which for large N leads to a true capacity of $\alpha = 1$, with substantial basins of attraction. The same modification also improves the behaviour of the Hebb rule.

2.1.4 Optimal stability solution

Both the Hebb and the projection rules share the property that we have an explicit formula for the calculation of the synapses. However, the second method requires the inversion of a usually very big matrix, which makes it difficult to be implemented. In 1987 Diederich and Opper [13] realized that this inversion could be done in an iterative local scheme. A further enhancement was carried out by

Krauth and Mézard [44]. They defined the *stability* Δ of the network as

$$\Delta \equiv \min_{\mu,i} \left(\xi_i^\mu \sum_j \omega_{ij} \xi_j^\mu \right) , \tag{2.14}$$

where Δ is a positive quantity whenever the weights are chosen so that all the stored patterns are stable. The stability of a network is a measure of the minimum size of the basins of attraction. Thus, the *optimal stability solution* is the one which solves the following constrained problem:

> maximize $\Delta > 0$ for $i = 1, \ldots, N$ satisfying $\xi_i^\mu \sum_j \omega_{ij} \xi_j^\mu \geq \Delta$, $\mu = 1, \ldots, p$ and $\sum_j \omega_{ij}^2 = 1$, where the independent variables are the weights.

The normalization condition is imposed to fix the invariance of the dynamics (2.1) under rescalings of the weights.

With this new geometrical point of view in mind, Krauth and Mézard proposed an iterative method, known as the *MinOver* algorithm, that converges to the optimal stability solution. Another rule, the *AdaTron* algorithm, based on quadratic optimization techniques, was derived by Anlauf and Biehl in 1989 [2]. Its main advantage is that the relaxation towards the optimal stability weights is much faster. The procedure is the following: first, one puts

$$\omega_{ij} = \frac{1}{N} \sum_{\nu=1}^{p} x_i^\nu \xi_i^\nu \xi_j^\nu , \tag{2.15}$$

where the embedding strengths x_i^ν are unknown. Then, any starting configuration with $x_i^\nu \geq 0$ is possible, in particular the *tabula rasa* $x_i^\nu = 0$. Finally, the strengths are sequentially updated through the substitution rule

$$x_i^\nu \longrightarrow x_i^\nu + \max \left\{ -x_i^\nu , \gamma \left(1 - \xi_i^\nu \sum_j \omega_{ij} \xi_j^\nu \right) \right\} , \tag{2.16}$$

with a $0 < \gamma < 2$ constant to ensure the convergence of the method.

The capacity of the solution of optimal stability for random patterns is, in principle, the maximum possible, i.e. $\alpha = 2$. Nonetheless, simulations show that the time needed to achieve a good approximation to this solution diverges when the number of patterns approaches this bound.

2.2 Maximum overlap neural networks

In the previous section we have exposed several methods of partially solving the associative memory problem with the aid of fully connected and deterministic artificial neural networks. Another improvement of physical nature consists in the introduction of a small fraction of thermal noise capable of pushing the system out of spurious minima. Nevertheless, it seems clear that, at large, they will not be able to correctly classify an arbitrary set of input patterns, since the noice induces errors, the capacity of the system has upper bounds, and the basins of attraction do not fill the input space.

Leaving aside the undoubted relevance of these methods, the possibility of connecting the units of the network in very different ways opens neural computing to wider fields of research and applications. From now on we will try to take advantage of this power.

In this section we will be concerned with the search for *optimal solutions* to the problem of associative memory [19]. By optimal we mean that every input pattern must make the network retrieve its nearest stored counterpart, with the only exception, at most, of those inputs equidistant from two or more of them (not to be confused with the optimal stability solutions of Subsect. 2.1.4). Moreover, the capacity of the network cannot be bounded by anything but the size of the input space. Proceeding in this way we clearly separate the specific interest in the problem of associative memory from the interest towards the study of fully connected spin-glass-like neural networks. The solutions found shall henceforth be called *Multilayer Associative Memory Optimal Networks* (MAMONets).

2.2.1 Optimal associative memory schemes

Pattern recognition boils down to finding the mutual overlaps between a given shape $\boldsymbol{\xi}$ and a set of stored binary patterns $\{\boldsymbol{\xi}^\mu, \mu = 1, \ldots, p\}$, of N bits each, in order to determine which is the closest. While a Hopfield network produces the result by persistence of its own configuration after evolving in time, our methods entail the 'actual' computation of the overlaps and the selection of the largest by means of multilayer nets with standard logistic functions.

The idea of calculating the overlaps among the input and stored patterns has already been put forward by Domany and Orland in [14], where some of the advantages we find were anticipated. However, Domany and Orland assume the existence of activation functions capable of finding the largest of two numbers and of picking its index, thus sidestepping the harder problem of doing so by means of 'available' types of neurons, i.e. units either linear or binary. Our schemes will precisely deal with this matter.

Binary units

Let $\mathcal{O}^\mu(\boldsymbol{\xi})$ be the *unnormalized overlap* between the input shape $\boldsymbol{\xi}$ and the stored pattern $\boldsymbol{\xi}^\mu$, i.e.

$$\mathcal{O}^\mu(\boldsymbol{\xi}) = \sum_{k=1}^{N} \left(\xi_k^\mu \xi_k + (1 - \xi_k^\mu)(1 - \xi_k)\right) , \tag{2.17}$$

$$\begin{array}{cccccccc}
\xi_1 & & \sigma^{12} & \sigma^{13} & \cdots & \sigma^{1p} & & S^1 \\
\xi_2 & & & \sigma^{23} & \cdots & \sigma^{2p} & & S^2 \\
\vdots & \longrightarrow & & & \ddots & \vdots & \longrightarrow & \vdots \\
\xi_N & & & & & \sigma^{p-1\ p} & & S^p
\end{array}$$

Figure 2.1: Scheme of a binary units three-layer perceptron for optimal associative memory.

where, for mathematical convenience, the activation values of the input units are 0 and 1. Consider an intermediate layer of units

$$\sigma^{\mu\nu}(\boldsymbol{\xi}) = \Theta\left(\mathcal{O}^{\mu}(\boldsymbol{\xi}) - \mathcal{O}^{\nu}(\boldsymbol{\xi})\right)\ ,\ \nu > \mu \text{ only,} \tag{2.18}$$

Θ being the logistic function defined in (1.2). Due to the linear character of $\mathcal{O}^{\mu}$ in the components of the input $\boldsymbol{\xi}$, the argument of our step function can be adequately written as a *weighted sum* by just identifying the weights and thresholds implicit in expression (2.17):

$$\left\{\begin{array}{lcl} \omega_k^{\mu\nu} & = & 2(\xi_k^{\mu} - \xi_k^{\nu})\,, \\ \theta^{\mu\nu} & = & \displaystyle\sum_{k=1}^{N}(\xi_k^{\mu} - \xi_k^{\nu})\,. \end{array}\right. \tag{2.19}$$

With them, we can write

$$\sigma^{\mu\nu}(\boldsymbol{\xi}) = \Theta\left(\sum_{k=1}^{N}\omega_k^{\mu\nu}\xi_k - \theta^{\mu\nu}\right)\ . \tag{2.20}$$

Further, we add an output layer of p units as shown in Fig. 2.1 and require that its α-th unit be on and the rest off, so that this index be singled out. Assume that $\boldsymbol{\xi}$ has its largest overlap with $\boldsymbol{\xi}^{\alpha}$, i.e. $\mathcal{O}^{\alpha}(\boldsymbol{\xi}) - \mathcal{O}^{\mu}(\boldsymbol{\xi}) > 0$ for every $\mu \neq \alpha$, and that the difference is always negative when the order is reversed. As a result

$$\left\{\begin{array}{lcl} \sigma^{\alpha\mu}(\boldsymbol{\xi}) = 1\ \forall\mu \neq \alpha & \Longrightarrow & \text{the } \alpha\text{-th row contains only ones,} \\ \sigma^{\mu\alpha}(\boldsymbol{\xi}) = 0\ \forall\mu \neq \alpha & \Longrightarrow & \text{the } \alpha\text{-th column contains only zeros.} \end{array}\right.$$

In rows and columns where the index μ does not occur there will always be a mixture of zeros and ones. Therefore, the feature to be detected within the $(\sigma^{\mu\nu})$

matrix is a column-row subarray of the sort

$$\begin{array}{ccccc}
\cdots & 0 & & \cdots & \\
\ddots & \vdots & & & \vdots \\
 & 0 & & \cdots & \\
 & & 1 & \cdots & 1 \\
 & & & \ddots & \vdots
\end{array}$$

with $\alpha - 1$ zeros in column α and $p - \alpha$ ones in row α. As can be easily checked, the combination

$$\left\{ \begin{array}{lcl} \omega^{\lambda,\mu\rho} & = & \left\{ \begin{array}{ll} \delta^{\lambda\mu} & \text{if } \rho > \lambda\,, \\ -\delta^{\lambda\rho} & \text{if } \mu < \lambda\,, \end{array} \right. \\ \theta^{\lambda} & = & p - \lambda - \varepsilon\,,\ 0 < \varepsilon < 1\,, \end{array} \right. \tag{2.21}$$

does the job nicely if the output activations are given by

$$S^{\lambda} = \Theta \left(\sum_{\substack{\mu,\rho \\ \mu<\rho}} \omega^{\lambda,\mu\rho} \sigma^{\mu\rho} - \theta^{\lambda} \right) . \tag{2.22}$$

Notice that the number of units on the hidden layer is nothing less than $\frac{p(p-1)}{2}$, which means that its size grows quadratically in p as the number of stored patterns increases. This signals a problem for all applications where p can be arbitrary large, and will be the major shortcoming of the method. The other drawback is its inability to cope with input patterns which are equidistant from two or more of the $\boldsymbol{\xi}^{\mu}$, as the above characteristic column-row configuration does no longer appear in these cases.

Decreasing thresholds

According to definition (2.17), $\mathcal{O}^{\mu}(\boldsymbol{\xi}) \in \{0, 1, 2, \ldots, N\}$. If we knew in advance the value of the largest overlap, say $\mathcal{O}_M$, it would suffice to choose a common threshold $\theta = \mathcal{O}_M - \varepsilon,\ 0 < \varepsilon < 1$, and compute

$$\begin{aligned} S^{\mu}(\boldsymbol{\xi}) &= \Theta\left(\mathcal{O}^{\mu}(\boldsymbol{\xi}) - \theta\right) \\ &= \Theta\left(\sum_{k=1}^{N} (2\xi_k^{\mu} - 1)\xi_k - \left(\sum_{k=1}^{N} \xi_k^{\mu} - N + \theta \right) \right) . \end{aligned} \tag{2.23}$$

With this, the activation of one S^{μ} unit on the second layer would be singling out the index of the closest pattern, and no hidden layers would be called for.

What can be done in practice is to start by using a threshold large enough and decrease it by time steps, until one of the overlaps be above it and the rest be below. Since all the overlaps can only be integers between 0 and N, the threshold

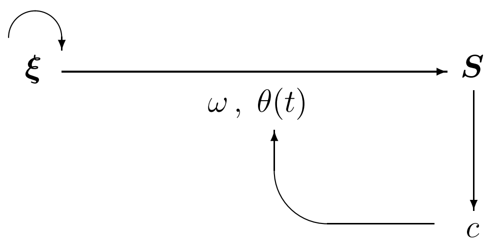

Figure 2.2: Scheme involving a control unit c with repeated threshold decrease for associative memory.

will be reduced by one unit every time, until the above condition is met. Since $\mathcal{O}_M$ can have at most the value of N, a good threshold to start with is

$$\theta(0) = N - \varepsilon\,,\ 0 < \varepsilon < 1\,. \tag{2.24}$$

At every step the same input pattern will be reprocessed, i.e.

$$\boldsymbol{\xi}(t+1) = \boldsymbol{\xi}(t)\,, \tag{2.25}$$

and an additional unit, say c, will take care of checking whether the end condition is satisfied or not (see Fig. 2.2). We define the state of this control unit as

$$c(\boldsymbol{S}) = \Theta\left(\sum_{\mu=1}^{p} S^{\mu}\right)\,, \tag{2.26}$$

i.e., it is activated only when there is a positive S^{μ}, which amounts to having $\mathcal{O}^{\mu}(\boldsymbol{\xi}) > \theta(t)$ for a certain index, say $\mu = \alpha$. Clearly, $c = 0$ when the threshold is still above all the overlaps. While this happens, θ will have to be cut down. Thus, the update rule for the variable threshold must be

$$\theta(t+1) = \theta(t) - (1 - c(\boldsymbol{S}))\,. \tag{2.27}$$

When $c = 1$, θ repeats its previous value and the network becomes stable. All the S^{μ} are zero except for S^{α}, thus providing the desired identification.

Unlike the previous scheme, this set-up allows for the recognition of a subset of patterns which are at the same minimal distance from the input $\boldsymbol{\xi}$. They appear in the form of several units simultaneously turned on at the $\boldsymbol{S}$ layer, after the threshold has got just below the elements of this subset. The same will happen with the model we propose next.

Quasilinear units

The so-called *MaxNet algorithm* was conceived for the purpose of picking winning units in neuron clusters for competitive learning [45]. The idea behind this method was to avoid the sequential calculation of overlap differences, thus making possible the selection of the maximum by a purely neural method. The technique we suggest here is yet another exploit of this nice algorithm.

Having computed the *normalized overlaps*, we store them into the units of a fully interconnected Hopfield-like network, $\boldsymbol{S}$, which, after time evolution under an appropriate update rule, will point to the maximum. Rather than a hidden layer, the present model contains a hidden time-evolving network.

From input to the hidden network at $t = 0$. We want each $S^\mu(t=0)$ to take on the value of $\frac{1}{N}\mathcal{O}^\mu(\boldsymbol{\xi})$. This is easily achieved by propagating forward the values of the $\boldsymbol{\xi}$ components in the way

$$S^\mu(0) = \sum_{k=1}^{N} \omega_k^\mu \xi_k - \theta^\mu \,, \tag{2.28}$$

i.e. with an identity (between 0 and 1) logistic function, and using the weights and thresholds

$$\left\{ \begin{array}{rcl} \omega_k^\mu &=& \dfrac{1}{N}(2\xi_k^\mu - 1)\,, \\ \theta^\mu &=& \dfrac{1}{N}\displaystyle\sum_{k=1}^{N} \xi_k^\mu - 1\,. \end{array} \right. \tag{2.29}$$

Time evolution. The rule chosen for the updating of the units, which we assume to be *synchronous*, is

$$S^\mu(t+1) = f\left(\sum_{\rho=1}^{p} \omega^{\mu\rho} S^\rho(t)\right) \,, \tag{2.30}$$

where

$$\omega^{\mu\rho} = \left\{ \begin{array}{ll} 1 & \text{if } \rho = \mu\,, \\ -\varepsilon & \text{if } \rho \neq \mu\,, \end{array} \right. \tag{2.31}$$

with

$$0 < \varepsilon \leq \frac{1}{p-1}\,, \tag{2.32}$$

and the activation is the *quasilinear function*

$$f(x) = \left\{ \begin{array}{ll} 0 & \text{if } x < 0\,, \\ x & \text{if } 0 \leq x \leq 1\,, \\ 1 & \text{if } x > 1\,. \end{array} \right. \tag{2.33}$$

Assume that a maximum exists, and let α denote its label:

$$S^{\alpha}(0) > S^{\mu}(0)\,,\ \forall \mu \neq \alpha\,. \tag{2.34}$$

Since the overlaps have been normalized, the initial arguments of f are between 0 and 1 and this function effectively is the identity. It is easy to realize that $S^{\mu}(t) \leq S^{\mu}(t-1)\,,\ \forall \mu\,,\ \forall t$, i.e. the values of all the units are monotonically decreasing.

For any t such that we still have $S^{\nu}(t) > 0$ and $S^{\lambda}(t) > 0$,

$$\begin{aligned} S^{\nu}(t) - S^{\lambda}(t) &= S^{\nu}(t-1) - \varepsilon \sum_{\mu\neq\nu} S^{\mu}(t-1) - S^{\lambda}(t-1) + \varepsilon \sum_{\mu\neq\lambda} S^{\mu}(t-1) \\ &= (1+\varepsilon)\left(S^{\nu}(t-1) - S^{\lambda}(t-1)\right)\,. \end{aligned} \tag{2.35}$$

Define

$$d^{\nu\lambda}(t) \equiv S^{\nu}(t) - S^{\lambda}(t)\,. \tag{2.36}$$

This quantity satisfies the recursive relation

$$d^{\nu\lambda}(t) = (1+\varepsilon)\, d^{\nu\lambda}(t-1)\,, \tag{2.37}$$

whose solution is

$$d^{\nu\lambda}(t) = (1+\varepsilon)^{t}\, d^{\nu\lambda}(0)\,. \tag{2.38}$$

Since the $d^{\nu\lambda}$ do not change their signs, the relative order of the values of the non-zero units remains constant. It is therefore obvious that

$$S^{\alpha}(t) > S^{\mu}(t)\,,\ \forall \mu \neq \alpha\,,\ \forall t \tag{2.39}$$

and that

$$S^{\mu}(t) = S^{\mu}(t-1) \Longleftrightarrow \begin{cases} S^{\mu}(t-1) = 0 \\ \vee \\ \mu = \alpha \text{ (the maximum) and } S^{\mu}(t-1) = 0 \text{ for } \mu \neq \alpha \end{cases} \tag{2.40}$$

i.e. the stable configuration takes the form

$$\boldsymbol{S} = (\overset{1}{0}, \ldots, \overset{\alpha-1}{0}, \overset{\alpha}{\Delta}, \overset{\alpha+1}{0}, \ldots, \overset{N}{0}) \tag{2.41}$$

which singles out the maximum, as desired.

Let $\boldsymbol{\xi}^{\beta}$ be the pattern second closest to $\boldsymbol{\xi}$. Then

$$S^{\alpha}(0) > S^{\beta}(0) \geq S^{\mu}(0)\,,\ \forall \mu \neq \alpha\,,\ \forall \mu \neq \beta\,. \tag{2.42}$$

The case $S^{\beta}(0) = 0$ is only possible if $p = 2$. In this situation the system cannot go any further, as it is already in a stable state. Otherwise, $S^{\beta}(0) > 0$ and some iterations are needed to reach the stable state. Let T be the least number of iterations necessary in order to ensure that $\boldsymbol{S}(t+1) = \boldsymbol{S}(t)$ for any $t \geq T$. By (2.39), $d^{\alpha\beta}(t)$ is less than one while $t < T$. Hence the inequality

$$(1+\varepsilon)^t \, d^{\alpha\beta}(0) < 1 \,, \text{ for } t < T \,, \tag{2.43}$$

follows. In addition, by considering the minimal difference between normalized overlaps we come to $d^{\alpha\beta}(0) = S^{\alpha}(0) - S^{\beta}(0) \geq \frac{1}{N}$, which gives a lower bound for $d^{\alpha\beta}(0)$. From this and (2.43) we get

$$\frac{1}{(1+\varepsilon)^t} > \frac{1}{N} \,, \tag{2.44}$$

which yields

$$t < \frac{\log N}{\log(1+\varepsilon)} \equiv T(N, \varepsilon) \,. \tag{2.45}$$

Since T must be an integer, the answer is

$$T = \text{upper integer part of } T(N, \varepsilon). \tag{2.46}$$

Using the most efficient ε, i.e. $\varepsilon = \frac{1}{p-1}$, we obtain

$$T(N, \varepsilon) = \frac{\log N}{\log \dfrac{p}{p-1}} \,. \tag{2.47}$$

Depending on the conditions at the outset, several cases may be distinguished:

(i). If $S^{\alpha}(0) > S^{\beta}(0) \geq S^{\mu}(0) \,, \ \mu \neq \alpha \,, \ \mu \neq \beta$, the system will eventually settle down on a state of the type

$$\boldsymbol{S}(t) = (0, \ldots, 0, \overset{\alpha}{\Delta}, 0, \ldots, 0) \,, \ 0 < \Delta < 1 \,, \text{ for } t \geq T \text{ or earlier.}$$

(ii). If $S^{\alpha_1}(0) = \cdots = S^{\alpha_r}(0) > S^{\mu}(0) \,, \ \mu \neq \alpha_1, \ldots, \alpha_r \,, \ r \leq p$, then the system does not stabilize, but:

(iia) if $r < p$ it comes to symmetric mixture states, of the sort

$$\boldsymbol{S}(t) = (0, \ldots, 0, \overset{\alpha_1}{\Delta(t)}, 0, \ldots, 0, \overset{\alpha_r}{\Delta(t)}, 0, \ldots, 0), \text{ for } t \geq T \text{ or earlier,}$$

with $0 < \Delta(t) < \Delta(t-1) < 1$;

(iib) if $r = p$ it will arrive at

$$\boldsymbol{S}(t) = (0, \ldots, 0) \,.$$

$$\begin{tikzcd}\boldsymbol{\xi} \arrow[r] & \boldsymbol{S}(t) \arrow[r] \arrow[loop above] & \boldsymbol{\zeta}\end{tikzcd}$$

Figure 2.3: Time-evolving MaxNet $\boldsymbol{S}(t)$ as part of a multilayer neural network for pattern recognition.

Thus, if the execution is stopped after exactly T iterations, the final state $\boldsymbol{S}(T)$ will be of one of the three kinds above. The interpretation of this fact is also simple. A class (i) state means that $\boldsymbol{\xi}^{\alpha}$ is the closest pattern to $\boldsymbol{\xi}$. If the network ends up in (iia), then there is a subset $\{\boldsymbol{\xi}^{\alpha_1}, \ldots, \boldsymbol{\xi}^{\alpha_r}\}$ of patterns equally similar to $\boldsymbol{\xi}$, all of them closer than the rest. The (iib) subclass corresponds to the rather unlikely case where all the stored patterns are at the same distance from $\boldsymbol{\xi}$.

The execution halt for $t = T$ may be formally regarded as equivalent to taking a time-dependent ε like

$$\varepsilon(t) = \begin{cases} \varepsilon & \text{if } t < T\,, \\ 0 & \text{if } t \geq T\,, \end{cases} \tag{2.48}$$

since, for $\varepsilon = 0$ nothing changes.

From the final state of the hidden network to the output. Here we will consider the question of actually rebuilding the pattern(s) selected by the network, i.e. of going from index-recognition to visual reconstruction. This discussion does also apply to the two previous schemes, as both end up with the same representation. The result will be 'visible' if we add an external output layer connected to the hidden network, which will not feed information into its units until the time evolution has come to an end (see Fig. 2.3).

An activation that provides the recovery of $\boldsymbol{\xi}^{\alpha}$ in case (i) is

$$\zeta_i = \Theta\left(\sum_{\mu=1}^{p} \omega_i^{\mu} S^{\mu}\right)\,, \tag{2.49}$$

with

$$\omega_i^{\mu} = \xi_i^{\mu}\,. \tag{2.50}$$

One can still wonder what comes up when applying the same method to type (ii) states. It does not take too long to realize that the shape retrieved from (iia) states is the result of adding all the single patterns in the selected subset by the boolean OR function. Even (iib) cases allow for recovery of the OR-sum of all the stored patterns if the scheme employed is step-by-step threshold reduction as explained before. An additional difficulty is the existence of apparent one-pattern retrieval states which emerge from special combinations of several $\boldsymbol{\xi}^\mu$ giving rise to another stored pattern of the same set. The difference between genuine retrieval states and these *fake* one-pattern configurations is that the former appear after the network settles on a stable state of class (i), while the latter are symmetric mixtures of the type (iia).

2.2.2 Examples and simulations

In order to assess the efficiency of the MAMONet methods, we have carried out numerical simulations of a few examples. The third MAMONet (the most adequate in our view) has been compared with the Hebb prescription and with two enhanced variants, based on the pseudo-inverse method and on the AdaTron algorithm, all of them under synchronous dynamics. Although they have been explained in Subsects. 2.1.3 and 2.1.4, some brief comments on their applicability are in order. The pseudo-inverse method is valid only when the overlap matrix Q is invertible, which amounts to requiring linear independence of all the stored patterns. An AdaTron net finds the weights by an iterative algorithm of self-consistent nature, which, in fact, leads to aimless wandering on quite a few occasions. We have used both techniques as improvements of the Hebb rule for fixing the weights in the sense that, whenever the system is posed with a $\{\boldsymbol{\xi}^\mu\}$ set leading to a singular Q matrix or otherwise preventing the AdaTron algorithm from achieving convergence, the 'straight' Hebb rule is enforced.

We fix the size of the net as well as p and, after choosing a random set $\{\boldsymbol{\xi}^\mu, \mu = 1, \dots, p\}$, all the 2^N possible initial configurations are fed into the input units. At every step, the retrieval frequency of each stored pattern, as well as those of the different kinds of non-retrieval final situations (such as spurious minima or unstable change) are computed separately. Notwithstanding that, if we want to describe the general behaviour of a particular model, the relevant quantities to be taken into account will be the cumulative averages of these frequencies over all the different iterations performed so far. Specifically, the following.

- *The average global retrieval frequency.* Consider the number of retrievals of every particular stored pattern $\boldsymbol{\xi}^\mu, \mu = 1, \dots, p$, and take its average over all the random generations of the $\{\boldsymbol{\xi}^\mu\}$ set. Since under these circumstances each $\boldsymbol{\xi}^\mu$ by itself is, of course, as stochastic as the rest, these numbers do not make sense as individual quantities, but their sum does in fact provide a measure of the power of the system to produce retrievals of single patterns belonging to the set. As an indicator of the performance of the network, such a magnitude must depend on the typical sizes and distributions of the basins of attraction produced by the method in question [1], and gives us an idea of the relative extent (referred to the whole input space) to which the system is capable of making unambiguous decisions.

- *The average frequency of spurious minima.* By spurious minima we mean configurations stable under evolution which do not reproduce, however, any of the embedded patterns. Our application of such a general definition calls for establishing at least two separate categories here.

    1. Patterns exactly opposite to the embedded $\boldsymbol{\xi}^\mu$, which are as stable as the original set, since their retrieval is actually symmetrical. For this reason they are usually counted as one-pattern retrieval states in the classical literature.
    2. Other non-retrieval stable states, including superpositions of several embedded patterns, either with the same or different coefficients (symmetrical or asymmetrical mixtures).

    When a cost or energy function exists, spurious configurations correspond to *local minima* in the energy landscape which are different from the valleys occupied by the $\boldsymbol{\xi}^\mu$ themselves.

- *The average frequency of oscillating states.*

- *The average frequency of unstable states.* In principle, none of these states repeats itself under evolution. We call *oscillating* the unstable situation where a given pair of states lead one to the other endlessly, and reserve the word *unstable* for any other case where the system shows its reluctance to settle down.

Concerning the evolution of our MAMONet model, some slightly different concepts have to be introduced.

- *Fake retrievals.* As already remarked, there are $\boldsymbol{\xi}$ which give rise to (iia) states, corresponding to more than one $\boldsymbol{\xi}^\mu$, but this may happen in such a manner that the OR-sum of those turns out to coincide with one single $\boldsymbol{\xi}^\mu$. Under unending time evolution of the hidden network these configurations would be unstable, but since the time is limited by the bound we have chosen to impose, it happens that when the evolution is stopped they pop up looking like retrieval states; thus the adjective ‘fake’.

- *Hesitant configurations.* This refers to all the remaining unstable set-ups. Contrary to the previous case, the network’s indecision is this time externally noticeable. As in fake retrievals, the system hedges its bets between

	Hebb rule	Q^{-1} method	AdaTron	MAMONet	
global retrieval	253.50	206.57	174.59		729.84
spurious	480.45	816.36	262.66		
oscillating	290.05	1.07	586.30	fake	34.55
unstable	0.00	0.00	0.45	hesitant	259.61
No. of iterations	222	1002	1002		494

Table 2.1: Average frequencies for the parallel simulation corresponding to the example $N = 10$, $p = 4$, by the four methods explained in the text. The numbers of iterations quoted were the necessary for obtaining a largest relative increase below 10^{-3}.

two or more equi-overlapping (and thus equally close) stored patterns, but, after being halted, it produces an OR-sum of patterns which is not necessarily recognizable. At that moment, the network is caught in its 'hesitation'.

- *Spurious states*. We must stress that they are absent from this scheme, as only the stored pattern themselves are truly stable under dynamic evolution.

The rule for stopping the simulation is repetition of the average values. To be more precise, we set upper bounds to the relative increase of the averaged quantities rather than to their absolute increase. Usual Montecarlo algorithms do not explore the whole input space, but produce a 'random walk' through the pattern hypercube until some convergence condition is met. However, we make an exhaustive examination of the 2^N input patterns, which in fact means the actual computation of the *quenched* average over the $\boldsymbol{\xi}$ space. Thus, the randomness is limited to the generation of sets. In addition to some smaller examples, we have studied $N = 10$. The information gathered is of the kind shown in Table 2.1 and all the normalized global retrieval rates obtained are displayed in Table 2.2.

Both the Q^{-1} and AdaTron enhancements yield rates which fall often within the same range as those for the original Hopfield network with the Hebb rule. One must bear in mind that their actual use is limited to the subclass of $\{\boldsymbol{\xi}^\mu\}$ sets which allow for their application. Therefore the results shown are for mixtures Hebb-Q^{-1} and Hebb-AdaTron in which the proportions are subject to variation. For instance, the virtually equal values for Hebb and AdaTron corresponding to α close to 1 are no coincidence, but rather the result of the (expectable) lack of convergence of the AdaTron algorithm for large p, which gave rise to the use

α	Hebb rule	Q^{-1} method	AdaTron	MAMONet
0.2	0.35	0.36	0.35	0.82
0.3	0.22	0.32	0.18	0.75
0.4	0.25	0.20	0.17	0.71
0.5	0.18	0.12	0.13	0.69
0.6	0.21	0.07	0.20	0.67
0.7	0.20	0.03	0.18	0.65
0.8	0.18	0.03	0.19	0.63
0.9	0.18	0.02	0.17	0.62
1	0.19	0.02	0.18	0.61

Table 2.2: Global retrieval rates obtained by each procedure for $N = 10$ and for different values of $\alpha = \frac{p}{N}$. The estimated error is 5×10^{-2} for the first three methods and less than 10^{-2} for the MAMONet figures.

of the Hebb rule almost throughout. In most of the cases studied, the unstable configurations found are of the oscillating type. Larger numbers of embedded patterns will surely give rise to more unstabilities of other sorts. MAMONet provides significantly larger attraction basins, while getting rid of spurious minima. Also worthy of comment is the observed growth in the rate of fake retrievals as α increases.

The simulations might have been continued for p larger than N, were it not for the prohibitively long times involved. At least as far as MAMONet is concerned, the process might go on without problems until $p = 2^N$. The constraint that all the $\boldsymbol{\xi}^\mu$ should be different allows us to predict the behaviour of the retrieval rate. When $p = 2^N$ every possible pattern must appear exactly once, and thus the rate has to be one. On the other hand, $p = 1$ would also give a unit value, as no fake retrieval or hesitation could take place either. Given the observed fall of the rate when increasing p from $p = 2$, at least one minimum has to exist (and is easily seen by doing the whole simulation for small N). The value of this minimum is an interesting subject that can be a matter of further research.

Chapter 3

Supervised learning with discrete activation functions

In Sect. 2.2 we have seen how multilayer neural networks can be applied to solve the associative memory problem. However, the characteristic which has attracted most of the interest towards them is their role as an input-output machine: whenever an input pattern is presented it responds giving out a certain output pattern. Thus, multilayer perceptrons may be regarded as families of functions whose adjustable parameters are the weights and thresholds. It must be taken into account that the architecture of the network is generally not given beforehand. That is the reason why we are free to adjust it as necessary. The leading criterion will be, of course, simplicity.

In this and in the next chapter we will be concerned with *supervised learning*, i.e. with the calculation of the parameters of multilayer feed-forward neural networks which transform several known input patterns into their corresponding output patterns (according to a given interpretation of inputs and outputs). First, we will concentrate our attention in multilayer perceptrons made of units with discrete activation functions and, afterwards, we will consider the continuous and differentiable case.

3.1 Encoding of binary patterns

The original problem of *encoding* is to turn p possible input patterns described by N digital units into a specified set of p patterns of M units, and to do it with the least possible number of intermediate processing elements [17, 18]. This may be seen as trying to condense all the information carried by the initial set of patterns into the tiniest space possible (*data compression*), and then to recover it in the form of the corresponding output patterns (*decoding*). For the sake of simplicity we will deal with the case where $N = M = p$ only, the reason being that, for this set-up, the association between every particular pattern and the position of each excited unit is quite easy to keep in mind.

As a technical subject, data compression can play a decisive role in the issue of encryption, as it uses many similar principles. The idea behind this is to increase the capacity of any storage device without having to alter the actual hardware architecture, and only by an effective reduction of the storage needs of the user. Computer-based cryptography is a modern answer to the necessity for keeping sensitive data on shared systems secure, as well as a resource for data transmission, e.g. the protection of sky-to-earth station broadcasts. In addition to storage enhancement and higher security levels, the encoding of information prior to transmission saves transfer time, e.g. on phone lines.

3.1.1 Encoding schemes

Unary input and output sets

This is the simplest set-up, from which more involved encoding systems can be devised, as we shall show later. Let us assume an input alphabet of N symbols, each of them defined by a binary pattern of N units. The choice of unary patterns (in the Ising formalism) amounts to defining every element of the input set as

$$\boldsymbol{\xi}^\mu \equiv (\overset{1}{-}, \ldots, \overset{\mu-1}{-}, \overset{\mu}{+}, \overset{\mu+1}{-}, \ldots, \overset{N}{-}) \, , \; \mu = 1, \ldots, N \, , \tag{3.1}$$

or, in components,

$$\xi_k^\mu = 2\delta_k^\mu - 1 \, . \tag{3.2}$$

We will start by requiring our network to turn a given unary input pattern of N units into an output configuration reproducing the same pattern, by means of an intermediate layer. Furthermore, for the sake of economising on memory storage, it will be quite desirable to demand that this layer be as small as possible.

The encoding strategy to be put into practice will consist in using a hidden layer (see Fig. 3.1) forming a binary representation of the N input characters in terms of -1 and $+1$ (instead of 0 and 1). Each element of this representation will be the binary translation of the number $\mu - 1$, associated to every pattern $\boldsymbol{\xi}^\mu$. As a result, the dimension of this representation (in fact, the effective byte length), henceforth called R, has the following value:

$$R = \begin{cases} \log_2 N & \text{if } \log_2 N \in \mathbb{N} \, , \\ [\log_2 N] + 1 & \text{if } \log_2 N \notin \mathbb{N} \, . \end{cases} \tag{3.3}$$

For instance, taking an input set of 4 unary patterns, one has to attach to them the numbers 0, 1, 2, 3 and put them into binary form when going to the intermediate layer, which will take up only two units:

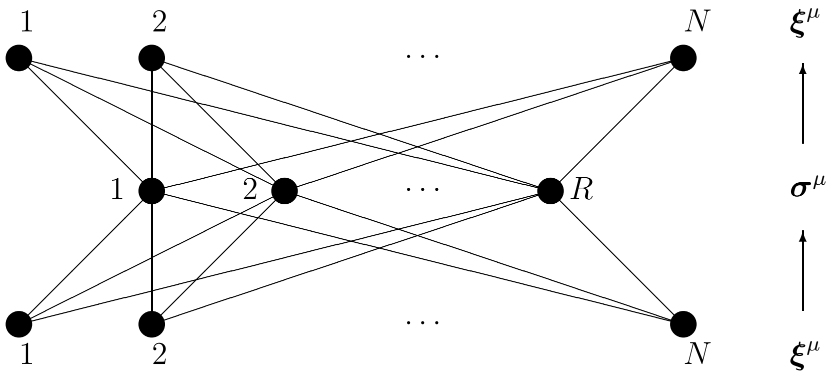

Figure 3.1: Scheme of a multilayer perceptron for the encoding of N unary patterns with a 'bottle-neck' hidden layer of $R \sim \log_2 N$.

μ	$\boldsymbol{\xi}^\mu$				$\longrightarrow$	$\boldsymbol{\sigma}^\mu$	
1	+	−	−	−	$\longrightarrow$	−	−
2	−	+	−	−	$\longrightarrow$	−	+
3	−	−	+	−	$\longrightarrow$	+	−
4	−	−	−	+	$\longrightarrow$	+	+

This sort of change of basis may be implemented by a number of techniques on any ordinary (i.e. sequential) computer, but, since we are working on a neural network, it must be achieved by just an adequate choice of the weights or connection strengths ω_{jk} and of the threshold constants θ_j, which will relate the values of the units in both layers in the way

$$\sigma_j = \text{sign}\left(\sum_{k=1}^{N} \omega_{jk}\xi_k - \theta_j\right) \ , \ j = 1, \ldots, R \, . \tag{3.4}$$

To begin with, we will tackle the previous example $N = 4$ ($R = 2$), for which the above relations lead to two systems of linear inequations:

$$\left.\begin{array}{lrcl} & \multicolumn{3}{c}{\sigma_1} \\ \boldsymbol{\xi}^1) & +\omega_{11}-\omega_{12}-\omega_{13}-\omega_{14}-\theta_1 & < & 0 \\ \boldsymbol{\xi}^2) & -\omega_{11}+\omega_{12}-\omega_{13}-\omega_{14}-\theta_1 & < & 0 \\ \boldsymbol{\xi}^3) & -\omega_{11}-\omega_{12}+\omega_{13}-\omega_{14}-\theta_1 & > & 0 \\ \boldsymbol{\xi}^4) & -\omega_{11}-\omega_{12}-\omega_{13}+\omega_{14}-\theta_1 & > & 0 \end{array}\right\}$$

$$\left.\begin{array}{lrcl} & \multicolumn{3}{c}{\sigma_2} \\ \boldsymbol{\xi}^1) & +\omega_{21}-\omega_{22}-\omega_{23}-\omega_{24}-\theta_2 & < & 0 \\ \boldsymbol{\xi}^2) & -\omega_{21}+\omega_{22}-\omega_{23}-\omega_{24}-\theta_2 & > & 0 \\ \boldsymbol{\xi}^3) & -\omega_{21}-\omega_{22}+\omega_{23}-\omega_{24}-\theta_2 & < & 0 \\ \boldsymbol{\xi}^4) & -\omega_{21}-\omega_{22}-\omega_{23}+\omega_{24}-\theta_2 & > & 0 \end{array}\right\}$$

The unknowns to be solved are not just the eight coefficients for the connection strengths ω_{jk}, but the thresholds θ_j , $j = 1, 2$ as well. A possible and relatively simple solution of this double system is:

$$\left\{\begin{array}{ccc} \omega_{11} & = & -1 \\ \omega_{12} & = & -1 \\ \omega_{13} & = & +1 \\ \omega_{14} & = & +1 \\ \theta_1 & = & 0 \end{array}\right. \qquad \left\{\begin{array}{ccc} \omega_{21} & = & -1 \\ \omega_{22} & = & +1 \\ \omega_{23} & = & -1 \\ \omega_{24} & = & +1 \\ \theta_2 & = & 0 \end{array}\right.$$

Considering the weights as the coefficients of a connection strength or *weight matrix*, this part of the solution may be put into the form:

$$(\omega_{jk}) = \begin{pmatrix} - & - & + & + \\ - & + & - & + \end{pmatrix} .$$

This way of writing the weights does already exhibit a key feature of the solution we have chosen, namely, the coincidence of the weight coefficients with the values of the intermediate units for the different input patterns in the sense that for each pattern $\mu = i$, we have $\omega_{ij} = \sigma_j^i$.

In order to guess the general solution for arbitrary N, we will make one step further and study the case $N = 5$. This is also of interest because it shows what happens when $\log_2 N$ is not an integer. Given that now $R = 3$, and in contrast with the previous example, the intermediate layer is now, so to speak, largely unexploited because the five patterns corresponding to the input set are only a fraction of the 2^3 theoretically possible sequences that might be formed in this substructure. However, since at present we are concerned with the configurations coming from our unary input patterns only, we limit our requirements to this set:

μ	$\boldsymbol{\xi}^\mu$					$\longrightarrow$	$\boldsymbol{\sigma}^\mu$		
1	+	−	−	−	−	$\longrightarrow$	−	−	−
2	−	+	−	−	−	$\longrightarrow$	−	−	+
3	−	−	+	−	−	$\longrightarrow$	−	+	−
4	−	−	−	+	−	$\longrightarrow$	−	+	+
5	−	−	−	−	+	$\longrightarrow$	+	−	−

Again, we look for suitable weights and thresholds leading to the 'good' combinations of input and result. Like before, we have taken the values of the $\boldsymbol{\sigma}^\mu$ to be the translated binary digits of $\mu - 1$, but we have not yet cared to write an explict expression for the figures. One can observe that these numbers have to do with the quantity $\frac{\mu-1}{2^{3-j}}$, which, for the present range of μ and j takes on the following values

	$\frac{\mu-1}{2^{3-j}}$		
$\mu\backslash j$	1	2	3
1	$\frac{0}{2^2}=0$	$\frac{0}{2^1}=0$	$\frac{0}{2^0}=0$
2	$\frac{1}{2^2}=\frac{1}{4}$	$\frac{1}{2^1}=\frac{1}{2}$	$\frac{1}{2^0}=1$
3	$\frac{2}{2^2}=\frac{1}{2}$	$\frac{2}{2^1}=1$	$\frac{2}{2^0}=2$
4	$\frac{3}{2^2}=\frac{3}{4}$	$\frac{3}{2^1}=\frac{3}{2}$	$\frac{3}{2^0}=3$
5	$\frac{4}{2^2}=1$	$\frac{4}{2^1}=2$	$\frac{4}{2^0}=4$

Now we can see that, if the digits of $\boldsymbol{\sigma}^\mu$ were binary 0 and 1, their value would be precisely

$$\sigma^\mu_{j\,\mathrm{bin}} = \left[\frac{\mu-1}{2^{R-j}}\right] \bmod 2\,. \tag{3.5}$$

When translating this back into −1 and +1, the state of each intermediate neuron reads

$$\sigma^\mu_j = (-1)^{\left[\frac{\mu-1}{2^{R-j}}\right]+1}\,. \tag{3.6}$$

Next, in the spirit of the solution found for $N = 4$, we will seek an answer based on the general ansatz that the values of the weights and of the hidden neurons for the input set can always be taken to coincide, i.e. choosing

$$\omega_{jk} = \sigma^k_j = (-1)^{\left[\frac{k-1}{2^{R-j}}\right]+1} \tag{3.7}$$

it will perhaps be possible to find thresholds allowing us to implement the desired relations. For $N = 5$, this ansatz means taking the weight matrix to be

$$(\omega_{jk}) = \begin{pmatrix} - & - & - & - & + \\ - & - & + & + & - \\ - & + & - & + & - \end{pmatrix} .$$

Proceeding similarly to the $N = 4$ case, we would realize that possible values for $\theta_j\,,\ j = 1, 2, 3$ do in fact exist, thus justifying the validity of the assay. A possible solution is $\theta_1 = 3$, $\theta_2 = \theta_3 = 1$. What remains to be checked is the acceptability of our assay for any N. We will show that this is sustained by finding a suitable set of thresholds $\theta_j\,,\ j = 1, \ldots, R$, valid for an arbitrary number of input patterns. Taking the expression in components for the unary patterns and making the ansatz for the ω_{jk} we obtain

$$\begin{aligned} \sigma_j^\mu &= \operatorname{sign}\left(\sum_{k=1}^N \omega_{jk}\xi_k^\mu - \theta_j\right) \\ &= \operatorname{sign}\left(\sum_{k=1}^N (-1)^{\left[\frac{k-1}{2^{R-j}}\right]+1}(2\delta_k^\mu - 1) - \theta_j\right) \\ &= \operatorname{sign}\left(2(-1)^{\left[\frac{\mu-1}{2^{R-j}}\right]+1} - \sum_{k=1}^N (-1)^{\left[\frac{k-1}{2^{R-j}}\right]+1} - \theta_j\right) . \end{aligned} \tag{3.8}$$

Since $\sigma_j^\mu = (-1)^{\left[\frac{\mu-1}{2^{R-j}}\right]+1}$, the equality will be satisfied if

$$\theta_j + \sum_{k=1}^N (-1)^{\left[\frac{k-1}{2^{R-j}}\right]+1} = 0\,, \tag{3.9}$$

i.e.

$$\theta_j = \sum_{k=1}^N (-1)^{\left[\frac{k-1}{2^{R-j}}\right]} . \tag{3.10}$$

Since this solution does always exist, the ansatz has been proved to work for arbitrary N, thus providing a general answer given by the weights (3.7) and the thresholds (3.10).

The next step is to go from the intermediate layer to the output units. Given that the output set of patterns will be identical to the input one, the whole encoding process from one into the other means taking a certain $\boldsymbol{\xi}^\mu$ to obtain some $\boldsymbol{\xi}^\nu$, where the index ν may be different from the given μ. If we demand that the translation be injective, i.e. no pair of different input patterns can yield the same output pattern, and bearing in mind that the number of patterns in each set is the same, when encoding for all possible μ the relation between the set of output indices ν and the input labels μ can be no other than a permutation of

N elements. Selecting a translation scheme amounts to making the choice of a specific permutation. It is therefore reasonable to make a first approach to this problem by choosing the simplest element of the symmetric group, namely the identity. Thus, if we denote by $\boldsymbol{S}^\mu$ the output pattern resulting from entering $\boldsymbol{\xi}^\mu$ into the network, the situation corresponding to the identity is that in which $\boldsymbol{S}^\mu = \boldsymbol{\xi}^\mu$, which, for instance, in the case $N = 5$ can be represented by

μ		$\boldsymbol{\sigma}^\mu$		$\longrightarrow$			$\boldsymbol{S}^\mu$		
1	$-$	$-$	$-$	$\longrightarrow$	$+$	$-$	$-$	$-$	$-$
2	$-$	$-$	$+$	$\longrightarrow$	$-$	$+$	$-$	$-$	$-$
3	$-$	$+$	$-$	$\longrightarrow$	$-$	$-$	$+$	$-$	$-$
4	$-$	$+$	$+$	$\longrightarrow$	$-$	$-$	$-$	$+$	$-$
5	$+$	$-$	$-$	$\longrightarrow$	$-$	$-$	$-$	$-$	$+$

The set of weights and thresholds accomplishing this for any N will be guessed from the study of a particular case and justified in general afterwards. These connection weights and thresholds must make possible the relation

$$S_i^\mu = \operatorname{sign}\left(\sum_{j=1}^{R} \omega_{ij}\sigma_j^\mu - \theta_i\right) . \tag{3.11}$$

We will focus now on the set-up for $N = 4$. This particular case is read from the previous table by removing the first column and the last row for the $\boldsymbol{\sigma}$ and the last column and row for the $\boldsymbol{S}$. Then, the above sign relation leads to four systems of inequations, i.e.

$$S_1^\mu \quad \left.\begin{array}{lrcl} \boldsymbol{\xi}^1) & -\omega_{11}-\omega_{12}-\theta_1 & > & 0 \\ \boldsymbol{\xi}^2) & -\omega_{11}+\omega_{12}-\theta_1 & < & 0 \\ \boldsymbol{\xi}^3) & +\omega_{11}-\omega_{12}-\theta_1 & < & 0 \\ \boldsymbol{\xi}^4) & +\omega_{11}+\omega_{12}-\theta_1 & < & 0 \end{array}\right\}$$

$$S_2^\mu \quad \left.\begin{array}{lrcl} \boldsymbol{\xi}^1) & -\omega_{21}-\omega_{22}-\theta_2 & < & 0 \\ \boldsymbol{\xi}^2) & -\omega_{21}+\omega_{22}-\theta_2 & > & 0 \\ \boldsymbol{\xi}^3) & +\omega_{21}-\omega_{22}-\theta_2 & < & 0 \\ \boldsymbol{\xi}^4) & +\omega_{21}+\omega_{22}-\theta_2 & < & 0 \end{array}\right\}$$

$$S_3^\mu \quad \left.\begin{array}{lrcl} \boldsymbol{\xi}^1) & -\omega_{31}-\omega_{32}-\theta_3 & < & 0 \\ \boldsymbol{\xi}^2) & -\omega_{31}+\omega_{32}-\theta_3 & < & 0 \\ \boldsymbol{\xi}^3) & +\omega_{31}-\omega_{32}-\theta_3 & > & 0 \\ \boldsymbol{\xi}^4) & +\omega_{31}+\omega_{32}-\theta_3 & < & 0 \end{array}\right\}$$

$$S_4^\mu \quad \left.\begin{array}{lrcl} \boldsymbol{\xi}^1) & -\omega_{41}-\omega_{42}-\theta_4 & < & 0 \\ \boldsymbol{\xi}^2) & -\omega_{41}+\omega_{42}-\theta_4 & < & 0 \\ \boldsymbol{\xi}^3) & +\omega_{41}-\omega_{42}-\theta_4 & < & 0 \\ \boldsymbol{\xi}^4) & +\omega_{41}+\omega_{42}-\theta_4 & > & 0 \end{array}\right\}$$

One of the simplest solutions one can think of is:

$$\left\{\begin{array}{lcl} \omega_{11} & = & -1 \\ \omega_{12} & = & -1 \\ \theta_1 & = & +1 \end{array}\right. \qquad \left\{\begin{array}{lcl} \omega_{21} & = & -1 \\ \omega_{22} & = & +1 \\ \theta_2 & = & +1 \end{array}\right.$$

$$\left\{\begin{array}{lcl} \omega_{31} & = & +1 \\ \omega_{32} & = & -1 \\ \theta_3 & = & +1 \end{array}\right. \qquad \left\{\begin{array}{lcl} \omega_{41} & = & +1 \\ \omega_{42} & = & +1 \\ \theta_4 & = & +1 \end{array}\right.$$

Once more, we observe coincidence between the weight coefficients and the values taken on by the intermediate units in the way

$$\omega_{ij} = \sigma_j^i = (-1)^{\left[\frac{i-1}{2^{R-j}}\right]+1} . \tag{3.12}$$

Next, this relationship will be assumed as tenable for arbitrary N, and its validity demonstrated by showing the existence of possible thresholds θ_i fulfilling (3.11).

By our assumption, we have

$$\sum_{j=1}^{R} \omega_{ij}\sigma_j^\mu = \sum_{j=1}^{R} \sigma_j^i \sigma_j^\mu \leq \sum_{j=1}^{R} (\sigma_j^i)^2 = R, \tag{3.13}$$

i.e., since each term is a product of two signs, the weighted sum of the values of the hidden units achieves a maximum equal to R when all the pairs of signs coincide, which happens precisely for $\mu = i$. Otherwise, there must be at least one pair of opposed signs and therefore

$$\sum_{j=1}^{R} \omega_{ij}\sigma_j^\mu \leq R-2\,, \text{ for } \mu \neq i\,. \tag{3.14}$$

Going back to (3.11), given that $S_i^\mu = \xi_i^\mu$, that has a plus sign for the unit at $i = \mu$ and minuses elsewhere, the thresholds θ_i must be such that

$$\left\{ \begin{array}{ll} \displaystyle\sum_{j=1}^{R} \omega_{ij}\sigma_j^\mu - \theta_i > 0 & \text{for the maximum } (\mu = i)\,, \\ \displaystyle\sum_{j=1}^{R} \omega_{ij}\sigma_j^\mu - \theta_i < 0 & \text{for the rest } (\mu \neq i)\,. \end{array} \right. \tag{3.15}$$

This is compatible with (3.13) and (3.14). In fact, by simply taking the thresholds within a certain range the fulfilment of these conditions is automatically ensured. This range is

$$\theta_i = R - 2 + \varepsilon\,,\ i = 1, \ldots, N\,,\ 0 < \varepsilon < 2\,, \tag{3.16}$$

but, in order to work with determined objects, we content ourselves with choosing

$$\theta_i = R - 1\,,\ i = 1, \ldots, N\,. \tag{3.17}$$

For an arbitrary permutation of N elements, the picture is slightly altered to

$$\begin{array}{ccccc} \xi_k^\mu & \longrightarrow & \sigma_j^\mu & \longrightarrow & S_i = \xi_i^\nu \\ & \omega_{jk} & & \omega_{ij} & \\ & \theta_j & & \theta_i & \end{array}$$

$$\nu = \tau(\mu)\,,\ \tau \in \{\text{permutations of } N \text{ elements}\}$$

All these steps can be equally retraced with the only difference that the weights ω_{ij} now coincide with the $\boldsymbol{\sigma}$ up to a label reshuffle, i.e., instead of (3.12) we have $\omega_{\tau(\mu)j} = \sigma_j^\mu$, or, equivalently,

$$\omega_{\mu j} = \sigma_j^{\tau^{-1}(\mu)} = (-1)^{\left[\frac{\tau^{-1}(\mu)-1}{2^{R-j}}\right]+1}\,. \tag{3.18}$$

Thus, our general solution is

$$\left\{ \begin{array}{lcll} \omega_{ij} & = & (-1)^{\left[\frac{\tau^{-1}(i)-1}{2^{R-j}}\right]+1}, & i = 1, \ldots, N\,,\ j = 1, \ldots, R\,, \\ \theta_i & = & R - 1\,, & i = 1, \ldots, N\,. \end{array} \right. \tag{3.19}$$

Arbitrary input and output sets

The obvious continuation of the work so far is an enhancement of the above described system so as to make it capable of translating binary patterns of a given arbitrary input set into elements of another arbitrary (but also specified) output set. If $\boldsymbol{\zeta}^\mu\,,\ \mu = 1, \ldots, N$ denotes the arbitrary input set and $\boldsymbol{S}^\mu\,,\ \mu = 1, \ldots, N$ are the output patterns, in general different from the $\boldsymbol{\zeta}^\mu$, we will require our network to produce $\boldsymbol{S}^{\tau(\mu)}$ as output whenever $\boldsymbol{\zeta}^\mu$ is read as input, being τ any specified permutation of N elements. Actually, the use of τ is redundant in the sense that, as there is now no natural relationship between the ordering of the input and output patterns, the use of different τ may at any rate be interpreted as using always the identity permutation after a previous reshuffle of the labels of the output set.

a) Five layers. A quite simple alternative is the actual enlargement of our unary pattern permutator system, by turning the old input and output layers into intermediate ones and adding two layers where the new arbitrary sets can be read and written, as depicted in the following diagram:

$$\begin{array}{ccccccccc}
\zeta_l^\mu & \longrightarrow & \xi_k^\mu & \longrightarrow & \sigma_j^\mu & \longrightarrow & \xi_i^{\tau(\mu)} & \longrightarrow & S_h^{\tau(\mu)} \\
 & \omega_{kl} & & \omega_{jk} & & \omega_{ij} & & \omega_{hi} & \\
 & \theta_k & & \theta_j & & \theta_i & & \theta_h &
\end{array}$$

We use indices l to denote each unit of the input patterns and indices h to label each neuron in the output layer. While the three intermediate levels work exactly as in the previous network, two new sets of connection weights and thresholds will have to implement the translation from arbitrary sequences to unary patterns and the other way round.

First, we look at the step from the input layer to the first intermediate layer, in which the weights ω_{kl} and the thresholds θ_k have to satisfy

$$\xi_k^\mu = \operatorname{sign}\left(\sum_{k=1}^{N} \omega_{kl}\zeta_l^\mu - \theta_k\right) . \tag{3.20}$$

It is not difficult to guess

$$\omega_{kl} = \zeta_l^k\,, \tag{3.21}$$

the reason for this choice being that it has the property of making the weighted sum of the input $\boldsymbol{\zeta}^\mu$ achieve a maximum of value N precisely for $\mu = k$, i.e.

$$\sum_{l=1}^{N} \omega_{kl}\zeta_l^\mu = \sum_{l=1}^{N} \zeta_l^k\zeta_l^\mu \leq \sum_{l=1}^{N} (\zeta_l^k)^2 = N\,. \tag{3.22}$$

As we have seen, this type of reasoning works when we require the next layer to be in a state where one neuron is on and the others are off, which is indeed the case for the unary configurations $\boldsymbol{\xi}^\mu$. Taking this into account, our choice of the threshold must be such that

$$\begin{cases} \displaystyle\sum_{l=1}^{N} \omega_{kl}\zeta_l^\mu - \theta_k > 0 & \text{for the maximum } (\mu = k)\,, \\ \displaystyle\sum_{l=1}^{N} \omega_{kl}\zeta_l^\mu - \theta_k < 0 & \text{for the rest } (\mu \neq k)\,, \end{cases} \tag{3.23}$$

because the components of $\boldsymbol{\xi}^\mu$ have to be $+1$ for $k = \mu$ and -1 for $k \neq \mu$. The possibility that we will take is

$$\theta_k = N - 1\,,\ k = 1, \ldots, N\,. \tag{3.24}$$

As for the last step, the equality to be satisfied is

$$\begin{aligned} S_h^{\tau(\mu)} &= \operatorname{sign}\left(\sum_{i=1}^{N} \omega_{hi}\xi_i^{\tau(\mu)} - \theta_j\right) \\ &= \operatorname{sign}\left(\sum_{i=1}^{N} \omega_{hi}(2\delta_i^{\tau(\mu)} - 1) - \theta_h\right) \\ &= \operatorname{sign}\left(2\omega_{h\tau(\mu)} - \sum_{i=1}^{N} \omega_{hi} - \theta_h\right)\,. \end{aligned} \tag{3.25}$$

By analogy with the calculation of the weights ω_{jk}, we try

$$\omega_{h\tau(\mu)} = S_h^{\tau(\mu)}\,, \tag{3.26}$$

i.e.

$$\omega_{hi} = S_h^i\,, \tag{3.27}$$

which leads us to an equation for the thresholds:

$$S_h^{\tau(\mu)} = \operatorname{sign}\left(2S_h^{\tau(\mu)} - \sum_{i=1}^{N} S_h^{(i)} - \theta_h\right)\,. \tag{3.28}$$

Clearly, it will hold if

$$\sum_{i=1}^{N} S_h^i + \theta_h = 0\,. \tag{3.29}$$

Thus,

$$\theta_h = -\sum_{\nu=1}^{N} S_h^\nu\,. \tag{3.30}$$

This constitutes a solution. Even though we have retained both the type of structure and the conceptual simplicity of the first network, this design has the disadvantage that it uses up to $2N+R$ intermediate units in its three intermediate layers, which can be rather wasteful as N grows larger.

b) Three Layers. Another option is to give up the use of the reduced layer, i.e. the one with R units. For $N = 2^R$ this substructure acts as a filter in the sense that, even if a non-unary pattern reaches the previous layer, the possible states of the reduced one are such that the signals sent forward to the next layer will give rise to a unary sequence anyway. As a result of this construction, no matter whether an input pattern belongs to the set $\{\boldsymbol{\zeta}^\mu\}$ or not, the corresponding output will be one of the $\boldsymbol{S}^\mu$. Nevertheless, we shall see that, as far as the input and output alphabets themselves are concerned, the same translation task can be performed by a network with just one intermediate layer of N units. Although the removal of the reduced layer may mean the loss of this sifting power, it will no doubt be a substantial gain in memory economy.

There are several possible schemes of this sort, one of them being

$$\begin{array}{ccccc} \zeta_l^\mu & \longrightarrow & \xi_k^\mu & \longrightarrow & S_h^{\tau(\mu)} \\ & \omega_{kl} & & \omega_{hk} & \\ & \theta_k & & \theta_h & \end{array}$$

Since this is almost like cutting out two layers and two sets of connections from the five-level device, the weights and thresholds for what remains are easily found to be

$$\left\{ \begin{array}{rcl} \omega_{kl} & = & \zeta_l^k \,, \\ \theta_k & = & N-1 \,, \end{array} \right. \tag{3.31}$$

and

$$\left\{ \begin{array}{rcl} \omega_{hk} & = & S_h^{\tau(k)} \,, \\ \theta_h & = & -\displaystyle\sum_{\nu=1}^{N} S_h^\nu \,, \end{array} \right. \tag{3.32}$$

respectively. Although good, this solution does not seem to be optimal, as one might wish to do the same task with a reduced intermediate level instead of one of N units. However, the answer we have found is a bit discouraging, and lays in the following

Theorem: It is not possible to encode through the scheme

$$\begin{array}{ccccc} \zeta_l^\mu & \longrightarrow & \sigma_j^\mu & \longrightarrow & S_h^{\tau(\mu)} \\ & \omega_{jl} & & \omega_{hj} & \\ & \theta_j & & \theta_h & \end{array}$$

for arbitrary sets $\{\zeta_l^\mu\}$ and $\{S_h^{\tau(\mu)}\}$.

Proof: It suffices to show particular examples of pattern sets leading to contradiction:

1. Special choice of output patterns

μ	$\boldsymbol{\sigma}^\mu$		$\longrightarrow$	$\boldsymbol{S}^\mu$			
1	$-$	$-$	$\longrightarrow$	$\boxed{-}$	$-$	$-$	$-$
2	$-$	$+$	$\longrightarrow$	$\boxed{+}$	$-$	$+$	$+$
3	$+$	$-$	$\longrightarrow$	$\boxed{+}$	$-$	$+$	$-$
4	$+$	$+$	$\longrightarrow$	$\boxed{-}$	$+$	$+$	$-$

For this choice of the output alphabet, the first column of the $\boldsymbol{S}$ patterns, i.e. S_1^μ , $\mu = 1, 2, 3, 4$ (marked out in the table) happens to be the *exclusive-OR*, or *XOR*, boolean function. As has been shown in [51] (see also [34] and other works, or Subsect. 3.2.1), this rather elementary computation cannot be solved by a simple perceptron, which amounts to stating that the task of obtaining S_1^μ from the $\boldsymbol{\sigma}^\mu$ can by no means be performed by a single step from the reduced layer to that containing the $\boldsymbol{S}^\mu$. Moreover, this sort of inconsistency will show up whenever we take an $N = 4$, $R = 2$ system where one of the output columns reproduces the values of the XOR function. For arbitary N we would encounter the same hindrance if an output column took on the values of the generalized *parity* (or rather *oddness*) function, which is defined to be $+1$ when there is an odd number of plus signs in the input and -1 otherwise, and constitutes the R-dimensional extension of XOR.

2. Special case of input patterns

μ	$\boldsymbol{\zeta}^\mu$				$\longrightarrow$	$\boldsymbol{\sigma}^\mu$	
1	$-$	$-$	$+$	$+$	$\longrightarrow$	$\boxed{-}$	$-$
2	$+$	$+$	$-$	$-$	$\longrightarrow$	$\boxed{-}$	$+$
3	$-$	$+$	$-$	$+$	$\longrightarrow$	$\boxed{+}$	$-$
4	$+$	$-$	$+$	$-$	$\longrightarrow$	$\boxed{+}$	$+$

Making use of our freedom to select arbitrary sets of input patterns, we have taken one whose elements are not linearly independent. As a result, a contradiction now arises from the ensuing expressions limiting the thresholds. Consideration of the relations for $\mu = 1$ and $\mu = 2$ leads to $\theta_1 > 0$ whereas the unequalities for $\mu = 3$ and $\mu = 4$ require $\theta_1 < 0$, leaving no chance of realizing this scheme. The same kind of reasoning is applicable to arbitrary N.

c) Four Layers. Even though the above theorem bans the possibility of implementing the theoretically optimal scheme, we can still hope to get close to it in some sense. The difficulty found in the step from the input to the intermediate layer will be removed by demanding that the $\boldsymbol{\zeta}^\mu$, although arbitrary, be linearly independent. As for the way from the $\boldsymbol{\sigma}$ units to the output cells, we will introduce a further intermediate layer, working exactly as in the five-layer scheme, i.e.

$$\begin{array}{ccccccc} \zeta_l^\mu & \longrightarrow & \sigma_j^\mu & \longrightarrow & \xi_i^{\tau(\mu)} & \longrightarrow & S_h^{\tau(\mu)} \\ & \omega_{jl} & & \omega_{ij} & & \omega_{hi} & \\ & \theta_j & & \theta_i & & \theta_h & \end{array}$$

where the only unknown things are the ω_{jl} and θ_j. We will start by going back to the solution in the five-layer network, but this time we will be a bit more audacious and look for alternatives where the sign function be redundant. Thus, we will look for two successive affine transformations such that

$$\begin{array}{ccccc} \boldsymbol{\zeta}^\mu & \longrightarrow & \boldsymbol{\xi}^\mu & \longrightarrow & \boldsymbol{\sigma}^\mu \\ & \boldsymbol{\xi} = A\boldsymbol{\zeta} + \boldsymbol{B} & & \boldsymbol{\sigma} = C\boldsymbol{\xi} + \boldsymbol{D} & \end{array}$$

The advantage of doing so is that the result of composing both will be another transformation of the same kind providing the direct passage from the $\boldsymbol{\zeta}^\mu$ to the $\boldsymbol{\sigma}^\mu$.

The first affine map in terms of components reads

$$\xi_k = \sum_l A_{kl}\zeta_l + B_k \, , \tag{3.33}$$

where the coefficients of the matrix A and of the vector $\boldsymbol{B}$ are to be found. By recalling the form of the unary $\boldsymbol{\xi}$ patterns, we must have

$$\begin{aligned} \xi_k^\mu &= \sum_l A_{kl}\zeta_l^\mu + B_k \\ &= 2\delta_k^\mu - 1 \, . \end{aligned} \tag{3.34}$$

A solution satisfying this is

$$\left\{ \begin{array}{lll} A_{kl} = 2(\zeta)^{-1}{}_{kl} \, , & k, l = 1, \ldots, N \, , \\ B_k = -1 \, , & k = 1, \ldots, N \, , \end{array} \right. \tag{3.35}$$

where $(\zeta)^{-1}$ is the inverse of the matrix

$$(\zeta)_{l\mu} \equiv \zeta_l^\mu . \tag{3.36}$$

Therefore, this solution exists only when the matrix (ζ) is inversible, thus the necessity of requiring all the different $\boldsymbol{\zeta}^\mu$ to be linearly independent.

The conditions on the second transformation are

$$\sigma_j = \sum_k C_{jk}\xi_k + D_j \, , \tag{3.37}$$

and, for each unary pattern, they lead to

$$\begin{aligned} \sigma_j^\mu &= \sum_k C_{jk}\xi_k^\mu + D_j \\ &= \sum_k C_{jk}(2\delta_k^\mu - 1) + D_j \\ &= 2C_{j\mu} - \sum_k C_{jk} + D_j \, , \end{aligned} \tag{3.38}$$

which are seen to be fulfilled by the solution:

$$\left\{ \begin{array}{rcll} C_{j\mu} & = & \frac{1}{2}\sigma_j^{\mu}\,, & j=1,\dots,R\,,\ \mu=1,\dots,N\,, \\ D_j & = & \dfrac{1}{2}\displaystyle\sum_{\nu=1}^{N}\sigma_j^{\nu}, & j=1,\dots,R\,. \end{array} \right. \tag{3.39}$$

Composing both maps one gets

$$\begin{aligned} \boldsymbol{\sigma} &= C\boldsymbol{\xi}+\boldsymbol{D} \\ &= (CA)\,\boldsymbol{\zeta}+(C\boldsymbol{B}+\boldsymbol{D})\,. \end{aligned} \tag{3.40}$$

Putting this into components and replacing all the coefficients with the expressions for the solutions we have just found,

$$\begin{aligned} \sigma_j &= \sum_l\sum_\nu C_{j\nu}A_{\nu l}\;\zeta_l+\sum_k C_{jk}B_k+D_j \\ &= \sum_l\sum_\nu \frac{1}{2}\sigma_j^{\nu}\;2(\zeta)^{-1}{}_{\nu l}\;\zeta_l+\underbrace{\sum_k\frac{1}{2}\sigma_j^{k}(-1)+\frac{1}{2}\sum_\nu\sigma_j^{\nu}}_{0} \\ &= \sum_l\sum_\nu \sigma_j^{\nu}(\zeta)^{-1}{}_{\nu l}\zeta_l\,. \end{aligned} \tag{3.41}$$

As we see, the resulting transformation has the appealing feature of being free from the inhomogeneous term, which has vanished on composing the two maps. Thus, the transformation reduces to just multiplying a matrix by the components of $\boldsymbol{\zeta}^{\mu}$. Therefore, sticking to the type of conventions used up to now, we can say that all the thresholds are zero and the weight matrix, having ω_{jl} as coefficients, is specified by

$$\left\{ \begin{array}{rcl} \sigma_j & = & \displaystyle\sum_l\omega_{jl}\zeta_l\,, \\ \omega_{jl} & = & \displaystyle\sum_\nu\sigma_j^{\nu}(\zeta)^{-1}{}_{\nu l} \;=\; \sum_{\nu=1}^{N}(-1)^{\left[\frac{\nu-1}{2^{R-j}}\right]+1}(\zeta)^{-1}{}_{\nu l}\,. \end{array} \right. \tag{3.42}$$

d) Further variants. In addition to the preceding ones, we have found other schemes which are, in fact, only variations of those already described. For instance, departing from the five layer network N:N:R:N:N, we have composed the two intermediate transformations, thus getting rid of the $\boldsymbol{\sigma}$ layer at the expense of using some more involved weights and thresholds, the result being an N:N:N:N structure called a' in the diagram. Next, we have found b' moving back the permutation τ in b from the second to the first transformation. Finally, the composition of the first two steps of c gives a three-layer network N:N:N called c'. By way of summarizing and completing this picture, all the quantities occurring are listed in the Tables 3.1 and 3.2.

$a)$
$$\begin{tikzcd}[column sep=huge]
\zeta_l^\mu \arrow[r, "\left\{\begin{smallmatrix}\omega_{kl}\\ \theta_k\end{smallmatrix}\right."'] & \xi_k^\mu \arrow[r, mapsto, "\left\{\begin{smallmatrix}\omega_{jk}\\ \theta_j\end{smallmatrix}\right."'] & \sigma_j^\mu \arrow[r, "\left\{\begin{smallmatrix}\omega_{ij}\\ \theta_i\end{smallmatrix}\right."'] & \xi_i^{\tau(\mu)} \arrow[r, mapsto, "\left\{\begin{smallmatrix}\omega_{hi}\\ \theta_h\end{smallmatrix}\right."'] & S_h^{\tau(\mu)}
\end{tikzcd}$$

$a')$
$$\begin{tikzcd}[column sep=huge]
\zeta_l^\mu \arrow[r, "\left\{\begin{smallmatrix}\omega_{kl}\\ \theta_k\end{smallmatrix}\right."'] & \xi_k^\mu \arrow[r, "\left\{\begin{smallmatrix}\omega_{ik}\\ \eta_i\end{smallmatrix}\right."'] & \xi_i^{\tau(\mu)} \arrow[r, mapsto, "\left\{\begin{smallmatrix}\omega_{hi}\\ \theta_h\end{smallmatrix}\right."'] & S_h^{\tau(\mu)}
\end{tikzcd}$$

$b)$
$$\begin{tikzcd}[column sep=huge]
\zeta_l^\mu \arrow[r, "\left\{\begin{smallmatrix}\omega_{kl}\\ \theta_k\end{smallmatrix}\right."'] & \xi_k^\mu \arrow[r, mapsto, "\left\{\begin{smallmatrix}\omega_{hk}\\ \theta_h\end{smallmatrix}\right."'] & S_h^{\tau(\mu)}
\end{tikzcd}$$

$b')$
$$\begin{tikzcd}[column sep=huge]
\zeta_l^\mu \arrow[r, "\left\{\begin{smallmatrix}\omega_{il}\\ \kappa_i\end{smallmatrix}\right."'] & \xi_i^{\tau(\mu)} \arrow[r, mapsto, "\left\{\begin{smallmatrix}\omega_{hi}\\ \theta_h\end{smallmatrix}\right."'] & S_h^{\tau(\mu)}
\end{tikzcd}$$

$c)$
$$\begin{tikzcd}[column sep=huge]
\zeta_l^\mu \arrow[r, Rightarrow, "\left\{\begin{smallmatrix}\omega_{jl}\\ 0\end{smallmatrix}\right."'] & \sigma_j^\mu \arrow[r, "\left\{\begin{smallmatrix}\omega_{ij}\\ \theta_i\end{smallmatrix}\right."'] & \xi_i^{\tau(\mu)} \arrow[r, mapsto, "\left\{\begin{smallmatrix}\omega_{hi}\\ \theta_h\end{smallmatrix}\right."'] & S_h^{\tau(\mu)}
\end{tikzcd}$$

$c')$
$$\begin{tikzcd}[column sep=huge]
\zeta_l^\mu \arrow[r, "\left\{\begin{smallmatrix}\Omega_{il}\\ \theta_i\end{smallmatrix}\right."'] & \xi_i^{\tau(\mu)} \arrow[r, mapsto, "\left\{\begin{smallmatrix}\omega_{hi}\\ \theta_h\end{smallmatrix}\right."'] & S_h^{\tau(\mu)}
\end{tikzcd}$$

$$\begin{cases} \longrightarrow & \mathrm{sign}(x) \\ \longmapsto & \mathrm{sign}(x) = \dfrac{x}{2} \\ \Longrightarrow & \mathrm{sign}(x) = x \end{cases}$$

Table 3.1: Different network structures for encoding. The type of arrow drawn indicates the sort of functions of the weighted sum minus threshold that can be alternatively used to yield the same result. A simple arrow denotes the sign function, one with tail means that the argument is twice a sign (so instead of taking the sign we can just divide by two). The double arrow means that the sign function is absolutely redundant.

$R = [\log_2 N] + 1 - \delta_{[N]N}$

$\xi_k^\mu = 2\delta_k^\mu - 1$
$\sigma_j^\mu = (-1)^{\left[\frac{\mu-1}{2^{R-j}}\right]+1}$

$\omega_{kl} = \zeta_l^k$	$\theta_k = N - 1$
$\omega_{jk} = \sigma_j^k$	$\theta_j = -\sum_{\nu=1}^{N} \sigma_j^\nu \quad (0 \text{ if } N = 2^R)$
$\omega_{ij} = \sigma_j^{\tau^{-1}(i)}$	$\theta_i = R - 1$
$\omega_{hi} = S_h^i$	$\theta_h = -\sum_{\nu=1}^{N} S_h^\nu$
$\omega_{ik} = \frac{1}{2}\sum_{j=1}^{R} \omega_{ij}\omega_{jk}$	$\eta_i = R - 1 - \sum_{k=1}^{N} \omega_{ik}$
$\omega_{hk} = S_h^{\tau(k)}$	θ_h
$\omega_{il} = \zeta_l^{\tau^{-1}(i)}$	$\kappa_i = N - 1$
$\omega_{jl} = \sum_{\nu=1}^{N} \sigma_j^\nu (\zeta)^{-1}{}_{\nu l}$	0
$\Omega_{il} = \sum_{j=1}^{R} \omega_{ij}\omega_{jl}$	θ_i

Table 3.2: Expressions for the weights and thresholds in the different network structures for encoding.

3.1.2 Accessibilities

Once an encoding scheme has been chosen, one might wonder which is the result when the input pattern is none of the input alphabet. It may seem unjustified, since different encoding solutions will produce different outputs. However, this is the basis of almost all the current applications of multilayer neural networks: first, weights and thresholds are calculated (e.g. by means of learning) and then the network is used to predict, classify or interpolate. Lots of examples may be given, such as hyphenation algorithms, protein secondary structure determiners and family tree relationship predictors [67].

In what follows we shall concern ourselves with the working of the initial unary-pattern three-layer permuter device. In fact, if the input pattern is not unary the network does not work! The reason is that the *fields*

$$h_j = \sum_{k=1}^{N} \omega_{jk}\xi_k - \theta_j \tag{3.43}$$

may vanish for some j, and then $\sigma_j = \text{sign}(h_j)$ is no longer well defined. There are several possible ways out:

1. Redefining the sign function, either as

$$\text{sign}(x) \equiv \begin{cases} -1 & \text{if } h < 0\,, \\ +1 & \text{if } h \geq 0\,, \end{cases} \tag{3.44}$$

   or the other way around

$$\text{sign}(x) \equiv \begin{cases} -1 & \text{if } h \leq 0\,, \\ +1 & \text{if } h > 0\,. \end{cases} \tag{3.45}$$

   This, however, is a rather unpleasant solution because it brings about a manifest asymmetry between the chances of obtaining -1 and $+1$.

2. Shifting the thresholds

$$\theta_j \longrightarrow \theta_j + \varepsilon\,, \ |\varepsilon| < 1\,, \tag{3.46}$$

   i.e. non-integer values are now allowed. Again, we get an unwanted asymmetry, since all the zero fields would, from now on, give a certain sign depending on the target unit but not on the input pattern.

3. Making the intermediate σ_j units take on three values, -1, 0 and $+1$:

$$\text{sign}(x) \equiv \begin{cases} -1 & \text{if } x < 0\,, \\ 0 & \text{if } x = 0\,, \\ +1 & \text{if } x > 0\,. \end{cases} \tag{3.47}$$

4. Introducing a finite (but low) *temperature*, and making the activations be stochastic. Then, the sign taken on by every unit is no longer the result of a deterministic function, but rather a random variable, for which the probabilities of obtaining -1 or $+1$ are given by sigmoid curves whose shapes depend on $\beta \equiv \frac{1}{T}$ and approach that of a step function as β goes to infinity (deterministic limit). The condition that this temperature should be low is necessary in order to preserve (after taking an average over many realizations) the same result as for $T = 0$ when the input patterns are the $\boldsymbol{\xi}^{\mu}$.

Accessibilities of a three-valued unit intermediate layer

The third option calls for a study of the accessibility of the different $\boldsymbol{\sigma}$. By accessibility of a binary pattern, thought of as a *memory*, we mean the fraction of starting arbitrary states which leads to that particular pattern [37]:

$$A(\boldsymbol{\sigma}) \equiv \frac{\text{No. of input patterns giving } \boldsymbol{\sigma}}{\text{No. of possible different input patterns}} . \tag{3.48}$$

Since the input layer has been supposed to have N two-state units,

$$A(\boldsymbol{\sigma}) = \frac{\text{No. of input patterns giving } \boldsymbol{\sigma}}{2^N} . \tag{3.49}$$

As happens in associative memory networks, different memories of the same size may be in general not equally easy to recall. The parallel to the appearance of *spurious* memories in an associative memory device is now the existence of the (to some extent unwanted) zero states. An open question about our zero-temperature encoding system is how to interpret the different sequences which end up in the same $\boldsymbol{\sigma}$ state. These sequences, rather than resembling each other in the sense of being close by Hamming distance (as happens in associative memory) are such that they tend to produce a value σ_j in the j-th unit depending on the similarity between the input pattern $\boldsymbol{\xi}$ and the j-th row of (ω_{jk}), which we shall call $\boldsymbol{\omega}_j$.

A most interesting property of our scheme is the vanishing of all the input thresholds whenever the number of external units equals an exact power of two, i.e.

$$\theta_j = \sum_{k=1}^{N} (-1)^{\left[\frac{k-1}{2^{R-j}}\right]} = -\sum_{k=1}^{N} \omega_{jk} = 0 \,, \text{ for } N = 2^R \,, \; j = 1, \ldots, R \,, \tag{3.50}$$

as can be seen by looking at the (ω_{jk}) matrix, since for $N = 2^R$ the sum of all the coefficients in each row is zero.

At zero temperature, the values of the σ_j are determined by the value of the fields h_j. A little thought shows that it can take as value any two integers between

$-N$ and N, and that the frequency with which every value occurs is a binomial coefficient arising from simple combinatorics:

$$h_j = \begin{cases} N & \text{if } \boldsymbol{\xi} \text{ differs from } \boldsymbol{\omega}_j \text{ in no signs} \quad \Rightarrow \quad 1 \quad \text{possible } \boldsymbol{\xi} \\ N-2 & \text{if } \boldsymbol{\xi} \text{ differs from } \boldsymbol{\omega}_j \text{ in 1 sign} \quad \Rightarrow \quad N \quad \text{possible } \boldsymbol{\xi} \\ N-4 & \text{if } \boldsymbol{\xi} \text{ differs from } \boldsymbol{\omega}_j \text{ in 2 signs} \quad \Rightarrow \quad \binom{N}{2} \quad \text{possible } \boldsymbol{\xi} \\ \vdots & \qquad\qquad\qquad\qquad\qquad\qquad \vdots \\ 2 & \text{if } \boldsymbol{\xi} \text{ differs from } \boldsymbol{\omega}_j \text{ in } \frac{N}{2}-1 \text{ signs} \quad \Rightarrow \quad \binom{N}{\frac{N}{2}-1} \quad \text{possible } \boldsymbol{\xi} \\ 0 & \text{if } \boldsymbol{\xi} \text{ differs from } \boldsymbol{\omega}_j \text{ in } \frac{N}{2} \text{ signs} \quad \Rightarrow \quad \binom{N}{\frac{N}{2}} \quad \text{possible } \boldsymbol{\xi} \\ -2 & \text{if } \boldsymbol{\xi} \text{ differs from } \boldsymbol{\omega}_j \text{ in } \frac{N}{2}+1 \text{ signs} \quad \Rightarrow \quad \binom{N}{\frac{N}{2}+1} \quad \text{possible } \boldsymbol{\xi} \\ \vdots & \qquad\qquad\qquad\qquad\qquad\qquad \vdots \\ -(N-2) & \text{if } \boldsymbol{\xi} \text{ differs from } \boldsymbol{\omega}_j \text{ in } N-1 \text{ sign} \quad \Rightarrow \quad N \quad \text{possible } \boldsymbol{\xi} \\ -N & \text{if } \boldsymbol{\xi} \text{ differs from } \boldsymbol{\omega}_j \text{ in } N \text{ signs} \quad \Rightarrow \quad 1 \quad \text{possible } \boldsymbol{\xi} \end{cases}$$

where $\boldsymbol{\xi}$ and $\boldsymbol{\omega}_j$ mean the sets of signs $\{\xi_k\}$ and $\{\omega_{jk}\}$, $k = 1, \ldots, N$. Denoting by $f(h_j)$ the *frequency* of h_j, or number of possibilities that the weighted sum equals h_j, we have thus obtained

$$f(h_j) = \binom{N}{\frac{N-h_j}{2}} , \tag{3.51}$$

and therefore

$$f(h_j = 0) \quad = \quad \binom{N}{\frac{N}{2}} , \tag{3.52}$$

$$f(h_j \neq 0) \quad = \quad f(h_j > 0) + f(h_j < 0) = 2f(h_j > 0) = 2^N - \binom{N}{\frac{N}{2}} . \tag{3.53}$$

We shall reason below that the accessibility of every *non-spurious* pattern (i.e. free from zeros) may be put in terms of just the joint frequencies or probabilities that a number of field components vanish. It is for this reason that the calculation of these joint frequencies must be understood first. We start by considering

$$f(h_i = 0, h_j = 0) , \ i \neq j .$$

A fundamental property of our connection weight matrix is that for this same situation, $N = 2^R$, their rows are mutually orthogonal. Since the coefficients are

-1 and $+1$, this means that for any two given rows, one half of the coefficients coincide and the other half are just opposite.

The frequency we are going to evaluate is the total number of input possibilities for the $\boldsymbol{\xi}$, unary or not, such that the equations

$$\left.\begin{array}{rcl} \omega_{i1}\xi_1 + \omega_{i2}\xi_2 + \cdots + \omega_{iN}\xi_N & = & 0 \\ \omega_{j1}\xi_1 + \omega_{j2}\xi_2 + \cdots + \omega_{jN}\xi_N & = & 0 \end{array}\right\} \tag{3.54}$$

are simultaneously satisfied. By the above orthogonality property, we can put

$$\begin{array}{rclcrcl} \omega_{ik_1} & = & \omega_{jk_1} & ,\ldots, & \omega_{ik_{N/2}} & = & \omega_{jk_{N/2}}\,, \\ \omega_{ik'_1} & = & -\omega_{jk'_1} & ,\ldots, & \omega_{ik'_{N/2}} & = & -\omega_{jk'_{N/2}}\,, \end{array} \tag{3.55}$$

where we have denoted by $k_1, \ldots, k_{N/2}$ the indices for which the coefficients coincide and by $k'_1, \ldots, k'_{N/2}$ those for which they are opposite. In terms of these sets of indices, the system of two equations reads

$$\left.\begin{array}{rcl} \underbrace{\omega_{ik_1}\xi_{k_1} + \cdots + \omega_{ik_{N/2}}\xi_{k_{N/2}}}_{A} + \underbrace{\omega_{ik'_1}\xi_{k'_1} + \cdots + \omega_{ik'_{N/2}}\xi_{k'_{N/2}}}_{B} & = & 0 \\ \omega_{ik_1}\xi_{k_1} + \cdots + \omega_{ik_{N/2}}\xi_{k_{N/2}} - \omega_{ik'_1}\xi_{k'_1} - \cdots - \omega_{ik'_{N/2}}\xi_{k'_{N/2}} & = & 0 \end{array}\right\} \tag{3.56}$$

where A and B are partial weighted sums defined as shown. The resulting system for these two new variables is immediately solved:

$$\left.\begin{array}{l} A + B = 0 \\ A - B = 0 \end{array}\right\} \;\Rightarrow\; A = B = 0 \tag{3.57}$$

which, in turn, implies

$$\left.\begin{array}{rcl} \omega_{ik_1}\xi_{k_1} + \cdots + \omega_{ik_{N/2}}\xi_{k_{N/2}} & = & 0 \\ \omega_{ik'_1}\xi_{k'_1} + \cdots + \omega_{ik'_{N/2}}\xi_{k'_{N/2}} & = & 0 \end{array}\right\} \tag{3.58}$$

Now, the unknowns in each equation are independent. Thus, for each of them, we can make the same reasoning as before when $h_j = 0$, with the only difference that N has to be replaced with $\frac{N}{2}$, as each identity contains just a half of the original number of terms. Thus

$$f_{N/2}(h_i = 0) = \begin{pmatrix} \frac{N}{2} \\ \frac{N}{4} \end{pmatrix}\,, \tag{3.59}$$

and the joint frequency is found like a joint probability:

$$f_N(h_i = 0, h_j = 0) = f_{N/2}(h_i = 0)\, f_{N/2}(h_j = 0) = \begin{pmatrix} \frac{N}{2} \\ \frac{N}{4} \end{pmatrix}^2 . \tag{3.60}$$

The next case is

$$f(h_i = 0, h_j = 0, h_k = 0)\,,\ i,\ j,\ k \text{ all different.}$$

Let $\boldsymbol{\omega}_i$, $\boldsymbol{\omega}_j$ and $\boldsymbol{\omega}_k$ denote the i-th, j-th and k-th rows of coefficients in the weight matrix, which are known to be mutually orthogonal. Based on this knowledge, we proceed analogously to the previous case, and realize that the three equations for the field components may be put in terms of four partial weighted sums, that we will call A, B, C and D, of the same sort that A and B above, but containing $\frac{N}{4}$ terms each one.

$$\left\{\begin{array}{ll} A & \text{common to } \boldsymbol{\omega}_i,\ \boldsymbol{\omega}_j \text{ and } \boldsymbol{\omega}_k, \\ B & \text{common to } \boldsymbol{\omega}_i \text{ and } \boldsymbol{\omega}_j, \text{ and opposed in } \boldsymbol{\omega}_k, \\ C & \text{common to } \boldsymbol{\omega}_i \text{ and } \boldsymbol{\omega}_k, \text{ and opposed in } \boldsymbol{\omega}_j, \\ D & \text{common to } \boldsymbol{\omega}_j \text{ and } \boldsymbol{\omega}_k, \text{ and opposed in } \boldsymbol{\omega}_i. \end{array}\right.$$

In terms of these partial sums, the equations are

$$\left.\begin{array}{rcl} A+B+C-D & = & 0 \\ A+B-C+D & = & 0 \\ A-B+C+D & = & 0 \end{array}\right\} \tag{3.61}$$

Since there are three equations and four unknowns, we can leave one as a free variable and solve the others as a function of the first. Taking A as free, the solution is

$$B = C = D = -A. \tag{3.62}$$

Now, let us consider what values A can take on. This partial sum has an expression of the type

$$A = \omega_{ik_1}\xi_{k_1} + \cdots + \omega_{ik_{N/4}}\xi_{k_{N/4}}\,. \tag{3.63}$$

Hence, if we now imagine that A is a fixed number, the possibilities that this sum has this precise value are, by the same rule as at the beginning,

$$f_{N/4}(A) = \binom{\frac{N}{4}}{\frac{\frac{N}{4}-A}{2}}\,. \tag{3.64}$$

Next, we look at the three other variables. The reasoning is the same for each of them. Since $B = -A$, once A takes on a given value, B is determined, and we must therefore count in how many different ways the equality

$$\omega_{ik'_1}\xi_{k'_1} + \cdots + \omega_{ik'_{N/4}}\xi_{k'_{N/4}} = -A \tag{3.65}$$

is accomplished. This is a weighted sum having $\frac{N}{4}$ independent terms of the kind studied. Therefore

$$f_{N/4}(B(A)) = f_{N/4}(-A) = \binom{\frac{N}{4}}{\frac{\frac{N}{4}+A}{2}} = \binom{\frac{N}{4}}{\frac{\frac{N}{4}-A}{2}} = f_{N/4}(A)\,. \tag{3.66}$$

Doing the same for the other two variables, we arrive at

$$\begin{aligned} f(h_i = 0, h_j = 0, h_k = 0) &= \sum_A f(A)\, f(B(A))\, f(C(A))\, f(D(A)) \\ &= \sum_{\substack{A=-N/4 \\ \text{step } 2}}^{N/4} \binom{\frac{N}{4}}{\frac{\frac{N}{4}-A}{2}}^4 = \sum_{k=0}^{N/4} \binom{\frac{N}{4}}{k}^4 . \end{aligned} \qquad (3.67)$$

The following joint frequency is a bit more difficult to compute, but it gives an idea of what has to be done for any number of vanishing field components. If we want to calculate

$$f(h_i = 0, h_j = 0, h_k = 0, h_l = 0)\,,\ i,\ j,\ k,\ l \text{ all diffferent},$$

after writing down the equations, we pick up the partial sums common to two or more of them:

$$\left\{ \begin{array}{ll} A & \text{common to } \boldsymbol{\omega}_i,\ \boldsymbol{\omega}_j,\ \boldsymbol{\omega}_k \text{ and } \boldsymbol{\omega}_l, \\ B & \text{common to } \boldsymbol{\omega}_i,\ \boldsymbol{\omega}_j \text{ and } \boldsymbol{\omega}_k, \text{ and opposed in } \boldsymbol{\omega}_l, \\ C & \text{common to } \boldsymbol{\omega}_i,\ \boldsymbol{\omega}_j \text{ and } \boldsymbol{\omega}_l, \text{ and opposed in } \boldsymbol{\omega}_k, \\ D & \text{common to } \boldsymbol{\omega}_i,\ \boldsymbol{\omega}_k \text{ and } \boldsymbol{\omega}_l, \text{ and opposed in } \boldsymbol{\omega}_j, \\ E & \text{common to } \boldsymbol{\omega}_j,\ \boldsymbol{\omega}_k \text{ and } \boldsymbol{\omega}_l, \text{ and opposed in } \boldsymbol{\omega}_i, \\ F & \text{common to } \boldsymbol{\omega}_i \text{ and } \boldsymbol{\omega}_j, \text{ and opposed in } \boldsymbol{\omega}_k \text{ and } \boldsymbol{\omega}_l, \\ G & \text{common to } \boldsymbol{\omega}_i \text{ and } \boldsymbol{\omega}_k, \text{ and opposed in } \boldsymbol{\omega}_j \text{ and } \boldsymbol{\omega}_l, \\ H & \text{common to } \boldsymbol{\omega}_i \text{ and } \boldsymbol{\omega}_l, \text{ and opposed in } \boldsymbol{\omega}_j \text{ and } \boldsymbol{\omega}_k. \end{array} \right.$$

Then, we express the equations using the variables that denote these sums

$$\left. \begin{array}{rcl} A+B+C+D-E+F+G+H &=& 0 \\ A+B+C-D+E+F-G-H &=& 0 \\ A+B-C+D+E-F+G-H &=& 0 \\ A-B+C+D+E-F-G+H &=& 0 \end{array} \right\} \qquad (3.68)$$

Next, we find the degree of indetermination (eight unknowns minus four equations yield four degrees of freedom) in order to know how many unknowns remain arbitrary. The system will be solved by putting the rest as a function of the arbitrary ones. Considering A, B, C and D to be free, we get

$$\left\{ \begin{array}{rcl} E &=& -2A-B-C-D\,, \\ F &=& -A-B-C\,, \\ G &=& -A-B-D\,, \\ H &=& -A-C-D\,. \end{array} \right. \qquad (3.69)$$

For the free variables, the same considerations are repeated. For instance, A can take on every two integers between $-\frac{N}{8}$ and $\frac{N}{8}$

$$-\frac{N}{8} \leq A \leq \frac{N}{8} \text{ (step 2)}\,,$$

with frequencies

$$f_{N/8}(A) = \binom{\frac{N}{8}}{\frac{\frac{N}{8}-A}{2}} . \tag{3.70}$$

As a result, the whole joint frequency is given by

$$\begin{aligned}
&f(h_i = 0, h_j = 0, h_k = 0, h_l = 0) \\
&= \sum_A \sum_B \sum_C \sum_D f(A)\, f(B)\, f(C)\, f(D)\, f(E(A,B,C,D)) \\
&\quad \times f(F(A,B,C,D))\, f(G(A,B,C,D))\, f(H(A,B,C,D)) \\
&= \sum_{\substack{A=-N/8\\ \text{step } 2}}^{N/8} \sum_{\substack{B=-N/8\\ \text{step } 2}}^{N/8} \sum_{\substack{C=-N/8\\ \text{step } 2}}^{N/8} \sum_{\substack{D=-N/8\\ \text{step } 2}}^{N/8} \binom{\frac{N}{8}}{\frac{\frac{N}{8}-A}{2}} \binom{\frac{N}{8}}{\frac{\frac{N}{8}-B}{2}} \binom{\frac{N}{8}}{\frac{\frac{N}{8}-C}{2}} \binom{\frac{N}{8}}{\frac{\frac{N}{8}-D}{2}} \\
&\quad \times \binom{\frac{N}{8}}{\frac{\frac{N}{8}+2A+B+C+D}{2}} \binom{\frac{N}{8}}{\frac{\frac{N}{8}+A+B+C}{2}} \binom{\frac{N}{8}}{\frac{\frac{N}{8}+A+B+D}{2}} \binom{\frac{N}{8}}{\frac{\frac{N}{8}+A+C+D}{2}} ,
\end{aligned} \tag{3.71}$$

or, rearranging indices,

$$\begin{aligned}
&f(h_i = 0, h_j = 0, h_k = 0, h_l = 0) \\
&= \sum_{a=0}^{N/8} \sum_{b=0}^{N/8} \sum_{c=0}^{N/8} \sum_{d=0}^{N/8} \binom{\frac{N}{8}}{a} \binom{\frac{N}{8}}{b} \binom{\frac{N}{8}}{c} \binom{\frac{N}{8}}{d} \binom{\frac{N}{8}}{2a+b+c+d-\frac{N}{4}} \\
&\quad \times \binom{\frac{N}{8}}{\frac{N}{4}-(a+b+c)} \binom{\frac{N}{8}}{\frac{N}{4}-(a+b+d)} \binom{\frac{N}{8}}{\frac{N}{4}-(a+c+d)} .
\end{aligned} \tag{3.72}$$

Up to this point, the binomial coefficients are to be understood in the general sense, i.e. when the number downstairs is negative or when the difference between upstairs and downstairs is a negative integer, they must be taken to be zero. Otherwise we would have to explicitly state that the sum is restricted to a, b, c and d between the bounds and also fulfilling

$$\left\{ \begin{array}{ccccc}
0 & \leq & 2a+b+c+d-\frac{N}{4} & \leq & \frac{N}{8} \\
0 & \leq & \frac{N}{4}-(a+b+c) & \leq & \frac{N}{8} \\
0 & \leq & \frac{N}{4}-(a+b+d) & \leq & \frac{N}{8} \\
0 & \leq & \frac{N}{4}-(a+c+d) & \leq & \frac{N}{8}
\end{array} \right. \tag{3.73}$$

In fact, since all the terms that fail to satisfy this give a zero contribution, the calculation of these sums is much easier than it looks.

The procedure described is completely general. Following these steps for any number of vanishing fields, one considers the common pieces in the initial equations, solves an indetermined linear system, uses the expressions of the frequencies for the values of weighted sums and arrives at multiple sums involving binomial coefficients only. The multiplicity of the final sum is always the degree of indetermination of the linear system.

As anticipated, we are going to find the accessibilities in terms of the preceding frequencies only, namely the $f(h_1 = 0, \ldots, h_j = 0)$, $1 \leq j \leq R$, which we shall call *orthogonalities*. We start with the total number of possible different input binary patterns, i.e. 2^N. This figure must be equal to the sum of the frequencies for all the possible sorts of field configurations for the $\boldsymbol{\sigma}$ level, thus

$$2^N = \sum_{j=0}^{R} \sum_{\{k_1,\ldots,k_j\}} f(h_1 \neq 0, \ldots, h_{k_1} = 0, \ldots, h_{k_j} = 0, \ldots, h_R \neq 0)\,, \tag{3.74}$$

where $\{k_1, \ldots, k_j\}$ denotes a choice of j indices among the R existing ones. The indices picked are those for which the associated field component vanishes, while the rest are non-zero. f denotes the corresponding rate of occurrence, i.e. the number of input patterns yielding that type of field configuration. Since j runs from 0 to R, this sum ranges over all the possibilities that can take place. It can be argued that these frequencies depend on the number of components that vanish, but not on the position they are located at, i.e.

$$\begin{aligned} & f(h_1 \neq 0, \ldots, h_{k_1} = 0, \ldots, h_{k_j} = 0, \ldots, h_R \neq 0) \\ = \;& f(h_1 = 0, \ldots, h_j = 0, h_{j+1} \neq 0, \ldots, h_R \neq 0) \end{aligned} \tag{3.75}$$

for all possible rearrangements. Therefore,

$$2^N = \sum_{j=0}^{R} \binom{R}{j} f(h_1 = 0, \ldots, h_j = 0, h_{j+1} \neq 0, \ldots, h_R \neq 0)\,. \tag{3.76}$$

Separating the term $j = 0$

$$\begin{aligned} & f(h_1 \neq 0, \ldots, h_R \neq 0) \\ = \;& 2^N - \sum_{j=1}^{R} \binom{R}{j} f(h_1 = 0, \ldots, h_j = 0, h_{j+1} \neq 0, \ldots, h_R \neq 0)\,. \end{aligned} \tag{3.77}$$

After this, the considerations made so far for all the possible configurations can be reproduced to all the sequences for which the first j field components vanish. Notice that this gives no information about the other h, i.e. some of them may be vanishing as well and therefore we have to put

$$\begin{aligned} & f(h_1 = 0, \ldots, h_j = 0) \\ = \;& \sum_{k=0}^{R-j} \binom{R-j}{k} f(h_1 = 0, \ldots, h_{j+k} = 0, h_{j+k+1} \neq 0, \ldots, h_R \neq 0)\,. \end{aligned} \tag{3.78}$$

Once more, the first term in the summatory is separated:

$$\begin{aligned}
&f(h_1=0,\ldots,h_j=0,h_{j+1}\neq 0,\ldots,h_R\neq 0)\\
&\quad= \; f(h_1=0,\ldots,h_j=0)\\
&\qquad -\sum_{k=1}^{R-j}\binom{R-j}{k} f(h_1=0,\ldots,h_{j+k}=0,h_{j+k+1}\neq 0,\ldots,h_R\neq 0) \qquad (3.79)
\end{aligned}$$

Eq. (3.77), together with eqs. (3.79), constitute a set of interrelated recursive equations, whose solution we have worked out with some labour in Appendix A, the result being given by the beautiful expression

$$f(h_1\neq 0,\ldots,h_R\neq 0)=2^N+\sum_{k=1}^{R}(-1)^k\binom{R}{k} f(h_1=0,\ldots,h_k=0) \qquad (3.80)$$

and therefore, the accessibilities of the $\boldsymbol{\sigma}$ patterns are given by

$$A(\boldsymbol{\sigma}^\mu)=\frac{1}{N2^N} f(h_1\neq 0,\ldots,h_R\neq 0)\,,\ \mu=1,\ldots,N\,. \qquad (3.81)$$

The calculations of this section may be useful in other fields of physics and mathematics owing to the fact that binary input patterns may be regarded as the vertices of an *N-dimensional hypercube* or, equivalently, as the vectors which go from the center of the hypercube to its corners. Following this geometrical interpretation, the orthogonality $f(h_1=0,\ldots,h_j=0)$ counts the number of vectors perpendicular to a given set of j mutually orthogonal vectors, $j=1,\ldots,R\,,\ N=2^R$, and so on. This sort of analysis is applicable, for instance, to the configuration space of Ising models.

a) Example $N=4,\ R=2$. Taking all the possible input patterns, we have got:

$\boldsymbol{\sigma}$		$f(\boldsymbol{h})$	$A(\boldsymbol{\sigma})$
0	0	4	0.25
$+$	$+$	2	0.125
$+$	$-$	2	0.125
$+$	0	1	0.0625
$-$	$+$	2	0.125
0	$+$	1	0.0625
$-$	$-$	2	0.125
0	$-$	1	0.0625
$-$	0	1	0.0625

Table 3.3: Accessibilities for $N = 4$, $R = 2$.

	$\boldsymbol{\xi}$				$\longrightarrow$	$\boldsymbol{\sigma}$	
	$-$	$-$	$-$	$-$	$\longrightarrow$	0	0
$\boldsymbol{\xi}^4$	$-$	$-$	$-$	$+$	$\longrightarrow$	$+$	$+$
$\boldsymbol{\xi}^3$	$-$	$-$	$+$	$-$	$\longrightarrow$	$+$	$-$
	$-$	$-$	$+$	$+$	$\longrightarrow$	$+$	0
$\boldsymbol{\xi}^2$	$-$	$+$	$-$	$-$	$\longrightarrow$	$-$	$+$
	$-$	$+$	$-$	$+$	$\longrightarrow$	0	$+$
	$-$	$+$	$+$	$-$	$\longrightarrow$	0	0
	$-$	$+$	$+$	$+$	$\longrightarrow$	$+$	$+$
$\boldsymbol{\xi}^1$	$+$	$-$	$-$	$-$	$\longrightarrow$	$-$	$-$
	$+$	$-$	$-$	$+$	$\longrightarrow$	0	0
	$+$	$-$	$+$	$-$	$\longrightarrow$	0	$-$
	$+$	$-$	$+$	$+$	$\longrightarrow$	$+$	$-$
	$+$	$+$	$-$	$-$	$\longrightarrow$	$-$	0
	$+$	$+$	$-$	$+$	$\longrightarrow$	$-$	$+$
	$+$	$+$	$+$	$-$	$\longrightarrow$	$-$	$-$
	$+$	$+$	$+$	$+$	$\longrightarrow$	0	0

The accessibilities for each resulting $\boldsymbol{\sigma}$ configuration are shown in Table 3.3. As we see from this table

$$\begin{aligned} f(h_i = 0) &= 6 = \binom{4}{2}\,, \; i = 1,2\,, \\ f(h_1 = 0, h_2 = 0) &= 4 = \binom{2}{1}^2\,, \end{aligned}$$

in agreement with the theoretical values (3.51) and (3.60). What is more,

$$f(h_1 \neq 0, h_2 \neq 0) = 8 = 2^4 - \binom{2}{1} f(h_i = 0) + \binom{2}{2} f(h_1 = 0, h_2 = 0)$$

as predicted by (3.80).

b) Example $N = 8$**,** $R = 3$**.** From the results shown in Table 3.4 we have

$$\begin{aligned} f(h_i = 0) &= 70 = \binom{8}{4}\,, \; i = 1,2\,, \\ f(h_i = 0, h_j = 0) &= 36 = \binom{4}{2}^2\,, \; i \neq j\,, \; 1 \leq i,j \leq 2\,, \\ f(h_1 = 0, h_2 = 0, h_3 = 0) &= 18 = \sum_{k=0}^{2} \binom{4}{k}^4\,, \end{aligned}$$

which provide a confirmation of (3.67). From the results is also clear that

$$\begin{aligned} & f(h_1 \neq 0, h_2 \neq 0, h_3 \neq 0) \\ = \; & 136 = 2^8 - \binom{3}{1} f(h_i = 0) + \binom{3}{2} f(h_i = 0, h_j = 0) \\ & - \binom{3}{3} f(h_1 = 0, h_2 = 0, h_3 = 0) \end{aligned}$$

in agreement with (3.80).

c) Example $N = 16$**,** $R = 4$**.** In this case the results given by the simulations are

$$\begin{aligned} f(h_i = 0) &= 12870\,, \\ f(h_i = 0, h_j = 0) &= 4900\,, \\ f(h_i = 0, h_j = 0, h_k = 0) &= 1810\,, \\ f(h_1 = 0, h_2 = 0, h_3 = 0, h_4 = 0) &= 648\,, \end{aligned}$$

which does also provide a check of (3.72). Furthermore, the simulation yields

$$\begin{aligned} & f(h_1 \neq 0, h_2 \neq 0, h_3 \neq 0, h_4 \neq 0) \\ = \; & 36864 = 2^{16} - \binom{4}{1} f(h_i = 0) + \binom{4}{2} f(h_i = 0, h_j = 0) \\ & - \binom{4}{3} f(h_i = 0, h_j = 0, h_k = 0) + \binom{4}{4} f(h_1 = 0, h_2 = 0, h_3 = 0, h_4 = 0)\,, \end{aligned}$$

which offers a new confirmation of (3.80).

$\boldsymbol{\sigma}$			$f(\boldsymbol{h})$	$A(\boldsymbol{\sigma})$
0	0	0	18	0.0703125
$+$	$+$	$+$	17	0.06440625
$+$	$+$	$-$	17	0.06440625
$+$	$+$	0	4	0.015625
$+$	$-$	$+$	17	0.06440625
$+$	0	$+$	4	0.015625
$+$	0	0	9	0.03515625
$+$	$-$	$-$	17	0.06440625
$+$	0	$-$	4	0.015625
$+$	$-$	0	4	0.015625
$-$	$+$	$+$	17	0.06440625
0	$+$	$+$	4	0.015625
0	$+$	0	9	0.03515625
0	0	$+$	9	0.03515625
$-$	$+$	$-$	17	0.06440625
0	$+$	$-$	4	0.015625
0	0	$-$	9	0.03515625
$-$	$+$	0	4	0.015625
$-$	$-$	$+$	17	0.06440625
0	$-$	$+$	4	0.015625
0	$-$	0	9	0.03515625
$-$	0	$+$	4	0.015625
$-$	0	0	9	0.03515625
$-$	$-$	$-$	17	0.06440625
0	$-$	$-$	4	0.015625
$-$	0	$-$	4	0.015625
$-$	$-$	0	4	0.015625

Table 3.4: Accessibilities for $N = 8$, $R = 3$.

Accessibilities at finite temperature.

As we have seen, at zero temperature some of the $\boldsymbol{\xi}$ that do not belong to the set $\{\boldsymbol{\xi}^\mu\}$ can yield $\sigma_j = 0$ for one or more values of j. The chance of having vanishing components makes the number of possible different σ patterns increase from 2^R to 3^R. A way of coping with this is to introduce random noise in the form of finite temperature. Then, the state of the unit σ_j is given by a stochastic function which can take either the value of $+1$ or -1, with probabilities provided by the sigmoid curve

$$P(\sigma_j = \pm 1) = \frac{1}{1 + e^{\mp 2\beta h_j}} \,. \tag{3.82}$$

In the limit where β goes to infinity, this reproduces a deterministic step function, associated to the 0 and 1 'probabilities' (or rather certainties) when taking the sign function, while for $\beta \to 0$ both probabilities tend to $\frac{1}{2}$, i.e. the system behaves absolutely randomly.

If the process is repeated for all the possible input patterns several times, we can consider average values of each $\boldsymbol{\sigma}$ unit for every $\boldsymbol{\xi}$ sequence. Let $\langle \boldsymbol{\sigma} \rangle_{\boldsymbol{\xi}=\boldsymbol{\xi}^\mu}$ denote the average of the $\boldsymbol{\sigma}$ pattern produced by the unary sequence $\boldsymbol{\xi}^\mu$ over many repetitions of the whole reading process. Obviously, the lower T, the closer $\langle \boldsymbol{\sigma} \rangle_{\boldsymbol{\xi}=\boldsymbol{\xi}^\mu}$ will be to $\boldsymbol{\sigma}^\mu$. Therefore, since we are interested in preserving the encoding from $\boldsymbol{\xi}^\mu$ to $\boldsymbol{\sigma}^\mu$ (if not always at least on average) the temperature will have to be low.

At $T > 0$, owing to the absence of vanishing σ_j, the only possible configurations are the $\boldsymbol{\sigma}^\mu$, for $\mu = 1, \ldots, N$. However, for any fixed μ there are $\boldsymbol{\xi}$ other than the $\boldsymbol{\xi}^\mu$ which end up by giving $\boldsymbol{\sigma}^\mu$. With respect to the situation at $T = 0$, the accessibility of each $\boldsymbol{\sigma}^\mu$ necessarily changes, as patterns which produced one or more zeros will now have to 'decide' among $\{\boldsymbol{\sigma}^\mu \,, \; \mu = 1, \ldots, N\}$. Since each realization in itself is a merely stochastic result, the only meaningful quantity to give us an idea of these new accessibilities will be the average over many repetitions, that we define as follows

$$\begin{aligned} \langle A(\boldsymbol{\sigma}^\mu) \rangle \;&\equiv\; \frac{\text{Cumulative no. of input patterns which have given } \boldsymbol{\sigma}^\mu}{\text{Cumulative no. of patterns read}} \\ &=\; \frac{\text{Cumulative no. of input patterns which have given } \boldsymbol{\sigma}^\mu}{\text{No. of repetitions} \times 2^N} \end{aligned} \tag{3.83}$$

The result of a simulation (see Fig. 3.2) for $N = 4$, $R = 2$ shows the tendency of all the accessibilities to be equal as the number of repetitions increases, i.e.

$$\langle A(\boldsymbol{\sigma}^\mu) \rangle \longrightarrow \frac{1}{2^R} \,. \tag{3.84}$$

Contrarily to other memory retrieval systems, this network has no critical temperature. This means that there is no phase transition in the sense that noise

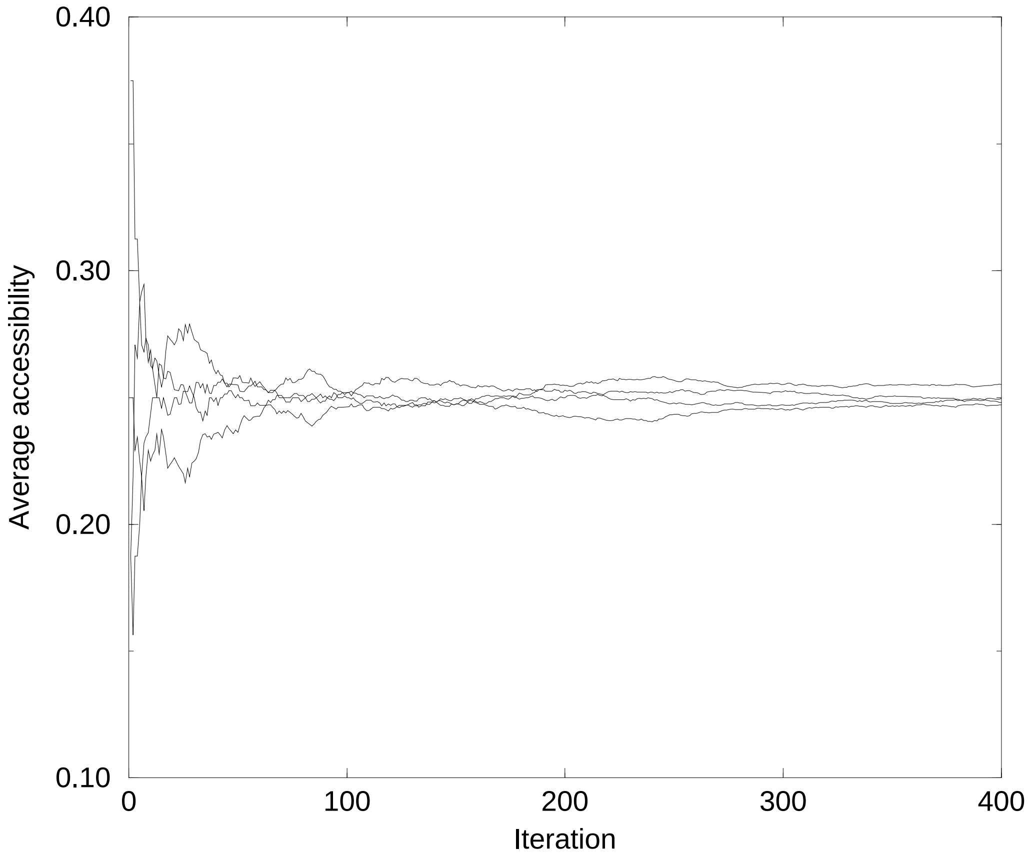

Figure 3.2: Result of a simulation for $N = 4$ at finite $T = 0.05$. The curves represent the cumulative average accesibilities of each $\boldsymbol{\xi}^{\mu}$.

degrades the interactions between processing elements in a continuous way, without leaving any phase where the reproduction of the original process as regards the $\boldsymbol{\xi}^\mu$ can be (on average) exact. By (3.82) we obtain

$$\begin{aligned}\langle\sigma_j\rangle_{\boldsymbol{\xi}=\boldsymbol{\xi}^\mu} &= (+1)\times P(\sigma_j^\mu=+1)+(-1)\times P(\sigma_j^\mu=-1)\\ &= \tanh(\beta h_j^\mu)\\ &= \tanh\left(\beta\left(\sum_k \omega_{jk}\xi_k^\mu-\theta_j\right)\right).\end{aligned} \tag{3.85}$$

With the components of $\boldsymbol{\xi}^\mu$ and the thresholds we are using, this is

$$\begin{aligned}\langle\sigma_j\rangle_{\boldsymbol{\xi}=\boldsymbol{\xi}^\mu} &= \tanh\left(\beta\left(\sum_k \omega_{jk}(2\delta_k^\mu-1)+\sum_k \omega_{jk}\right)\right)\\ &= \tanh(2\beta\omega_{j\mu}).\end{aligned} \tag{3.86}$$

If we look for solutions to

$$\langle\sigma_j\rangle_{\boldsymbol{\xi}=\boldsymbol{\xi}^\mu}=\sigma_j^\mu=\omega_{j\mu}, \tag{3.87}$$

taking into account that for our choice of weights $\omega_{j\mu}$ can be either $+1$ or -1, the equation for β will be in any case

$$1=\tanh(2\beta), \tag{3.88}$$

whose only solution is $\beta\to\infty$, i.e. $T=0$. Thus, in this sense, no critical temperature exists. However, this reasoning allows us to find error bounds. The difference between the average obtained and the desired result will be

$$\begin{aligned}\langle\sigma_j\rangle_{\boldsymbol{\xi}=\boldsymbol{\xi}^\mu}-\sigma_j^\mu &= \tanh(2\beta\sigma_j^\mu)-\sigma_j^\mu\\ &= \begin{cases}\tanh(2\beta)-1 & \text{if } \sigma_j^\mu=+1,\\ -\tanh(2\beta)+1 & \text{if } \sigma_j^\mu=-1.\end{cases}\end{aligned} \tag{3.89}$$

Hence,

$$|\langle\sigma_j\rangle_{\boldsymbol{\xi}=\boldsymbol{\xi}^\mu}-\sigma_j^\mu|=1-\tanh(2\beta). \tag{3.90}$$

If we wish to work in such conditions that

$$|\langle\sigma_j\rangle_{\boldsymbol{\xi}=\boldsymbol{\xi}^\mu}-\sigma_j^\mu|\le\varepsilon, \tag{3.91}$$

for a given ε, by the above relations we find that this temperature must have a value satisfying

$$\beta\ge\frac{1}{4}\log\frac{2-\varepsilon}{\varepsilon}. \tag{3.92}$$

For example, if, at a given moment, we want our average values to be reliable up to the fourth decimal digit, taking $\varepsilon=10^{-5}$ we get $\beta\ge 3.05$ or $T\le 0.33$, which agrees quite fairly with the behaviour observed in our simulations.

3.2 Simple perceptrons

Simple perceptrons constitute the simplest architecture for a layered feed-forward neural network. An input layer feeds the only unit of the second layer, where the output is read. Thus, there are as many weights ω_k as input units (say N) and just one threshold U. Taking the sign as the activation function which decides the final state O of the output unit, it will be given by

$$O = \mathrm{sign}(h) = \begin{cases} -1 & \text{if } h < 0\,, \\ +1 & \text{if } h \geq 0\,, \end{cases} \tag{3.93}$$

where the field h is calculated, as a function of the input pattern $\boldsymbol{\xi}$, through the formula

$$\begin{aligned} h &= \sum_{k=1}^{N} \omega_k \xi_k - U \\ &= \boldsymbol{\omega} \cdot \boldsymbol{\xi} - U\,. \end{aligned} \tag{3.94}$$

Therefore, *supervised learning* with a simple perceptron amounts to finding the weights $\boldsymbol{\omega}$ and the threshold U which map a set of known input patterns $\{\boldsymbol{\xi}^\mu\,,\ \mu = 1, \ldots, p\}$ into their corresponding desired outputs $\{\zeta^\mu\,,\ \mu = 1, \ldots, p\}$.

From now on we will eliminate the constraint that only binary input vectors (such as $\boldsymbol{\xi} \in \{-1, +1\}^N$) are possible, thus admitting as correct input any N-dimensional real vector $\boldsymbol{\xi} \in \mathbb{R}^N$.

3.2.1 Perceptron learning rule

Putting together the expressions (3.93) and (3.94), the output O is simply

$$O(\boldsymbol{\xi}) = \begin{cases} -1 & \text{if } \boldsymbol{\omega} \cdot \boldsymbol{\xi} < U\,, \\ +1 & \text{if } \boldsymbol{\omega} \cdot \boldsymbol{\xi} \geq U\,. \end{cases} \tag{3.95}$$

Eq. (3.95) says that, for any given values of the weights ω_k and the threshold U, the input space $\mathbb{R}^N$ is divided in two zones, one for which the output of all its patterns is -1, and the other with output $+1$. The border between them is the *hyperplane* of equation

$$\boldsymbol{\omega} \cdot \boldsymbol{\xi} = U\,. \tag{3.96}$$

Thus, from a geometrical point of view, a simple perceptron may be regarded simply as a hyperplane which separates the input space into two halves. Moreover, the weight vector $\boldsymbol{\omega}$ is perpendicular to this hyperplane, and it points to the half where the output is $+1$. Making use of this interpretation, supervised learning with a simple perceptron may be viewed just as the search for a hyperplane which separates a set of points of class $+1$ from another set of points of class -1.

In 1962 Rosenblatt proposed a 'Hebb-like' algorithm, known as the *perceptron learning rule*, which could be used to find such hyperplanes. The idea was that, starting from random weights and threshold, they could be modified step by step until all the patterns were correctly classified. In each step a pattern $\boldsymbol{\xi}^\mu$ is presented to the simple perceptron, producing an output O^μ. If $O^\mu = \zeta^\mu$, then $\boldsymbol{\xi}^\mu$ lies in the expected side of the hyperplane, and nothing has to be done. However, if $O^\mu \neq \zeta^\mu$, the hyperplane should be moved in the direction of correcting this mistake:

$$\left\{ \begin{array}{llllll} \text{If} & \zeta^\mu = +1 = -O^\mu & \text{then} & \boldsymbol{\omega} \longrightarrow \boldsymbol{\omega} + \boldsymbol{\xi}^\mu & \text{and} & U \longrightarrow U - 1\,, \\ \text{If} & \zeta^\mu = -1 = -O^\mu & \text{then} & \boldsymbol{\omega} \longrightarrow \boldsymbol{\omega} - \boldsymbol{\xi}^\mu & \text{and} & U \longrightarrow U + 1\,. \end{array} \right. \tag{3.97}$$

A more compact expression for this perceptron learning rule, which also includes a parameter η called the *learning rate*, is

$$\left\{ \begin{array}{lcl} \delta\boldsymbol{\omega} & = & \eta\,(\zeta^\mu - O^\mu)\,\boldsymbol{\xi}^\mu\,, \\ \delta U & = & -\eta\,(\zeta^\mu - O^\mu)\,, \end{array} \right. \tag{3.98}$$

where the symbol δ indicates the variation of the weights and the threshold after the presentation of any pattern, i.e.

$$\left\{ \begin{array}{lcl} \boldsymbol{\omega} & \longrightarrow & \boldsymbol{\omega} + \delta\boldsymbol{\omega}\,, \\ U & \longrightarrow & U + \delta U\,. \end{array} \right. \tag{3.99}$$

The introduction of the learning rate is made in order to adjust the magnitude of the changes in each iteration, which may increase the velocity of the learning process.

A *perceptron convergence theorem* guarantees that the perceptron learning rule always stops after a finite number of steps, provided a solution exists [51]. In fact, among all the possible input-output associations, only the so-called *linearily separable* problems have perceptron solutions. In Fig. 3.3 we have drawn an instance of a linearly separable problem, with ten patterns of class $+1$ (the filled dots) and nine of class -1 (the hollowed dots).

Unfortunately, the discovery of very simple problems which were not linearly separable revealed some of the underlying limitations of the simple perceptrons, putting an end to the study of neural networks in the late 1960s [51]. The exponent of these examples is the well-known *XOR problem*: it is not possible to constuct any simple perceptron capable of performing the *exclusive-OR logical function* of Table 3.5. Looking at Fig. 3.4 it is clear that the XOR function is not linearly separable, but other proofs are possible. For instance, it is easy to realize that

$$\zeta^\mu = \text{sign}(\omega_1 \xi_1^\mu + \omega_2 \xi_2^\mu - U)\,,\ \mu = 1, \ldots, 4$$

leads to an incompatible system of inequations when the XOR function values

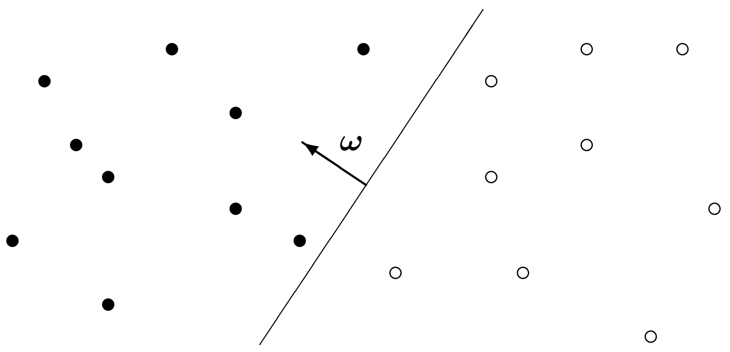

Figure 3.3: Example of a linearly separable set of patterns. The hollowed dots represent patterns whose desired outputs are $\zeta^\mu = -1$, and the filled dots patterns whose desired outputs are $\zeta^\mu = +1$.

μ	$\boldsymbol{\xi}^\mu$	$\longrightarrow$	ζ^μ
1	$(-1,-1)$	$\longrightarrow$	-1
2	$(-1,+1)$	$\longrightarrow$	$+1$
3	$(+1,-1)$	$\longrightarrow$	$+1$
4	$(+1,+1)$	$\longrightarrow$	-1

Table 3.5: The XOR logical function.

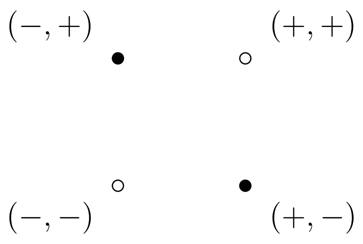

Figure 3.4: The XOR problem. There exists no line capable of separating the hollowed dots (desired outputs $\zeta^\mu = -1$) from the filled dots (desired outputs $\zeta^\mu = +1$).

are substituted:

$$\left.\begin{array}{rcl} -\,\omega_1 - \omega_2 - U & < & 0 \\ -\,\omega_1 + \omega_2 - U & \geq & 0 \\ +\,\omega_1 - \omega_2 - U & \geq & 0 \\ +\,\omega_1 + \omega_2 - U & < & 0 \end{array}\right\}$$

Adding the first and the last inequations you get $U > 0$, while doing the same with the second and the third the result is $U \leq 0$, showing up the incompatibility.

3.2.2 Perceptron of maximal stability

Oftenly, when a set of patterns is linearly separable, the number of possible different hyperplanes which separate them is infinite. Each running of the perceptron learning rule finds out one of them, which depends basically on the initial values given to the weights and the threshold, and on the order in which the patterns are presented to the simple perceptron. Among all the different solutions, however, there is one which has the distinguished features of being unique and more robust than the rest: the *perceptron of maximal stability*.

Let us call $\mathcal{F}_+$ and $\mathcal{F}_-$ the subsets of patterns with desired outputs $\zeta^\mu = +1$ and $\zeta^\mu = -1$, respectively. If $\mathcal{F}_+$ and $\mathcal{F}_-$ are linearly separable, there exist $\boldsymbol{\omega}$ and U such that

$$\left\{\begin{array}{lcl} \forall \boldsymbol{\xi}_-^\rho \in \mathcal{F}_- & \Longrightarrow & \boldsymbol{\omega} \cdot \boldsymbol{\xi}_-^\rho < U\,, \\ \forall \boldsymbol{\xi}_+^\gamma \in \mathcal{F}_+ & \Longrightarrow & \boldsymbol{\omega} \cdot \boldsymbol{\xi}_+^\gamma \geq U\,, \end{array}\right. \tag{3.100}$$

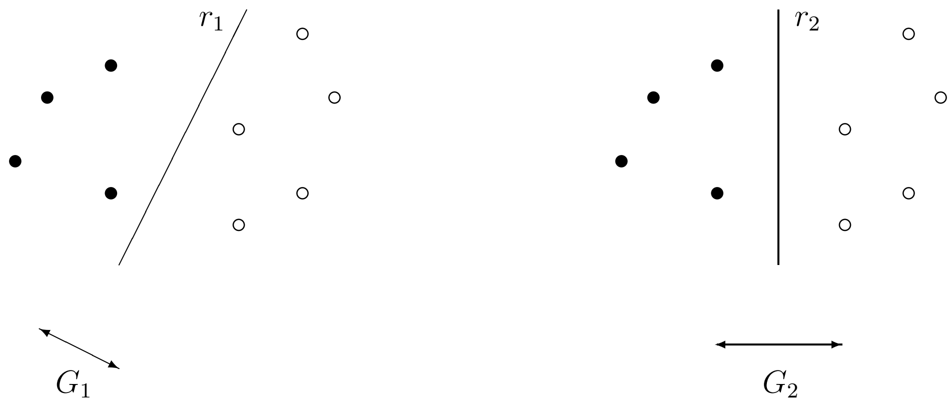

Figure 3.5: Perceptron of maximal stability. Both lines r_1 and r_2 are possible solutions to the problem of separating the five hollowed dots from the four filled dots. However, only the second one constitutes the perceptron of maximal stability, since the gap G_2 is the largest achievable and, therefore, it is larger than G_1.

Now, we can define the *gap* between $\mathcal{F}_+$ and $\mathcal{F}_-$ as the real number

$$G(\boldsymbol{\omega}) \equiv \min_{\rho,\gamma} \left(\frac{\boldsymbol{\omega}}{\|\boldsymbol{\omega}\|} \cdot (\boldsymbol{\xi}_+^\gamma - \boldsymbol{\xi}_-^\rho) \right) , \tag{3.101}$$

which measures the minimum distance between pairs of patterns belonging to different classes, calculated in the direction of $\boldsymbol{\omega}$, i.e. perpendicular to the hyperplane. The perceptron of maximal stability is formed, then, by the weights which minimize the gap $G(\boldsymbol{\omega})$, plus the threshold

$$U \equiv \frac{\max_\rho(\boldsymbol{\omega} \cdot \boldsymbol{\xi}_-^\rho) + \min_\gamma(\boldsymbol{\omega} \cdot \boldsymbol{\xi}_+^\gamma)}{2} , \tag{3.102}$$

which places the hyperplane in the middle of the gap. Fig. 3.5 shows two possible separations of four patterns of $\mathcal{F}_+$ from five patterns of $\mathcal{F}_-$, the second one being the perceptron of maximal stability.

Several procedures have been proposed to get this perceptron of maximal stability. For instance, the MinOver [44] and the AdaTron [2] algorithms mentioned in Subsect. 2.1.4 can be properly modified to achieve it. Nevertheless, recent works have developed fast converging methods based on the techniques of quadratic programming optimization. (e.g. the QuadProg method in [66]).

3.3 Multi-state perceptrons

The simple perceptrons of the previous section divide the input space in two half-spaces, one for each possible value of the output. The problem of classifying in more than two classes with the aid of a collection of perceptrons is well-known in the literature (see e.g. [15]). Likewise, if the mapping to be learned has a continuous output, it can be related to the previous classification scheme in two steps: partition of the interval of variation of the continuous parameter in a finite number of pieces (to arbitrary precision) and assignment of each one to a certain base 2 vector (see [23]). For instance, a 'thermometer' representation for the interval $[0, 1]$ could be

$$\boldsymbol{\zeta} = \begin{cases} (0,0,0,0) & \text{for outputs in } [0, 0.2)\,, \\ (1,0,0,0) & \text{for outputs in } [0.2, 0.4)\,, \\ (1,1,0,0) & \text{for outputs in } [0.4, 0.6)\,, \\ (1,1,1,0) & \text{for outputs in } [0.6, 0.8)\,, \\ (1,1,1,1) & \text{for outputs in } [0.8, 1]\,, \end{cases} \tag{3.103}$$

which reduces the learning problem to a five classes classification one. However, even if this four perceptrons network has learned the thermometer-like $\boldsymbol{\xi}^\mu \longmapsto \boldsymbol{\zeta}^\mu$, $\mu = 1, \ldots, p$ correspondence, new inputs supplied to the net may produce ouputs such as $(0, 0, 1, 1)$ or $(1, 0, 1, 0)$, which cannot be interpreted within this representation; in fact, most of the available codifying schemes suffer from the same inconsistency.

One natural way of avoiding these problematic and rather artificial conversions from continuous to binary data is the use of *multi-state units* perceptrons (see e.g. [16, 55, 62]). With them, only the first of the two steps mentioned above is necessary, i.e. the discretization of the continuous interval. Geometrically, multi-state units define a vector in the input space which points to the direction of increase of the output parameter, the boundaries being parallel hyperplanes. That is why this method gets rid of meaningless patterns, since this partition clearly incorporates the underlying relation of order.

3.3.1 Multi-state perceptron learning rule and convergence theorem

A Q-state neuron may be in anyone of Q different output values or *grey levels* $\sigma_1 < \cdots < \sigma_Q$. They constitute the result of the processing of an incoming stimulus through an activation function of the form

$$g_U(h) \equiv \begin{cases} \sigma_1 & \text{if } h < U_1 \,, \\ \sigma_v & \text{if } U_{v-1} \leq h < U_v \,,\ v = 2, \ldots, Q-1 \,, \\ \sigma_Q & \text{if } U_{Q-1} \leq h \,. \end{cases} \tag{3.104}$$

Therefore, $Q-1$ thresholds $U_1 < \cdots < U_{Q-1}$ have to be defined for each updating unit, which in the case of the simple perceptron is reduced to just the output unit. The field now simply reads

$$h \equiv \boldsymbol{\omega} \cdot \boldsymbol{\xi} \,. \tag{3.105}$$

Let us distribute the input patterns in the following subsets:

$$\mathcal{F}_v \equiv \{\boldsymbol{\xi}^\mu \mid \zeta^\mu = \sigma_v\} \,,\ v = 1, \ldots, Q \,. \tag{3.106}$$

From a geometrical point of view [65] the output processor corresponds to the set of $Q-1$ *parallel hyperplanes*

$$\boldsymbol{\omega} \cdot \boldsymbol{\xi} = U_v \,,\ v = 1, \ldots, Q-1 \,, \tag{3.107}$$

which divide the input space into Q ordered regions, one for each of the grey levels $\sigma_1, \ldots, \sigma_Q$. Thus, the map $\boldsymbol{\xi}^\mu \longmapsto \zeta^\mu$, $\mu = 1, \ldots, p$, is said to be *learnable* or *separable* if it is possible to choose parallel hyperplanes such that each $\mathcal{F}_v$ be in the zone of grey level σ_v (see Fig. 3.6).

This picture make us realize that the fundamental parameters to be searched for while learning are the components of the unit vector

$$\hat{\boldsymbol{\omega}} \equiv \frac{\boldsymbol{\omega}}{\|\boldsymbol{\omega}\|} \tag{3.108}$$

and not the thresholds, since these can be assigned a value as follows. If the input-output map is learnable then

$$\zeta^\mu = g_U(\boldsymbol{\omega} \cdot \boldsymbol{\xi}^\mu) \,,\ \mu = 1, \ldots, p \tag{3.109}$$

yields

$$\left.\begin{array}{r} \forall \boldsymbol{\xi}_v^\rho \in \mathcal{F}_v \\ \forall \boldsymbol{\xi}_{v+1}^\gamma \in \mathcal{F}_{v+1} \end{array}\right\} \implies \boldsymbol{\omega} \cdot \boldsymbol{\xi}_v^\rho < \boldsymbol{\omega} \cdot \boldsymbol{\xi}_{v+1}^\gamma \tag{3.110}$$

which means that, defining $\boldsymbol{\xi}_v^\alpha$ and $\boldsymbol{\xi}_v^\beta$ by

$$\begin{cases} \boldsymbol{\xi}_v^\alpha \in \mathcal{F}_v & \text{such that} \quad \boldsymbol{\omega} \cdot \boldsymbol{\xi}_v^\alpha \geq \boldsymbol{\omega} \cdot \boldsymbol{\xi}_v^\rho \quad \forall \boldsymbol{\xi}_v^\rho \in \mathcal{F}_v \,, \\ \boldsymbol{\xi}_v^\beta \in \mathcal{F}_v & \text{such that} \quad \boldsymbol{\omega} \cdot \boldsymbol{\xi}_v^\beta \leq \boldsymbol{\omega} \cdot \boldsymbol{\xi}_v^\gamma \quad \forall \boldsymbol{\xi}_v^\gamma \in \mathcal{F}_v \,, \end{cases} \tag{3.111}$$

we get

$$U_v \in \left]\boldsymbol{\omega} \cdot \boldsymbol{\xi}_v^\alpha \,, \boldsymbol{\omega} \cdot \boldsymbol{\xi}_{v+1}^\beta\right] \,,\ v = 1, \ldots, Q-1 \,. \tag{3.112}$$

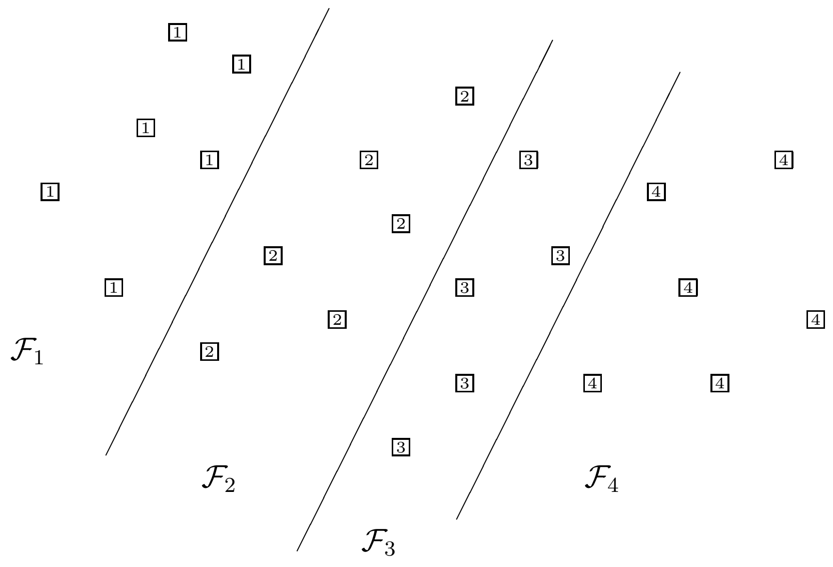

Figure 3.6: Example of a multi-state-separable set of patterns. The four sets of patterns $\mathcal{F}_1$, $\mathcal{F}_2$ $\mathcal{F}_3$ and $\mathcal{F}_4$ are separated by three parallel lines.

Hence, during the learning process it is possible to choose

$$U_v = \frac{\boldsymbol{\omega} \cdot \boldsymbol{\xi}_v^\alpha + \boldsymbol{\omega} \cdot \boldsymbol{\xi}_{v+1}^\beta}{2}\,, \; v = 1, \ldots, Q-1\,, \tag{3.113}$$

which is the best choice for the thresholds with the given $\boldsymbol{\omega}$. Here lies the difference between our approach and that of recent papers such as [49], where the thresholds are compelled to be inside certain intervals given beforehand. Consequently, we have somehow enlarged their notion of learnability.

Our proposal for the *multi-state perceptron learning rule* stems from the following

Theorem: If there exists $\boldsymbol{\omega}^*$ such that $\boldsymbol{\omega}^* \cdot \boldsymbol{\xi}_v^\rho < \boldsymbol{\omega}^* \cdot \boldsymbol{\xi}_{v+1}^\gamma$ for all $\boldsymbol{\xi}_v^\rho \in \mathcal{F}_v$ and $\boldsymbol{\xi}_{v+1}^\gamma \in \mathcal{F}_{v+1}$, $v = 1, \ldots, Q-1$, then the program

```
Start  choose any value for $\boldsymbol{\omega}$ and $\eta > 0$;
Test   choose $v \in \{1, \ldots, Q-1\}$, $\boldsymbol{\xi}_v^\rho \in \mathcal{F}_v$ and $\boldsymbol{\xi}_{v+1}^\gamma \in \mathcal{F}_{v+1}$;
       if $\boldsymbol{\omega} \cdot \boldsymbol{\xi}_v^\rho < \boldsymbol{\omega} \cdot \boldsymbol{\xi}_{v+1}^\gamma$ then go to Test
                                  else go to Add;
Add    replace $\boldsymbol{\omega}$ by $\boldsymbol{\omega} + \eta(\boldsymbol{\xi}_{v+1}^\gamma - \boldsymbol{\xi}_v^\rho)$;
       go to Test.
```

will go to `Add` only a finite number of times.

Corollary: The previous algorithm finds a multi-state perceptron solution to the map $\boldsymbol{\xi}^\mu \longmapsto \zeta^\mu$, $\mu = 1, \ldots, p$ whenever it exists, provided the maximum number of passes through `Add` is reached. This may be achieved by continuously choosing pairs $\{\boldsymbol{\xi}_v^\rho,\ \boldsymbol{\xi}_{v+1}^\gamma\}$ such that $\boldsymbol{\omega} \cdot \boldsymbol{\xi}_v^\rho \geq \boldsymbol{\omega} \cdot \boldsymbol{\xi}_{v+1}^\gamma$.

Proof: Define

$$A(\boldsymbol{\omega}) \equiv \frac{\boldsymbol{\omega} \cdot \boldsymbol{\omega}^*}{\|\boldsymbol{\omega}\|\,\|\boldsymbol{\omega}^*\|} \leq 1\,, \tag{3.114}$$

$$\delta \equiv \min_{v,\rho,\gamma} \left(\boldsymbol{\omega}^* \cdot \boldsymbol{\xi}_{v+1}^\gamma - \boldsymbol{\omega}^* \cdot \boldsymbol{\xi}_v^\rho\right) > 0\,, \tag{3.115}$$

$$M^2 \equiv \max_{v,\rho,\gamma} \left\|\boldsymbol{\xi}_{v+1}^\gamma - \boldsymbol{\xi}_v^\rho\right\|^2 > 0\,. \tag{3.116}$$

On successive passes of the program through `Add`,

$$\boldsymbol{\omega}^* \cdot \boldsymbol{\omega}_{t+1} \geq \boldsymbol{\omega}^* \cdot \boldsymbol{\omega}_t + \eta\delta\,, \tag{3.117}$$

$$\|\boldsymbol{\omega}_{t+1}\|^2 \leq \|\boldsymbol{\omega}_t\|^2 + \eta^2 M^2\,. \tag{3.118}$$

Therefore, after n applications of `Add`,

$$A(\boldsymbol{\omega}_n) \geq L(\boldsymbol{\omega}_n)\,, \tag{3.119}$$

$$L(\boldsymbol{\omega}_n) \equiv \frac{\boldsymbol{\omega}^* \cdot \boldsymbol{\omega}_0 + n\eta\delta}{\|\boldsymbol{\omega}^*\|\sqrt{\|\boldsymbol{\omega}_0\|^2 + n\eta^2 M^2}}\,, \tag{3.120}$$

which for large n goes as

$$L(\boldsymbol{\omega}_n) \approx \sqrt{n}\,\frac{\delta}{\|\boldsymbol{\omega}^*\|M}\,. \tag{3.121}$$

However, n cannot grow at will since $A(\boldsymbol{\omega}) \leq 1$, $\forall \boldsymbol{\omega}$, which implies that the number of passes through `Add` has to be finite.

It is interesting to note that no assumption has been made on the number and nature of the input patterns. Thus, the theorem applies even when an infinite number of pairs of patterns is present and also to inputs not belonging to the 'lattice' $\{\sigma_1, \ldots, \sigma_Q\}^N$.

3.3.2 Multi-state perceptron of maximal stability

In the previous subsection an algorithm for finding a set of parallel hyperplanes which separate the $\mathcal{F}_v$ sets in the correct order has been found, under the assumption that such solutions exist. The problem we are going to address now is that of selecting the 'best' of all such solutions.

It is our precise prescription that the *multi-state perceptron of maximal stability* has to be defined as the one whose smallest gap between the pairs $\{\mathcal{F}_v,\ \mathcal{F}_{v+1}\}$, $v = 1, \ldots, Q-1$ is maximal. These gaps are given by the numbers

$$\begin{aligned} G_v(\boldsymbol{\omega}) \;&\equiv\; \min_{\rho,\gamma}\left(\frac{\boldsymbol{\omega}}{\|\boldsymbol{\omega}\|}\cdot\left(\boldsymbol{\xi}^{\gamma}_{v+1} - \boldsymbol{\xi}^{\rho}_{v}\right)\right) \\ &= \frac{\boldsymbol{\omega}}{\|\boldsymbol{\omega}\|}\cdot\left(\boldsymbol{\xi}^{\beta}_{v+1} - \boldsymbol{\xi}^{\alpha}_{v}\right)\,, \end{aligned} \tag{3.122}$$

where to obtain the second expression we have made use of the definitions in (3.111). Therefore, calling $\mathcal{D} \subset \mathbb{R}^N$ the set of all the solutions to the multi-state perceptron problem, the function to be maximized is

$$G(\boldsymbol{\omega}) \equiv \begin{cases} \displaystyle\min_{v=1,\ldots,Q-1} G_v(\boldsymbol{\omega}) & \text{if } \boldsymbol{\omega} \in \mathcal{D}\,, \\ 0 & \text{if } \boldsymbol{\omega} \notin \mathcal{D}\,. \end{cases} \tag{3.123}$$

In fact, since $G(\lambda\boldsymbol{\omega}) = G(\boldsymbol{\omega})$, $\forall \lambda > 0$, it is actually preferable to restrict the domain of G to the hyper-sphere $S^{N-1} \subset \mathbb{R}^N$, i.e.

$$\begin{aligned} \widetilde{G}\ :\ S^{N-1} &\longrightarrow \mathbb{R}^+ \\ \hat{\boldsymbol{\omega}} &\longmapsto \widetilde{G}(\hat{\boldsymbol{\omega}}) \equiv G(\hat{\boldsymbol{\omega}}) \end{aligned} \tag{3.124}$$

The basic properties of $\widetilde{G}$ are:

1. $\widetilde{G}(\hat{\boldsymbol{\omega}}) > 0 \iff \hat{\boldsymbol{\omega}} \in \mathcal{D} \cap S^{N-1}$.

2. The set $\mathcal{D}$ is convex.

3. The restriction of $\widetilde{G}$ to $\mathcal{D} \cap S^{N-1}$ is a strictly concave function.

4. The restriction of $\widetilde{G}$ to $\mathcal{D} \cap S^{N-1}$ has a unique maximum.

This last property assures the existence and uniqueness of a perceptron of maximal stability, and it is a direct consequence of the preceding propositions. Moreover, it asserts that no other relative maxima are present, which is of great practical interest whenever this optimal perceptron has to be explicitly found.

In [49] the optimization procedure constitutes a forward generalization of the AdaTron algorithm (see Subsect. 2.1.4). Here the situation is much more complicated because the function we want to maximize is not simply quadratic with linear constraints, but a piecewise combination of them due to the previous discrete minimization taken over the gaps. Thus, we have not been able to find a suitable optimization method which could take advantage of the particularities of this problem. Of course, the designing of such converging algorithms is an open question which deserves further investigation.

3.4 Learning with growing architectures

Simple perceptrons, either binary or multi-state, have the limitation that only (multi-state) linearly separable problems can be learnt, as explained in Subsects. 3.2.1 and 3.3.1. Thus, it would be desirable to find new learning rules applicable to networks with hidden layers. Such methods exist, the most important one being the *error back-propagation*. We will explain it in the next chapter. However, back-propagation has the drawback that it can only deal with units whose activation functions are continuous. As a consequence, other strategies have to adopted for the learning of multilayer networks made of discrete units.

In 1989 Mézard and Nadal proposed a completely different approach [50]. Rather than starting from a certain architecture for the network, and then trying to adjust the weights and thresholds according to the set of training patterns, their *tiling algorithm* starts with no neurons, adding them one by one during the learning process. The procedure is:

1. We add a first hidden unit, and train it with the perceptron learning rule. If the training set turns out to be linearly separable, and we have performed a number of iterations large enough, the problem has been solved and no further learning is needed, so we stop. Otherwise, some patterns have been incorrectly classified.

2. Suppose we have already added some neurons to the same hidden layer in which the previous unit is located. It is said that the patterns form a *faithful representation* in this layer if there are no patterns with different desired outputs whose respectives *internal representations* at this level are the same, where the internal representations are the activations induced in

each hidden layer. Thus, if the representation is unfaithful, there exists a subset of patterns which produce the same internal representation, so we proceed to add a new unit to this layer, and train it, using once again the perceptron learning rule, with this subset.

3. When the hidden layer ends with a faithful representation, a new layer is started, going back to the first step.

Instead of using the perceptron learning rule as it is, Mézard and Nadal applied a variant known as the *pocket algorithm* [23]. The only difference lies in the way in which the weights are stored, which allows one to find a solution with a small number of errors whenever the set of patterns is not linearly separable.

Mézard and Nadal proved that this method converges in the sense that it always finds an architecture which correctly evaluates any boolean function with a single binary output. In practice, we have tested the tiling algorithm with several two-state valued functions with real variables, and it has also converged in most of the cases.

Although all the hidden layers seem to play the same role, i.e. that of producing a faithful representation of the internal states of the previous layer, the *first hidden layer* is rather special: each of its units is a hyperplane, all together defining a *tiling of the input space*. The important thing is that all the input patterns belonging to the same 'tile' obtain always the same output, no matter how many layers or units separate the first hidden layer from the output units. In consequence, all the network structure beyond the first hidden layer only serves for the purpose of assigning an output to each tile, without any capability of modifying the shape of the tiling. This fact is crucial, since it suggests that any learning method has to concentrate its efforts in the construction of the first hidden layer, and not in the rest, specially in order to improve the generalization ability of the network. This property is completely general and independent of the learning method (provided the activation functions were discrete).

In Fig. 3.7 we show the tiling of the input space obtained after the application of the tiling algorithm to the learning of a two-state function with two real variables. The training set contained 500 input patterns distributed uniformly over the rectangle $[-1,+1]\times[0,1]$, and whose desired outputs were $+1$ or -1 depending on whether they were located outside or inside the two solid curves. The resulting architecture was 2:5:1 (two input, five hidden and one output units).

Our implementation of the tiling algorithm includes several enhancements. For example, we repeat the building of the first hidden layer several times, preserving only the one with the lowest number of units. The objective is to increase the generalization capability of the network, since it is well-known that the smaller the number of parameters the better the performance of any fitted function (the condition that the number of parameters is large enough is guaranteed by the fact that the tiling always ends with all the patterns being correctly classified).

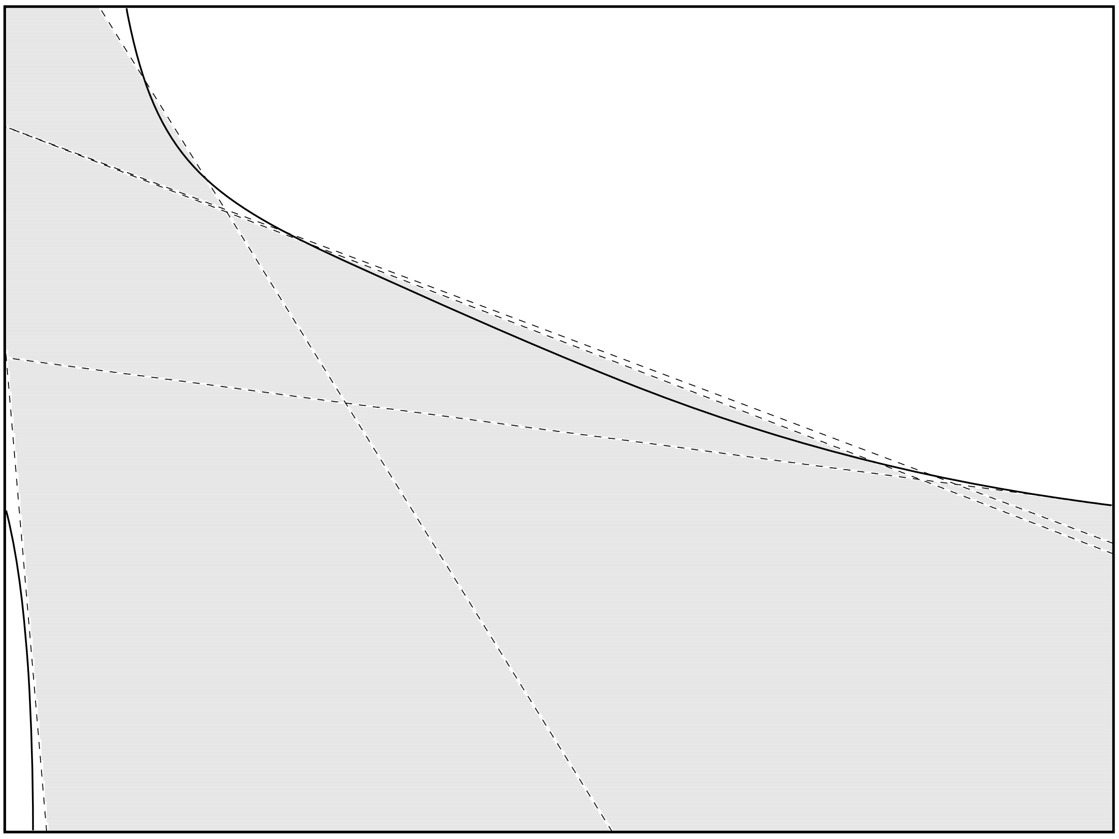

Figure 3.7: Example of a tiling of the input space. The five dashed lines correspond to the five units built by the tiling algorithm. The learnt network assigns an output -1 to the shadowed region, and $+1$ to the rest of the space. This result is in good agreement with the theoretical limits marked out by the two curved solid lines.

It is unnecessary to optimize the size of the rest of the layers since, as has been said above, they do not modify the shape of the tiling of the input space.

A second modification affects the standard pocket algorithm. We found that, when the learning is made with non-binary input patterns (i.e. not belonging to the vertices of a hypercube), there are times in which the solutions with a minimum number of errors are hyperplanes laying outside the training set, so the hyperplane assigns the same output to all the input patterns. For instance, a pattern of class $+1$ rounded in all directions only by patterns of class -1 have this property. When this happens, the tiling algorithm enters an infinite loop, adding units endlessly to the same hidden layer. Thus, we impose that each new unit (or hyperplane) has to 'cross' its training set, dividing it in two non-empty subsets. We do that by changing the threshold until at least one pattern is separated from the rest.

Another improvement consists in the use of multi-state units replacing the usual binary neurons. In principle, the designing of a multi-state version of the pocket algorithm is straightforward. However, our treatment of the thresholds gives rise to some difficulties. Namely, it is clear that eq. (3.113) is not necessarily the optimal way of calculating the thresholds when the training set is not multi-state linearly separable (as happens oftenly during the building of the network), since the solution with the minimum number of errors may have a completely different aspect. Therefore, we decided to choose the thresholds randomly within certain intervals defined from the numbers $\boldsymbol{\omega} \cdot \boldsymbol{\xi}_v^\alpha$ and $\boldsymbol{\omega} \cdot \boldsymbol{\xi}_{v+1}^\beta$, letting the pocket algorithm itself find the best values for them.

We have evaluated the performance of our version of the tiling algorithm when trained with random boolean functions, for different values of the number of input units n_1 and of the grey levels Q. We skipped the repetitions of the building of the first hidden layer, since for boolean functions the concept of generalization looses its meaning. In Table 3.6 we show the results for $n_1 = 4$ and $n_1 = 6$ with the standard $Q = 2$, and those for $n_1 = 3$ with $Q = 3$, and in Table 3.7 there are the figures for $n_1 = 2$ and $n_1 = 3$ with $Q = 4$, and those for $n_1 = 2$ with $Q = 5$. The number of patterns is given by Q^{n_1}. In all the cases the averages are taken over 100 random boolean functions.

The main limitation of the tiling algorithm is its inability to cope with noisy patterns. That is, if the classes we want to separate are distributed in such a way that their domains have a non-null overlap, the tiling will try to learn each single pattern of that region, thus putting aside patterns which should not be separated. We express this property saying that the tiling is good for the learning of functions, but not of probability distributions.

Other learning methods using growing architectures are the *sequential learning* of Marchand, Golea and Ruján [47], the *growth algorithm for neural network decision trees* of Golea and Marchand [28], the method of Nadal in [54], the *upstart algorithm* of Frean [22] and the *cascade correlation* of Fahlman and Lebiere [20].

	$Q = 2\,,\ n_1 = 4$	$Q = 2\,,\ n_1 = 6$	$Q = 3\,,\ n_1 = 3$
$\langle L\rangle$	3.00±0.03	4.07±0.07	4.35±0.08
$\langle n_2\rangle$	2.78±0.07 (100%)	9.73±0.13 (100%)	7.53±0.16 (100%)
$\langle n_3\rangle$	1.04±0.02 (96%)	3.77±0.16 (100%)	2.88±0.11 (100%)
$\langle n_4\rangle$	1.00±0.00 (4%)	1.28±0.05 (90%)	1.68±0.08 (91%)
$\langle n_5\rangle$		1.00±0.00 (21%)	1.09±0.04 (45%)
$\langle n_6\rangle$			1.00±0.00 (3%)

Table 3.6: Tiling learning of random boolean functions with $Q = 2$ and $Q = 3$. For each case we show the average number of layers and the average number of units in each hidden layer. For each layer the average is only over the number of trials which have produced that layer (some trials may have ended in a previous one). The numbers in parentheses give these percentages.

	$Q = 4\,,\ n_1 = 2$	$Q = 4\,,\ n_1 = 3$	$Q = 5\,,\ n_1 = 2$
$\langle L\rangle$	4.39±0.11	9.47±0.15	6.39±0.16
$\langle n_2\rangle$	5.85±0.13 (100%)	14.27±0.22 (100%)	8.93±0.15 (100%)
$\langle n_3\rangle$	2.28±0.10 (100%)	6.45±0.12 (100%)	3.64±0.09 (100%)
$\langle n_4\rangle$	1.71±0.08 (83%)	7.00±0.15 (100%)	3.39±0.11 (100%)
$\langle n_5\rangle$	1.43±0.08 (46%)	5.68±0.11 (100%)	2.44±0.11 (95%)
$\langle n_6\rangle$	1.25±0.04 (12%)	5.23±0.12 (100%)	1.89±0.10 (76%)
$\langle n_7\rangle$	1.00±0.00 (3%)	4.27±0.14 (100%)	1.51±0.08 (45%)
$\langle n_8\rangle$		2.82±0.13 (99%)	1.73±0.06 (15%)
$\langle n_9\rangle$		2.20±0.12 (81%)	1.40±0.07 (10%)
$\langle n_{10}\rangle$		1.51±0.08 (51%)	1.33±0.06 (3%)
$\langle n_{11}\rangle$		1.42±0.08 (19%)	2.00±0.00 (1%)
$\langle n_{12}\rangle$		1.20±0.04 (5%)	2.00±0.00 (1%)
$\langle n_{13}\rangle$		1.00±0.00 (1%)	1.00±0.00 (1%)

Table 3.7: Tiling learning of random boolean functions with $Q = 4$ and $Q = 5$.

Chapter 4

Supervised learning with continuous activation functions

When continuous and differentiable activation functions are used, a multilayer neural network becomes a differentiable map from a n-dimensional real input space into a m-dimensional output one. Thus, it may seem that this sort of nets do not deserve more attention than any other class of differentiable functions. However, the discovery of the *error back-propagation* algorithm to train multilayer networks from examples has proved to be an excellent tool in classification, interpolation and prediction tasks. In fact, it has been proved that any sufficiently well-behaved function can be approximated by a neural network provided the number of units is large enough. In this chapter we will explain the original and several variants of the back-propagation method, and some of the applications we have developed. Moreover, we will give an analytical interpretation of the net outputs obtained after the training.

4.1 Learning by error back-propagation

4.1.1 Back-propagation in multilayer neural networks

Let us consider the set

$$\{(\boldsymbol{x}^\mu, \boldsymbol{z}^\mu) \in \mathbb{R}^n \times \mathbb{R}^m \,,\ \mu = 1, \ldots, p\} \tag{4.1}$$

of pairs input-output and a multilayer neural network of the kind of that in Fig. 1.3, which has L layers with $n_1, \ldots, n_L$ units respectively ($n = n_1$ and $m = n_L$). Now the architecture is given beforehand, and it is not changed during the learning phase. The equations governing the state of the net are the recursive relations

$$\xi_i^{(\ell)} = g(h_i^{(\ell)}) \,,\ i = 1, \ldots, n_\ell \,,\ \ell = 2, \ldots, L \,, \tag{4.2}$$

where the fields are given by

$$h_i^{(\ell)} = \sum_{j=1}^{n_{\ell-1}} \omega_{ij}^{(\ell)} \xi_j^{(\ell-1)} - \theta_i^{(\ell)} . \tag{4.3}$$

We shall use a different notation for the input and output patterns:

$$\left\{ \begin{array}{rcl} \boldsymbol{x} & = & \boldsymbol{\xi}^{(1)} , \\ \boldsymbol{o}(\boldsymbol{x}) & = & \boldsymbol{\xi}^{(L)} . \end{array} \right. \tag{4.4}$$

Batched and online back-propagation

For any given values of the weights and thresholds it is possible to calculate the *quadratic error* between the actual and the desired output of the net, measured over the training set:

$$E[\boldsymbol{o}] \equiv \frac{1}{2} \sum_{\mu=1}^{p} \sum_{i=1}^{m} \left(o_i(\boldsymbol{x}^\mu) - z_i^\mu \right)^2 . \tag{4.5}$$

Therefore, the least squares estimate is that which minimizes $E[\boldsymbol{o}]$. Applying the gradient descent minimization procedure, what we have to do is just to look for the direction (in the space of weights and thresholds) of steepest descent of the error function (which coincides with minus the gradient), and then modify the parameters in that direction so as to decrease the actual error of the net:

$$\left\{ \begin{array}{rcl} \delta\omega_{ij}^{(\ell)} & = & -\eta \dfrac{\partial E}{\partial \omega_{ij}^{(\ell)}} , \quad i = 1, \ldots, n_\ell \, , \; j = 1, \ldots, n_{\ell-1} \, , \\ \\ \delta\theta_i^{(\ell)} & = & -\eta \dfrac{\partial E}{\partial \theta_i^{(\ell)}} , \quad i = 1, \ldots, n_\ell \, , \end{array} \right. \tag{4.6}$$

with the updating rule

$$\left\{ \begin{array}{rcl} \omega_{ij}^{(\ell)} & \longrightarrow & \omega_{ij}^{(\ell)} + \delta\omega_{ij}^{(\ell)} , \\ \\ \theta_i^{(\ell)} & \longrightarrow & \theta_i^{(\ell)} + \delta\theta_i^{(\ell)} . \end{array} \right. \tag{4.7}$$

The intensity of the change is controlled by the *learning rate* parameter η, which plays the same role here than in the perceptron learning rule of Subsect. 3.2.1.

Substituting eqs. (4.2) and (4.3) into (4.5), and taking the derivatives, it is easy to get (see e.g. [68])

$$\left\{ \begin{array}{rcl} \delta\omega_{ij}^{(\ell)} & = & -\eta \displaystyle\sum_{\mu=1}^{p} \Delta_i^{(\ell)\mu} \, \xi_j^{(\ell-1)\mu} , \\ \\ \delta\theta_i^{(\ell)} & = & \eta \displaystyle\sum_{\mu=1}^{p} \Delta_i^{(\ell)\mu} , \end{array} \right. \tag{4.8}$$

where the error is introduced in the units of the last layer through

$$\Delta_i^{(L)\mu} = g'(h_i^{(L)\mu})\,[o_i(\boldsymbol{x}^\mu) - z_i^\mu]\,, \tag{4.9}$$

and then is *back-propagated* to the rest of the network by

$$\Delta_j^{(\ell-1)\mu} = g'(h_j^{(\ell-1)\mu}) \sum_{i=1}^{n_\ell} \Delta_i^{(\ell)\mu}\, \omega_{ij}^{(\ell)}\,. \tag{4.10}$$

The appearence of the derivative g' of the activation function explains why we have supposed in advance that it has to be continuous and differentiable.

Summarizing, the *batched back-propagation* algorithm for the learning of the training set (4.1) consists in the following steps:

1. Initialize all the weights and thresholds randomly, and choose a small value for the learning rate η.
2. Run a pattern $\boldsymbol{x}^\mu$ of the training set using eqs. (4.2) and (4.3), and store the activations of all the units (i.e. $\{\xi_i^{(\ell)\mu}\,,\ \forall\ell\ \forall i\}$).
3. Calculate the $\Delta_i^{(L)\mu}$ with eqs. (4.9), and then back-propagate the error using eqs. (4.10).
4. Compute the contributions to $\delta\omega_{ij}^{(\ell)}$ and to $\delta\theta_i^{(\ell)}$ induced by this input-output pair $(\boldsymbol{x}^\mu, \boldsymbol{z}^\mu)$.
5. Repeat the last three steps until all the training patterns have been used.
6. Update the weights and thresholds using eqs. (4.7).
7. Go to the second step unless enough *training epochs* have been carried out.

The adjective 'batched' refers to the fact that the update of the weights and thresholds is done after all the patterns have been presented to the network. Nevertheless, simulations show that, in order to speed up the learning, it is usually preferable to perform this update each time a new pattern is processed, choosing them in random order: this is known as *non-batched* or *online back-propagation*.

Momentum term

It is clear that back-propagation seeks minimums of the error function (4.5), but it cannot assure that it ends in a global one, since the procedure may get stucked in a *local minimum*. Several modifications have been proposed to improve the algorithm so as to avoid these undesired local minimums and to accelerate its convergence. One of the most successful, simple and commonly used variants is the introduction of a *momentum term* to the updating rule, either in the batched or the online schemes. It consists in the substitution of (4.6) by

$$\left\{\begin{array}{rcl} \delta\omega_{ij}^{(\ell)} & = & -\eta\,\dfrac{\partial E}{\partial\omega_{ij}^{(\ell)}} + \alpha\,\delta\omega_{ij}^{(\ell)}{}_{\text{(last)}}\,, \\[3ex] \delta\theta_{i}^{(\ell)} & = & -\eta\,\dfrac{\partial E}{\partial\theta_{i}^{(\ell)}} + \alpha\,\delta\theta_{i}^{(\ell)}{}_{\text{(last)}}\,, \end{array}\right. \tag{4.11}$$

where ‘last’ means the values of the $\delta\omega_{ij}^{(\ell)}$ and $\delta\theta_i^{(\ell)}$ used in the previous updating of the weights and thresholds. The parameter α is called the *momentum* of the learning, and it has to be a positive number smaller than 1.

Local learning rate adaptation

For most of the applications online back-propagation (with or without a momentum term) suffices. However, lots of variants may be found in the literature (see [74] for a comparative study), some of them quite interesting. For instance, Silva and Almeida proposed a *local learning rate adaptation* procedure which works well in many situations. The main idea is the use of separate learning rates for each of the parameters to be adjusted, and then to increase or decrease their values according to the signs of the last two gradients. More precisely, the set (4.6) has to be substituted by

$$\left\{\begin{array}{rcl} \delta\omega_{ij}^{(\ell)} & = & -\eta_{ij}^{(\ell)}\,\dfrac{\partial E}{\partial\omega_{ij}^{(\ell)}}\,, \quad i=1,\ldots,n_\ell\,,\ j=1,\ldots,n_{\ell-1}\,, \\[3ex] \delta\theta_{i}^{(\ell)} & = & -\eta_{i}^{(\ell)}\,\dfrac{\partial E}{\partial\theta_{i}^{(\ell)}}\,, \quad i=1,\ldots,n_\ell\,, \end{array}\right. \tag{4.12}$$

and a new step has to be added to the main scheme just before the updating of the weights and thresholds, which reads

$$\left\{\begin{array}{rcl} \eta_{ij}^{(\ell)} & = & \left\{\begin{array}{ll} u\,\eta_{ij}^{(\ell)}{}_{\text{(last)}} & \text{if } \dfrac{\partial E}{\partial\omega_{ij}^{(\ell)}}\,\dfrac{\partial E}{\partial\omega_{ij}^{(\ell)}}{}_{\text{(last)}} \geq 0\,, \\ d\,\eta_{ij}^{(\ell)}{}_{\text{(last)}} & \text{otherwise}\,, \end{array}\right. \\[5ex] \eta_{i}^{(\ell)} & = & \left\{\begin{array}{ll} u\,\eta_{i}^{(\ell)}{}_{\text{(last)}} & \text{if } \dfrac{\partial E}{\partial\theta_{i}^{(\ell)}}\,\dfrac{\partial E}{\partial\theta_{i}^{(\ell)}}{}_{\text{(last)}} \geq 0\,, \\ d\,\eta_{i}^{(\ell)}{}_{\text{(last)}} & \text{otherwise}\,, \end{array}\right. \end{array}\right. \tag{4.13}$$

where the parameters $u > 1$ and $0 < d < 1$ could be chosen, for example, as $u = \frac{1}{d} = 1.2$. Thus, if two successive gradients have the same sign, the local learning rate is increased (we are still far from the minimum, so we want to reach it sooner), and if the signs are the opposite, it is decreased (we have crossed over the minimum, so we have to do the search with smaller movements).

Back-propagation with discrete networks

In the previous chapter we discussed the problem of supervised learning with discrete activation functions, but we did not provide any learning algorithm for multilayer networks: the perceptron learning rule or the pocket algorithm can only deal with simple perceptrons, while the tiling algorithm generates its own architecture. Since eq. (1.6) shows that the sigmoids are approximations to the step function, one possible way out consists in the realization of the training using back-propagation (with sigmoidal activation functions), and when it is finished we use the obtained weights and thresholds as if they were the solution in the discrete case.

We have studied a very simple problem to compare the performance of this 'discretized' back-propagation with the tiling algorithm, and also with the non-discretized back-propagation. It consists in the discrimination between patterns inside and outside a ring, centered in the square $[-1, 1]^2$. A number of points, ranging from 50 to 500, were randomly generated in the square, and the desired output is assigned to be 1 if the point is inside the ring, and 0 otherwise. The radii of the ring were chosen as 0.3 and 0.7. After the learning, the solutions found were tested with 10 000 new patterns, and the proportion of successfully classified patterns, called *generalization*, was stored. Fig. 4.1 shows the mean of the generalization over 25 realizations for different learning parameters and architectures. The best behaviour corresponds, as expected, to continuous back-propagation, since discrete networks cannot produce 'curved tilings' of the input space. In the discrete case, the tiling algorithm always works much better than the discretized back-propagation. We believe that this fact is due to the ability of the tiling algorithm to find out the right number of units in the first hidden layer, which in the discretized back-propagation has to be 'guessed and set' in advance.

4.1.2 Back-propagation in general feed-forward neural nets

The back-propagation algorithm of the previous subsection is applicable not only to multilayer neural networks but also to a larger class of architectures, which we will refer to as *general feed-forward neural networks*. In this class the units are also distributed in layers, but there may be also connections between non-consecutive layers. For instance, some weights may connect the input neurons with the ones in the second, third and fourth layers, but 'lateral' weights within a single layer or connections to previous layers are still forbidden.

Suppose that we have a general feed-forward neural network made of N units, and that we have numbered them in the order in which they are updated. The condition of being feed-forward means that the i-th neuron can only receive signals from the previous neurons, i.e. only weights ω_{ij} satisfying $i > j$ are possible. Let us call $\mathcal{J}_i$ the set of units with weights which end in the i-th one. With this

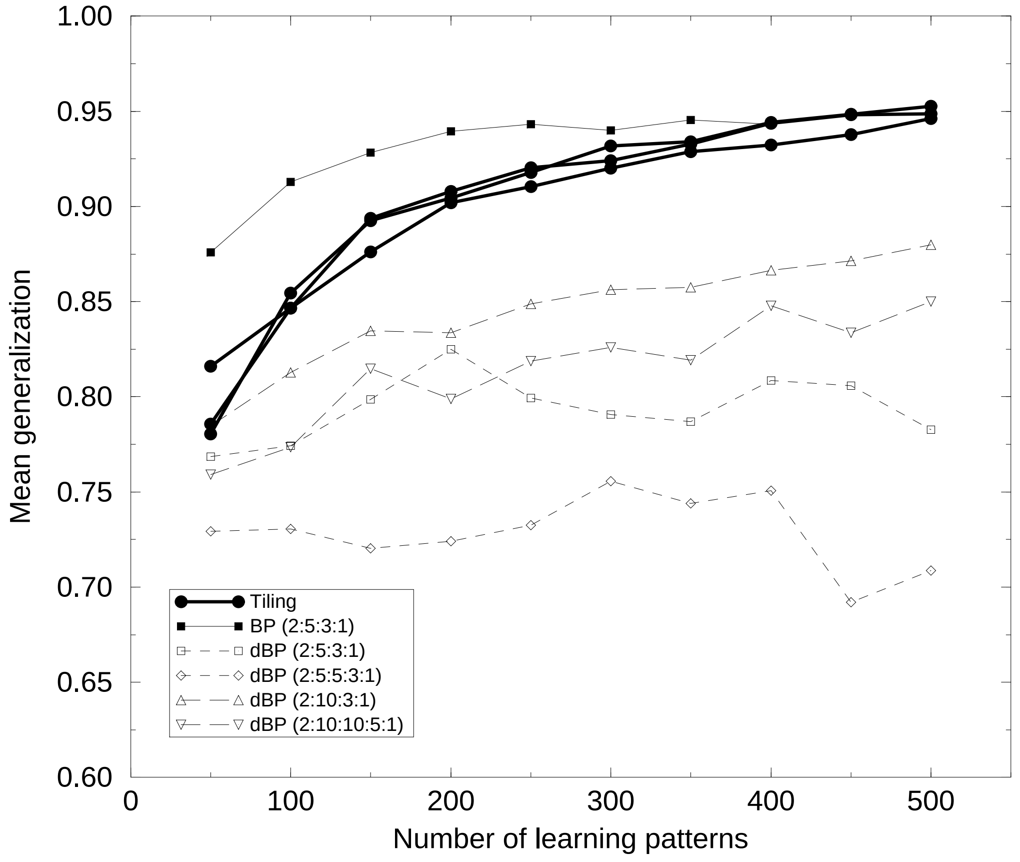

Figure 4.1: A comparison between the tiling algorithm, back-propagation (BP) and a discretized back-propagation (dBP).

notation, the state of the network is given by

$$\xi_i = g_i(h_i)\,,\ i = 1, \ldots, N\,, \tag{4.14}$$

where the fields are

$$h_i = x_i\, 1_{\{i \in \mathcal{I}\}} + \sum_{j \in \mathcal{J}_i} \omega_{ij}\, \xi_j - \theta_i\,. \tag{4.15}$$

Notice that we have enlarged the definition of the network in two ways: we let each unit have a different activation function g_i, and the inputs $\{x_i\,,\ i \in \mathcal{I}\}$ are considered as external additive fields which can enter the network at any unit. A standard input unit is recovered if it has no incoming weights, its threshold is null and its activation function is the identity ($\xi_i = g_i(h_i) = g_i(x_i) = x_i\,,\ i \in \mathcal{I}$). The symbol $1_{\{i \in \mathcal{I}\}}$ has been introduced to emphasize that only the neurons numbered in the set $\mathcal{I}$ have external inputs.

A further generalization consits in the possibility of 'reading' the output anywhere in the network: $\{o_i(\boldsymbol{x})\,,\ i \in \mathcal{O}\}$. Hence, the error function acquires the form

$$E[\boldsymbol{o}] \equiv \frac{1}{2} \sum_{\mu=1}^{p} \sum_{i \in \mathcal{O}} \left(o_i(\boldsymbol{x}^\mu) - z_i^\mu\right)^2\,. \tag{4.16}$$

Calling $\mathcal{K}_j$ the set of units which receive a connection from the j-th one, the formulas of the batched back-propagation algorithm with momentum are

$$\left\{ \begin{array}{rcll} \delta\omega_{ij} & = & -\eta \displaystyle\sum_{\mu=1}^{p} \Delta_i^\mu\, \xi_j^\mu + \alpha\, \delta\omega_{ij(\text{last})}\,, & j \in \mathcal{J}_i\,,\ i = 1, \ldots, N\,, \\ \delta\theta_i & = & \eta \displaystyle\sum_{\mu=1}^{p} \Delta_i^\mu + \alpha\, \delta\theta_{i(\text{last})}\,, & i = 1, \ldots, N\,, \end{array} \right. \tag{4.17}$$

where eqs. (4.9) and (4.10) have to be replaced by

$$\Delta_j^\mu = g_i'(h_j^\mu) \left[\sum_{i \in \mathcal{K}_j} \Delta_i^\mu\, \omega_{ij} + D_j^\mu\, 1_{\{j \in \mathcal{O}\}} \right]\,,\ j = N, \ldots, 1\,, \tag{4.18}$$

and the output errors are introduced through

$$D_j^\mu = o_j(\boldsymbol{x}^\mu) - z_j^\mu\,,\ j \in \mathcal{O}\,. \tag{4.19}$$

The online version is recovered by elimination of the summatories over the training set.

4.1.3 Back-propagation through time

The algorithms in the previous subsections are useful for the learning of *static* pairs input-output. However, in prediction tasks it is usually necessary to deal with *time series*. For instance, the demand of consumer and industrial goods depends, among other factors, on the evolution of the market during the last hours, days, weeks, months or years. Thus, it would be desirable to introduce into the training patterns all such information.

Let $\{\boldsymbol{s}_t\,,\ t \in \mathbb{N}\}$ be a time series, so the value of $\boldsymbol{s}_t$ depends on those of $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_{t-1}$ and, probably, on some other unknown variables and on some noise. The easiest solution consists in the use of ordinary multilayer neural networks, with an input layer which permits the introduction of k consecutive elements of the series, $\boldsymbol{x}^\mu = (\boldsymbol{s}_{\mu-k}, \ldots, \boldsymbol{s}_{\mu-1})$, and an output one where the desired output is $\boldsymbol{z}^\mu = \boldsymbol{s}_\mu$. For example, in [79] Weigend, Rumelhart and Huberman apply this method (which we will refer to as *time-delay neural networks*), to the prediction of the well-known 'sunspots' time series.

Nevetheless, time-delay neural networks have several important drawbacks which may complicate their application to real problems. First, the delay k has to be chosen in advance, even if we do not know which value is the more efficient one. And if the optimal k happens to be too big, the size of the network may become too large rendering the learning not possible. Moreover, all the training patterns have to have the same delay.

An alternative to time-delay nets is the use of *recurrent neural networks*. In a recurrent net the connections within a single layer or ending in a previous one are allowed. For instance, a fully connected network is a special case of a recurrent one. It is easy to realize that recurrent networks are equivalent to feed-forward ones which include as many copies of the initial basic structure as time steps are being considered. This *unfolding of time* gives rise to the *back-propagation through time* algorithm [68].

Before the explanation of our version of the algorithm, let us consider the network of Fig. 4.2. There are three layers, the first and the third being the input and the output ones respectively. The hidden layer is recurrent, in the sense that its state depends not only on the inputs from the first layer but also on its own state in the last time step. The unfolding of this net for three time steps is given in Fig. 4.3. It shows that, for each time t, all the neurons of the net have to be updated in the usual form, but with some units receiving additional signals from the previous time step. Thus, we distinguish two types of weights: the standard feed-forward ones (represented by horizontal arrows) and the weights connecting states at different consecutive times (the vertical arrows).

More involved examples could be given, e.g. having connections between states at non-consecutive times, or such that a single update of all the units requires more than one time step (this happens if one considers that units at different

$$\begin{tikzcd}
\boldsymbol{\xi}^{(1)} \arrow[r] & \boldsymbol{\xi}^{(2)} \arrow[r] \arrow[loop above] & \boldsymbol{\xi}^{(3)}
\end{tikzcd}$$

Figure 4.2: Example of a multilayer neural network with a recurrent layer.

$$\begin{tikzcd}
\boldsymbol{\xi}^{(1)} \arrow[r] & \boldsymbol{\xi}^{(2)} \arrow[r] & \boldsymbol{\xi}^{(3)} & t = 3 \\
\boldsymbol{\xi}^{(1)} \arrow[r] & \boldsymbol{\xi}^{(2)} \arrow[r] \arrow[u] & \boldsymbol{\xi}^{(3)} & t = 2 \\
\boldsymbol{\xi}^{(1)} \arrow[r] & \boldsymbol{\xi}^{(2)} \arrow[r] \arrow[u] & \boldsymbol{\xi}^{(3)} & t = 1
\end{tikzcd}$$

Figure 4.3: Unfolding of the network of Fig. 4.2 for three time steps.

layers are updated at different time steps). Nonetheless, these complications are rather artificial since the combination of an ordered updating of all the units and the existence of one time delayed weights suffice to give sense to any conceivable architecture. Thus, our back-propagation through time gets rid of the presence of other kinds of weights different to the two ones described above.

Let $\{\omega_{ij}\,,\ j\in\mathcal{J}_i\,,\ i=1,\ldots,N\}$ denote the standard feed-forward weights of the net ($j<i$), and let $\{\overline{\omega}_{ij}\,,\ j\in\overline{\mathcal{J}}_i\,,\ i=1,\ldots,N\}$ be the set of one step *delayed weights* (no restrictions apply to the sets $\overline{\mathcal{J}}_i$). Then, the evolution of the network for training series of length T is controlled by

$$\xi_i(t)=g_i(h_i(t))\,,\ i=1,\ldots,N\,,\ t=1,\ldots,T\,, \tag{4.20}$$

with fields given by

$$h_i(t)=x_i(t)\,1_{\{i\in\mathcal{I}(t)\}}+\sum_{j\in\mathcal{J}_i}\omega_{ij}\,\xi_j(t)+\sum_{j\in\overline{\mathcal{J}}_i}\overline{\omega}_{ij}\,\xi_j(t-1)-\theta_i\,. \tag{4.21}$$

A further initial condition is needed:

$$\xi_i(0)=0\,,\ i=1,\ldots,N\,. \tag{4.22}$$

The fact that the sets $\mathcal{I}(t)$ are not always the same means that we admit that the input units be different at each time step. The same is valid for the output units and the corresponding sets $\mathcal{O}(t)$.

The error function has now to take into account the time evolution:

$$E[\boldsymbol{o}]\equiv\frac{1}{2}\sum_{\mu=1}^{p}\sum_{t=1}^{T}\sum_{i\in\mathcal{O}(t)}\left(o_i(\boldsymbol{x}^\mu,t)-z_i^\mu(t)\right)^2\,. \tag{4.23}$$

Therefore, the formulas for the batched *back-propagation through time* with momentum terms are

$$\left\{\begin{array}{lcll}
\delta\omega_{ij} & = & -\eta\displaystyle\sum_{\mu=1}^{p}\sum_{t=1}^{T}\Delta_i^\mu(t)\,\xi_j^\mu(t)+\alpha\,\delta\omega_{ij(\mathrm{last})}\,, & j\in\mathcal{J}_i\,,\ i=1,\ldots,N\,,\\
\delta\overline{\omega}_{ij} & = & -\eta\displaystyle\sum_{\mu=1}^{p}\sum_{t=1}^{T}\Delta_i^\mu(t)\,\xi_j^\mu(t-1)+\alpha\,\delta\overline{\omega}_{ij(\mathrm{last})}\,, & j\in\overline{\mathcal{J}}_i\,,\ i=1,\ldots,N\,,\\
\delta\theta_i & = & \eta\displaystyle\sum_{\mu=1}^{p}\sum_{t=1}^{T}\Delta_i^\mu(t)+\alpha\,\delta\theta_{i(\mathrm{last})}\,, & i=1,\ldots,N\,.
\end{array}\right. \tag{4.24}$$

where

$$\Delta_j^\mu(t)=g_i'(h_j^\mu(t))\left[\sum_{i\in\mathcal{K}_j}\Delta_i^\mu(t)\,\omega_{ij}+\sum_{i\in\overline{\mathcal{K}}_j}\Delta_i^\mu(t+1)\,\overline{\omega}_{ij}+D_j^\mu(t)\,1_{\{j\in\mathcal{O}(t)\}}\right]\,, \tag{4.25}$$

the output errors through time are

$$D_j^\mu(t) = o_j(\boldsymbol{x}^\mu, t) - z_j^\mu(t)\,,\ j \in \mathcal{O}(t)\,, \tag{4.26}$$

and the contour condition

$$\Delta_j^\mu(T+1) = 0\,,\ j = 1, \ldots, N \tag{4.27}$$

is fulfilled. Once again, the online version is recovered when the sums over the whole training set are suppressed. The computation of the $\Delta_j^\mu(t)$ has to be done in the only possible order, i.e. proceeding from $t = T$ to $t = 1$, and for each step back-propagating in the network from $j = N$ to $j = 1$.

4.2 Analytical interpretation of the net output

Among the different types of neural networks, the ones which have found more applications are the multilayer feed-forward nets trained with the back-propagation method of Subsect. 4.1.1. This algorithm is based on the minimization of the squared error criterion of eq. (4.5). From the point of view of Statistics, supervised learning is, then, just a synonymous of *regression*, and it is well-known that the regression 'line' which minimizes the quadratic error is the function formed by the expectation values of the outputs conditioned to the inputs.

In this section we are going to make use of functional derivatives to find this *unconstrained global minimum*, which easily allows for the minimization of more involved error criterions [26, 29] (we exclude from this study the recurrent nets). Next, we will investigate the role played by the representation given to the training output patterns, specially whenever the number of different possible outputs is finite (e.g. in classification tasks).

The interest in this study lies in the fact that multilayer neural networks trained with back-propagation really find good approximations to the unconstrained global minimum of eq. (4.5). In fact, neural nets can approximate any sufficiently well-behaved function provided the number of units is large enough (see [4, 8, 32, 39] for several theorems on the approximation of functions with neural networks).

It must be stressed that the results will be derived with the only assumption that global minima are possible to be calculated, without any reference to the intrinsic difficulty of this problem nor to its dependence on the shape of the net; in fact, it need not be a neural network. That is, in this section the word 'net' should be understood as a short for 'big enough family of functions'.

4.2.1 Study of the quadratic error criterion

Let $(\xi, \zeta) \in \mathcal{X} \times \mathcal{Z}$ denote a certain pair of input-output patterns which has been produced with a given experimental setup. Since the sets $\mathcal{X}$ and $\mathcal{Z}$ are arbitrary, it is convenient to represent each pattern by a real vector in such a way that there is a one-to-one correspondence between vectors and feature patterns. We will make use of the vectors $\boldsymbol{x} \in \mathbb{R}^n$ for the input patterns and $\boldsymbol{z}(\boldsymbol{x}) \in \mathbb{R}^m$ for the output ones.

If $\{(\boldsymbol{x}^\mu, \boldsymbol{z}^\mu),\ \mu = 1, \ldots, p\}$ is a representative random sample of pairs input-output, our goal is to find the net

$$\boldsymbol{o} :\ \boldsymbol{x} \in \mathbb{R}^n \longmapsto \boldsymbol{o}(\boldsymbol{x}) \in \mathbb{R}^m \tag{4.28}$$

which closely resembles the unknown correspondence process. The least squares estimate is that which produces the lowest mean squared error $E[\boldsymbol{o}]$, where

$$E[\boldsymbol{o}] \equiv \frac{1}{2p} \sum_{\mu=1}^{p} \sum_{i=1}^{m} \lambda_i(\boldsymbol{z}^\mu, \boldsymbol{x}^\mu)\ (o_i(\boldsymbol{x}^\mu) - z_i^\mu)^2\ . \tag{4.29}$$

Usually the λ_i functions are set to 1. However, there are times in which it is useful to weight each contribution to the error with a term depending on the pattern. For instance, if the values of the desired outputs are known with uncertainties $\sigma_i(\boldsymbol{z}^\mu, \boldsymbol{x}^\mu)$, the right fitting or 'chi-squared' error should be

$$E[\boldsymbol{o}] \equiv \frac{1}{2p} \sum_{\mu=1}^{p} \sum_{i=1}^{m} \left(\frac{o_i(\boldsymbol{x}^\mu) - z_i^\mu}{\sigma_i(\boldsymbol{z}^\mu, \boldsymbol{x}^\mu)} \right)^2\ . \tag{4.30}$$

Under the three hypothesis that:

1. the different measurements ($\mu = 1, \ldots, p$) are independent (i.e., viewed as probability theory objects, they define independent random variables),
2. the underlying probability distribution of the differences $o_i(\boldsymbol{x}^\mu) - z_i^\mu$ has zero mean, $m_\mu = 0$, $\forall \mu$, and
3. the mean square deviations σ_μ are uniformily bounded, $\sigma_\mu < K$, for all $\mu = 1, \ldots, p$ (actually, in order to make use of Kolmogorov's theorem it is enough that $\displaystyle\sum_{\mu=1}^{p} \frac{\sigma_\mu}{\mu^2} < +\infty$, for any p),

the *Strong Law of Large Numbers* applies (see e.g. [21, 27]). It tells us that, with probability one (i.e., in the usual almost-everywhere convergence, common to the theory of functions and functional analysis) the limiting value of (4.29) for large p is given by

$$\begin{aligned} E[\boldsymbol{o}] &= \frac{1}{2} \sum_{i=1}^{m} \int_{\mathbb{R}^n} d\boldsymbol{x} \int_{\mathbb{R}^m} d\boldsymbol{z}\, p(\boldsymbol{z}, \boldsymbol{x})\, \lambda_i(\boldsymbol{z}, \boldsymbol{x})\, [o_i(\boldsymbol{x}) - z_i]^2 \\ &= \frac{1}{2} \sum_{i=1}^{m} \int_{\mathbb{R}^n} d\boldsymbol{x}\, p(\boldsymbol{x}) \int_{\mathbb{R}^m} d\boldsymbol{z}\, p(\boldsymbol{z}|\boldsymbol{x})\, \lambda_i(\boldsymbol{z}, \boldsymbol{x})\, [o_i(\boldsymbol{x}) - z_i]^2\ , \end{aligned} \tag{4.31}$$

where $p(\boldsymbol{z}, \boldsymbol{x})$ is the joint probability density of the random variables $\boldsymbol{z}$ and $\boldsymbol{x}$ in the sample, $p(\boldsymbol{x})$ stands for the probability density of $\boldsymbol{x}$, and $p(\boldsymbol{z}|\boldsymbol{x})$ is the conditional probability density of $\boldsymbol{z}$ knowing that the former random variable has taken on the value $\boldsymbol{x}$.

Notice that the first two of the conditions for the validity of the strong law of large numbers are naturally satisfied in most cases. In fact, while the first one is equivalent to the usual rule that the practical measurements must always be done properly (which is generally assumed), the second just tells us that the net is also to be constructed conveniently in order to fulfil the goal of closely resembling the unknown correspondence process (see above). But we also take for granted that we will be always able to do this, in the end. The third condition, however, is of a rather more technical nature and seems to be difficult to realize from the very begining (or even at the end, in a strict sense!). In practice, the thing to do is obviously to check *a posteriori* that it is fulfilled for p large enough, and to convince ourselves that there is no reason (in the case treated) for it to be violated at any value of p. We do think that this condition prevents one from being able to consider the use of the strong law of large numbers as something that can be 'taken for granted' in general, in the situation described in this section. This comment should be considered as a warning against the apparently indiscriminate application of the law which can be found sometimes in the related literature.

Assuming no constraint in the functional form of $\boldsymbol{o}(\boldsymbol{x})$, the minimum $\boldsymbol{o}^*(\boldsymbol{x})$ of E is easily found by annulling the first functional derivative:

$$\begin{aligned}\frac{\delta E[\boldsymbol{o}]}{\delta o_j(\boldsymbol{x})} &= \sum_{i=1}^{m} \int_{\mathbb{R}^n} d\boldsymbol{x}'\, p(\boldsymbol{x}') \int_{\mathbb{R}^m} d\boldsymbol{z}\, p(\boldsymbol{z}|\boldsymbol{x}')\, \lambda_i(\boldsymbol{z}, \boldsymbol{x}')\, [o_i(\boldsymbol{x}') - z_i]\, \delta_{ij}\, \delta(\boldsymbol{x} - \boldsymbol{x}') \\ &= p(\boldsymbol{x}) \int_{\mathbb{R}^m} d\boldsymbol{z}\, p(\boldsymbol{z}|\boldsymbol{x})\, \lambda_j(\boldsymbol{z}, \boldsymbol{x})[o_j(\boldsymbol{x}) - z_j] \\ &= p(\boldsymbol{x})\, [o_j(\boldsymbol{x}) \langle \lambda_j(\boldsymbol{z}, \boldsymbol{x}) \rangle_{\boldsymbol{x}} - \langle \lambda_j(\boldsymbol{z}, \boldsymbol{x})\, z_j \rangle_{\boldsymbol{x}}] = 0 \qquad (4.32)\end{aligned}$$

implies that the searched minimum is

$$o_j^*(\boldsymbol{x}) = \frac{\langle \lambda_j(\boldsymbol{z}, \boldsymbol{x})\, z_j \rangle_{\boldsymbol{x}}}{\langle \lambda_j(\boldsymbol{z}, \boldsymbol{x}) \rangle_{\boldsymbol{x}}}\,,\ \forall \boldsymbol{x} \in \mathbb{R}^n \text{ such that } p(\boldsymbol{x}) \neq 0\,,\ j = 1, \ldots, m\,, \qquad (4.33)$$

where we have made use of the *conditional expectation values*

$$\langle f(\boldsymbol{z}, \boldsymbol{x}) \rangle_{\boldsymbol{x}} \equiv \int_{\mathbb{R}^m} d\boldsymbol{z}\, p(\boldsymbol{z}|\boldsymbol{x})\, f(\boldsymbol{z}, \boldsymbol{x}) \qquad (4.34)$$

i.e. the average of any function f of the output vectors $\boldsymbol{z}$ once the input pattern $\boldsymbol{x}$ has been fixed. Eq. (4.33) is the key expression from which we will derive the possible interpretations of the net output (an alternative proof can be found for instance in [58]).

From a practical point of view unconstrained nets do not exist, which means that the achievable minimum $\tilde{\boldsymbol{o}}(\boldsymbol{x})$ is in general different to the desired $\boldsymbol{o}^*(\boldsymbol{x})$. The mean squared error between them is written as

$$\varepsilon[\tilde{\boldsymbol{o}}] \equiv \frac{1}{2} \sum_{i=1}^{m} \int_{\mathbb{R}^n} d\boldsymbol{x}\, p(\boldsymbol{x}) \int_{\mathbb{R}^m} d\boldsymbol{z}\, p(\boldsymbol{z}|\boldsymbol{x})\, \lambda_i(\boldsymbol{z},\boldsymbol{x})\, [\tilde{o}_i(\boldsymbol{x}) - o_i^*(\boldsymbol{x})]^2 \,. \tag{4.35}$$

However, it is straightforward to show that

$$E[\boldsymbol{o}] = \varepsilon[\boldsymbol{o}] + \frac{1}{2} \sum_{i=1}^{m} \int_{\mathbb{R}^n} d\boldsymbol{x}\, p(\boldsymbol{x}) \int_{\mathbb{R}^m} d\boldsymbol{z}\, p(\boldsymbol{z}|\boldsymbol{x})\, \lambda_i(\boldsymbol{z},\boldsymbol{x})\, [z_i - o_i^*(\boldsymbol{x})]^2 \,, \tag{4.36}$$

and, since the second term of the sum is a constant (it does not depend on the net), the minimizations of both $E[\boldsymbol{o}]$ and $\varepsilon[\boldsymbol{o}]$ are equivalent. Therefore, $\tilde{\boldsymbol{o}}(\boldsymbol{x})$ is a minimum squared-error approximation to the unconstrained minimum $\boldsymbol{o}^*(\boldsymbol{x})$.

In the rest of this subsection we will limit our study to problems for which it is satisfied that $\lambda_i(\boldsymbol{z},\boldsymbol{x}) = 1\,,\ \forall i\,,\ \forall \boldsymbol{z}\,,\ \forall \boldsymbol{x}$. In fact, they cover practically all the applications of back-propagation, since the training patterns are most of the times implicitly regarded as points without error bars. Then, eq. (4.33) gains the simpler form

$$o_j^*(\boldsymbol{x}) = \langle z_j \rangle_{\boldsymbol{x}} = \int_{\mathbb{R}^m} d\boldsymbol{z}\, p(\boldsymbol{z}|\boldsymbol{x})\, z_j \,,\ j = 1, \ldots, m \,, \tag{4.37}$$

whose meaning is that the unconstrained minimum of

$$E[\boldsymbol{o}] = \frac{1}{2p} \sum_{\mu=1}^{p} \sum_{i=1}^{m} \left(o_i(\boldsymbol{x}^\mu) - z_i^\mu\right)^2 \,, \tag{4.38}$$

is, for large p, the *conditional expectation value* or *mean* of the output vectors in the training sample for each input pattern represented by $\boldsymbol{x} \in \mathbb{R}^n$.

As a particular case, if the output representation is chosen to be discrete, say

$$\boldsymbol{z}(\boldsymbol{x}) \in \{\boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}, \ldots, \boldsymbol{z}^{(a)}, \ldots\} \,, \tag{4.39}$$

then eq. (4.37) reads

$$o_j^*(\boldsymbol{x}) = \sum_a P(\boldsymbol{z}^{(a)}|\boldsymbol{x})\, z_j^{(a)} \,,\ j = 1, \ldots, m \tag{4.40}$$

where $P(\boldsymbol{z}^{(a)}|\boldsymbol{x})$ is the probability of $\boldsymbol{z}^{(a)}$ conditioned to the knowledge of the value of the input vector $\boldsymbol{x}$.

4.2.2 Unary output representations and Bayesian decision rule

It is well known that nets trained to minimize (4.38) are good approximations to Bayesian classifiers, provided a *unary representation* is taken for the output patterns [25, 64, 78]. That is, suppose the input patterns have to be separated in C different classes $\mathcal{X}_a$, $a = 1, \dots, C$, and let

$$\boldsymbol{z}^{(a)} \equiv (\overset{1}{0}, \dots, \overset{a-1}{0}, \overset{a}{1}, \overset{a+1}{0}, \dots, \overset{m}{0}) \tag{4.41}$$

be the desired output of any input pattern $\boldsymbol{x} \in \mathcal{X}_a$. This assignment specializes each output component to recognize a distinct class ($m = C$). Substituting (4.41) in eq. (4.40) we get

$$o^*_a(\boldsymbol{x}) = \sum_b P(\boldsymbol{z}^{(b)}|\boldsymbol{x}) z^{(b)}_a = P(\boldsymbol{z}^{(a)}|\boldsymbol{x}) , \tag{4.42}$$

i.e. the a-th component of the net output turns out to be a minimum squared approximation to the conditional probability that pattern $\boldsymbol{x}$ belong to class $\mathcal{X}_a$. Therefore, if $\tilde{\boldsymbol{o}}(\boldsymbol{x})$ is the output of the net once the learning phase has finished, a good proposal for an almost *Bayesian decision rule* would be:

> $\boldsymbol{x}$ is most likely a member of class $\mathcal{X}_b$, where $\tilde{o}_b(\boldsymbol{x})$ is the largest component of the output vector $\tilde{\boldsymbol{o}}(\boldsymbol{x})$.

The applicability of eq. (4.42) goes beyond classifications. For example, suppose that you have a certain Markov chain $\{\boldsymbol{s}_t\, ,\ t \in \mathbb{N}\}$ of discrete states with constant transition probabilities, and you train a net to learn $\boldsymbol{s}_t$ as a function of $\boldsymbol{s}_{t-1}, \dots, \boldsymbol{s}_{t-\tau}$. Hence, the output of the net will tend to give these transition probabilities $P(\boldsymbol{s}_t|\boldsymbol{s}_{t-1}, \dots, \boldsymbol{s}_{t-\tau})$, which by hypothesis do not depend on t.

4.2.3 Other discrete output representations

In the previous subsection we showed how nets can solve, among others, classification problems through the use of unary output representations. The role played by these representations is fundamental, not because they easily give the right solution but because the output contains all the information needed to make a Bayesian decision. In fact, it is easy to find other representations for which some of the information will be unavoidably losen. For instance, consider a *binary representation* for a four classes classification task:

$$\left\{ \begin{array}{rcl} \boldsymbol{z}^{(1)} & \equiv & (0,0) \\ \boldsymbol{z}^{(2)} & \equiv & (1,0) \\ \boldsymbol{z}^{(3)} & \equiv & (0,1) \\ \boldsymbol{z}^{(4)} & \equiv & (1,1) \end{array} \right. \tag{4.43}$$

Then, eq. (4.40) leads to

$$\left\{\begin{array}{lcl} o_1^*(\boldsymbol{x}) & = & P(\boldsymbol{z}^{(2)}|\boldsymbol{x}) + P(\boldsymbol{z}^{(4)}|\boldsymbol{x}) \\ o_2^*(\boldsymbol{x}) & = & P(\boldsymbol{z}^{(3)}|\boldsymbol{x}) + P(\boldsymbol{z}^{(4)}|\boldsymbol{x}) \end{array}\right. \tag{4.44}$$

with the normalization condition

$$\sum_{a=1}^{4} P(\boldsymbol{z}^{(a)}|\boldsymbol{x}) = 1 \,. \tag{4.45}$$

Eqs. (4.44) and (4.45) constitute an indeterminated linear system of three equations with four unknown conditional probabilities. The situation will be the same whenever a binary output representation is taken. Thus, they should be avoided if useful solutions are required.

Of course, unary representations are not the only possible choice to find useful solutions. For example, a 'thermometer' representation [23] for the same problem could be

$$\left\{\begin{array}{lcl} \boldsymbol{z}^{(1)} & \equiv & (0,0,0) \\ \boldsymbol{z}^{(2)} & \equiv & (1,0,0) \\ \boldsymbol{z}^{(3)} & \equiv & (1,1,0) \\ \boldsymbol{z}^{(4)} & \equiv & (1,1,1) \end{array}\right. \tag{4.46}$$

which has as solution

$$\left\{\begin{array}{lcl} P(\boldsymbol{z}^{(1)}|\boldsymbol{x}) & = & 1 - o_1^*(\boldsymbol{x}) \\ P(\boldsymbol{z}^{(2)}|\boldsymbol{x}) & = & o_1^*(\boldsymbol{x}) - o_2^*(\boldsymbol{x}) \\ P(\boldsymbol{z}^{(3)}|\boldsymbol{x}) & = & o_2^*(\boldsymbol{x}) - o_3^*(\boldsymbol{x}) \\ P(\boldsymbol{z}^{(4)}|\boldsymbol{x}) & = & o_3^*(\boldsymbol{x}) \end{array}\right. \tag{4.47}$$

The interest towards these representations comes from the need of discretizing continuous output spaces. Simulations have shown that binary representations are more difficult to be learnt than thermometer-like ones. However, it is not so clear that those who selected them have interpreted their results in the proper way, putting the outputs in terms of conditional probabilities, and deciding as true output the class of maximal probability.

The final conclusion which could be extracted from what has been said is that, in the discrete and finite case, it is always possible to make an approximated Bayesian decision provided the representation $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(C)}\}$ is chosen such that the linear system

$$\left\{\begin{array}{l} \displaystyle\sum_{b=1}^{C} P(\boldsymbol{z}^{(b)}|\boldsymbol{x})\, z_a^{(b)} = o_a^*(\boldsymbol{x}) \,,\ a = 1, \ldots, d \,,\ d \in \{C-1, C\} \\ \displaystyle\sum_{b=1}^{C} P(\boldsymbol{z}^{(b)}|\boldsymbol{x}) = 1 \text{ needed if } d = C - 1 \end{array}\right. \tag{4.48}$$

has a non null determinant.

Class	m	σ
1	0.0	0.997
2	0.8	0.878
3	4.0	2.732
4	-0.8	1.333

Table 4.1: Averages and standard deviations of the normal probability densities for the four gaussians problem.

4.2.4 An example of learning with different discrete output representations

In order to compare what happens when different discrete output representations are considered we have designed the following example, which from now on we will refer to as the 'four gaussians problem'. Suppose we have one-dimensional real patterns belonging to one of four possible different classes. All the classes are equally probable, $P(\text{class } a) = 1/4,\ a = 1, \ldots, 4$, and their respective distributions $p(\boldsymbol{x}|\text{class } a)$ are normal $N(m, \sigma)$ with averages m and standard deviations σ given in Table 4.1 (see Fig. 4.4). Knowing them, the needed conditional probabilities are given by the Bayes theorem:

$$P(\text{class } a|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\text{class } a)}{\displaystyle\sum_{b=1}^{4} p(\boldsymbol{x}|\text{class } b)}\ ,\ a = 1, \ldots, 4. \tag{4.49}$$

We have trained three neural networks to classify these patterns using as many different output representations: unary, binary and real, as defined in Table 4.2. All the networks had one input unit, two hidden layers with six and eight units respectively, and four output units in the unary case, two in the binary case and one in the real case. The activations functions were sigmoids, and back-propagation was not batched, i.e. the weights were changed after the presentation of each pattern.

In Fig. 4.5 we have plotted the expected Bayesian classification (solid line), which according to eq. (4.42) should coincide with the minimum using unary output representation, together with a solution given by the back-propagation algorithm using that representation (dashed line). Both lines are almost the

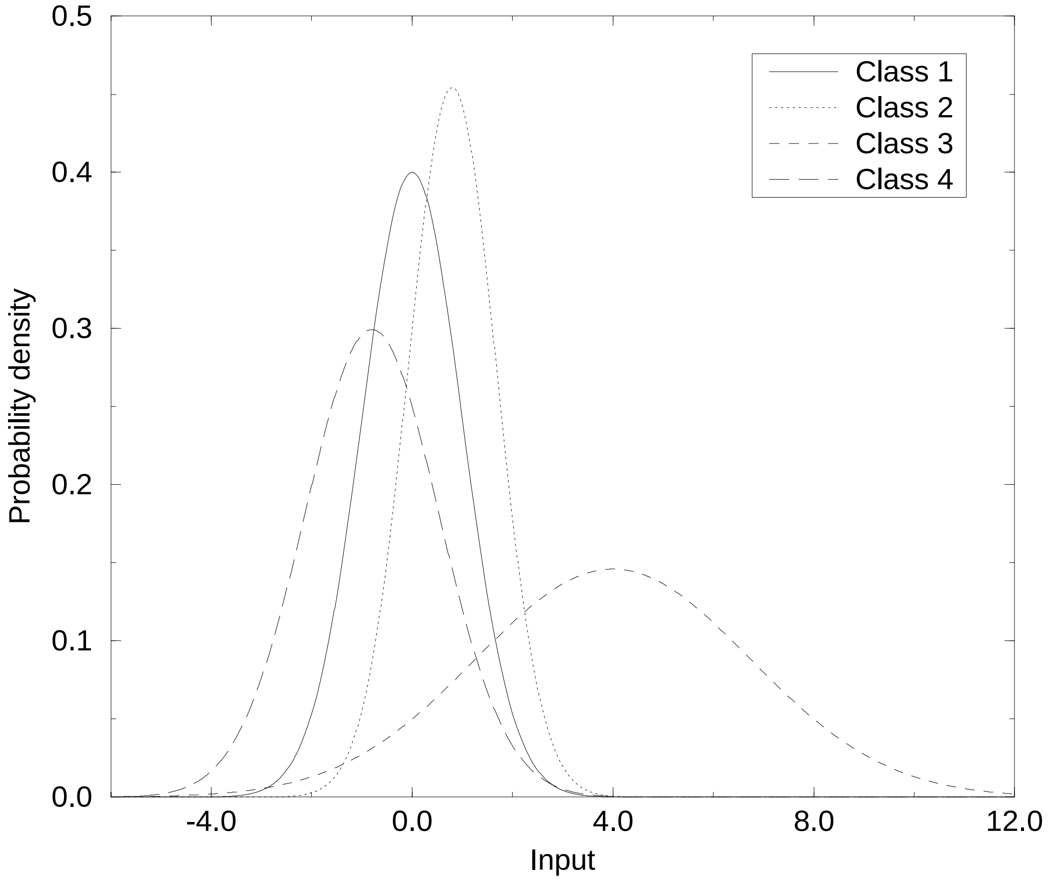

Figure 4.4: Probability densities for the four gaussians problem.

Class	Unary	Binary	Real
1	(1,0,0,0)	(0,0)	(1/8)
2	(0,1,0,0)	(0,1)	(3/8)
3	(0,0,1,0)	(1,0)	(5/8)
4	(0,0,0,1)	(1,1)	(7/8)

Table 4.2: Three representations for the four gaussians problem.

same, but the net assigns the forth class to patterns lower than -5.87 when it should be the third one. This discrepancy is easily understood if one realizes that the probability of having patterns below -5.87 is about 4.92×10^{-8}, which means that the number of patterns generated with such values is absolutely negligible. Thus, the net cannot learn patterns which it has not seen! This insignificant mistake appears several times in this subsection, but will not be commented any more. The conclusion is, then, that prediction and simulation agree fairly well, and since the theoretical output is the Bayes classifier, neural nets achieve good approximations to the best solution provided the different classes are encoded using a unary output representation.

To show that neural nets really approach eq. (4.42) we have added Fig. 4.6, which is a plot of both the predicted conditional probability and the learnt output of the first of the four output units. It must be taken into account that to make approximations to Bayesian decisions you just have to look at the unit which gives the largest output, but you do not need their actual values. However, in the Markov chain example previously proposed, those values would be certainly necessary.

When using binary and real representations, one has to decide which outputs should go to each class. For instance, the most evident solution for the binary case is to apply a cut at 0.5 to both output units, assigning 0 if the output is below the cut and 1 otherwise. For the real representation a logical procedure would be the division of the interval $[0, 1]$ in four parts of equal length, say $[0, 0.25]$, $[0.25, 0.50]$, $[0.50, 0.75]$ and $[0.75, 1]$, and then assign 1/8, 3/8, 5/8 and 7/8 respectively. These interpretations have been exploited in Fig. 4.7, where we have plotted the expected results for the four gaussians problem in the three cases of Table 4.2, according to eq. (4.40). The three lines coincide only in the input regions in which the probability of one class is much bigger than that of the rest. Both binary and real cases fail to distinguish the first class in the interval

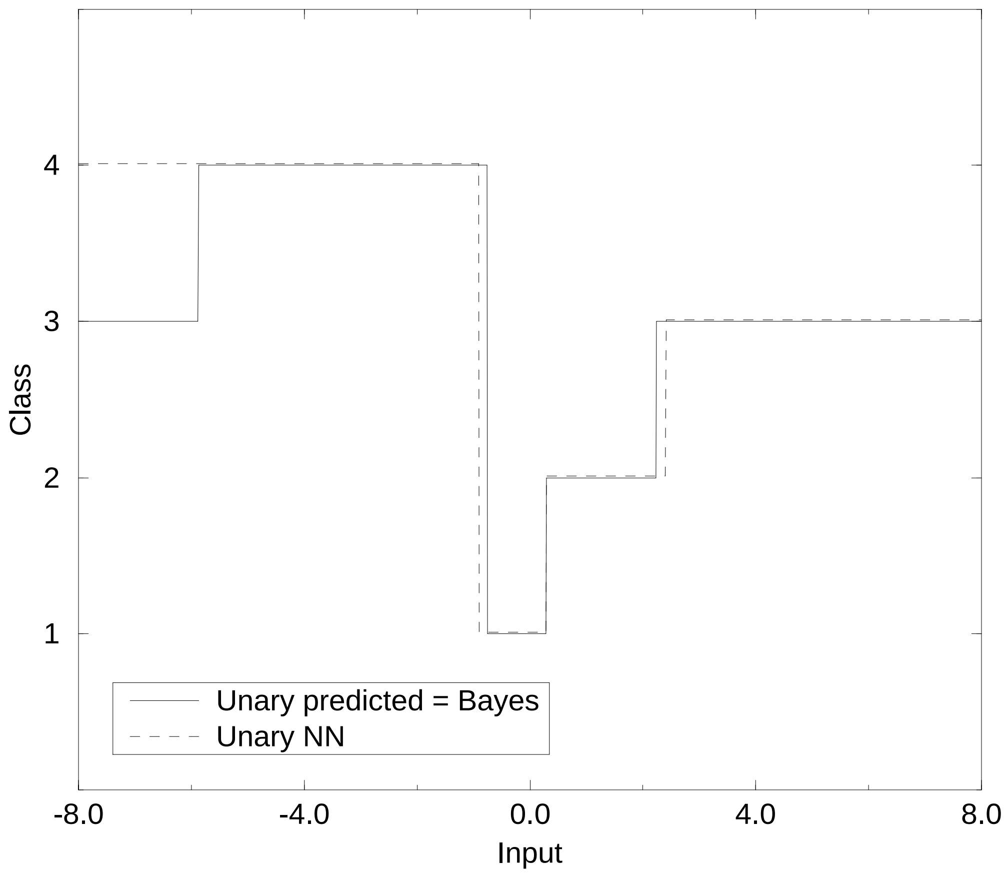

Figure 4.5: Predicted and neural network classifications for the four gaussians problem using unary output representations.

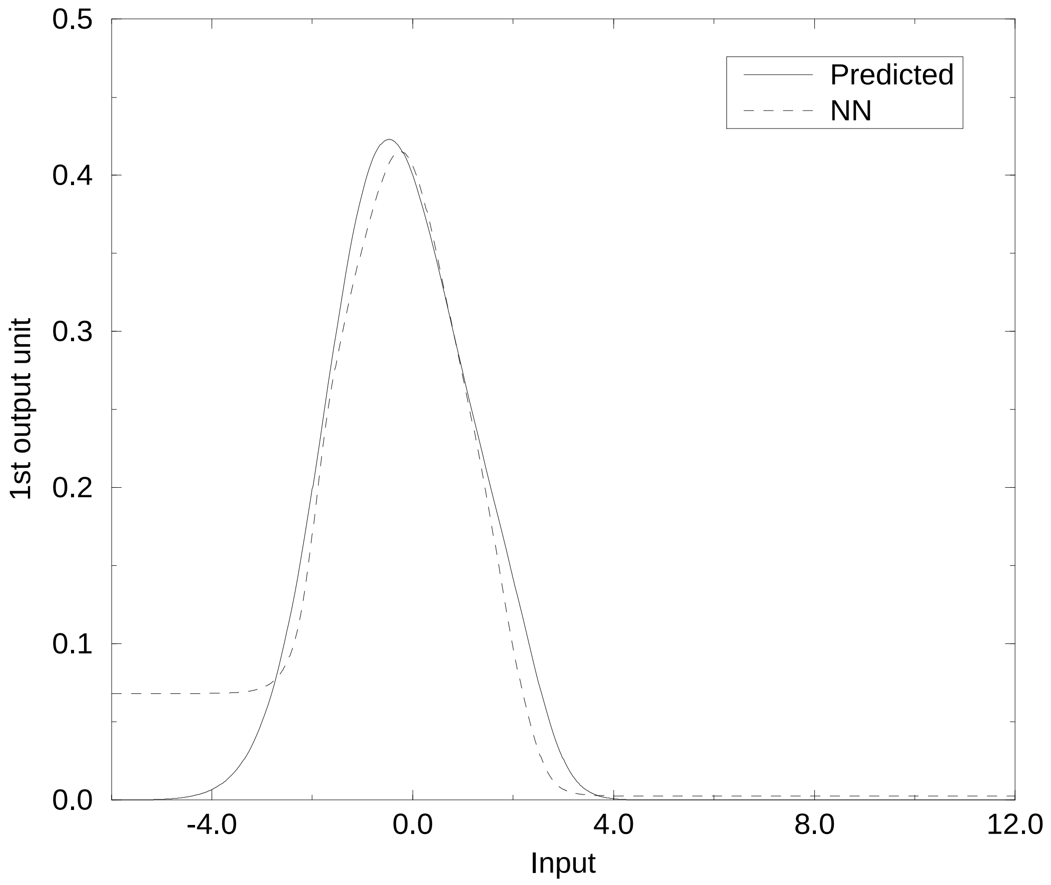

Figure 4.6: First output unit state for the four gaussians problem using the unary output representation.

$[-0.76, 0.29]$. Moreover, the real case incorrectly assigns the third class in the interval $[-2.27, -0.84]$ instead of the forth. The narrow peak at $[2.20, 2.27]$ of the binary representation is just an effect of the transition between the third and the second classes, which are represented as $(1, 0)$ and $(0, 1)$ respectively, making very difficult that both output units cross the cut simultaneously in opposite directions.

Figs. 4.8 and 4.9 show predicted and neural networks classifications when binary and real representations are employed. Two examples for the binary neural network outputs are shown, with the expected peaks around 2.23, either as a transition through $(1, 1)$, as in NN1, or through $(0, 0)$, as in NN2.

4.2.5 Continuous output representations

Discrete and finite output representations arise quite naturally in the treatment of classification problems. For each class, an arbitrary but different vector is assigned and, taking into account the system (4.48), the best way of doing it so is by imposing linear independence of this set of vectors. Now, with the aid of a sample of patterns, we will be able to determine good approximations to the conditional probabilities that the patterns belong to each class and, knowing them, Bayesian decisions will be immediate.

On the other hand, prediction and interpolation tasks usually amount to finding the 'best' value of several continuous variables for each given input. One possible but unsatisfactory solution is the discretization of these variables, which has to be made carefully in order to skip various problems. If the number of bins is to big, the number of training patterns should be very large. Otherwise, if the size of the bins is relatively big, the partitioning may fail to distinguish relevant differences, specially if the output is not uniformly distributed. Therefore, it may be stated that a good discretization needs a fair understanding of the unknown output distribution!

Fortunately, neural nets have proved to work well even when the output representation is left to be continuous, without any discretization. For instance, feed-forward networks have been applied to time series prediction of continuous variables, outperforming standard methods [79]. The explanation to this success lies precisely in eq. (4.37), which reveals the tendency of nets to learn, for each input, the *mean* of its corresponding outputs in the training set. Thus, the net is automatically doing what everyone would do in the absence of more information, i.e. substituting the sets of points with common abcise by their average value.

To illustrate that learning with neural networks really tends to give the minimum squared error solution given by eq. (4.37) we have trained them with a set of patterns distributed as the dots in Fig. 4.10. The solid line is the theoretical limit, while the dashed lines are two different solutions found by neural nets. The first one has been produced using the ordinary sigmoidal activation function, while in the second they have been replaced by sinusoidal ones (scaled between 0 and

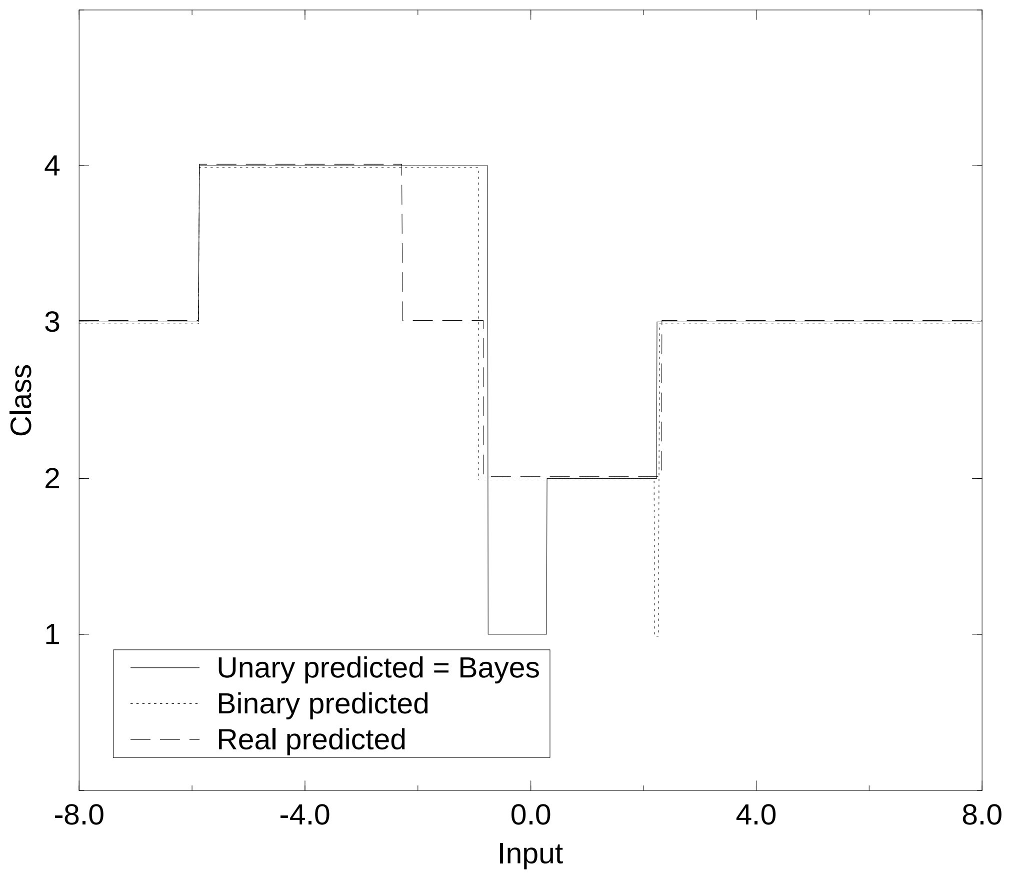

Figure 4.7: Predicted classifications for the four gaussians problem. Only the unary output representation achieves the desired Bayesian classification, whereas both binary and real representations give the wrong answer for certain values of the inputs.

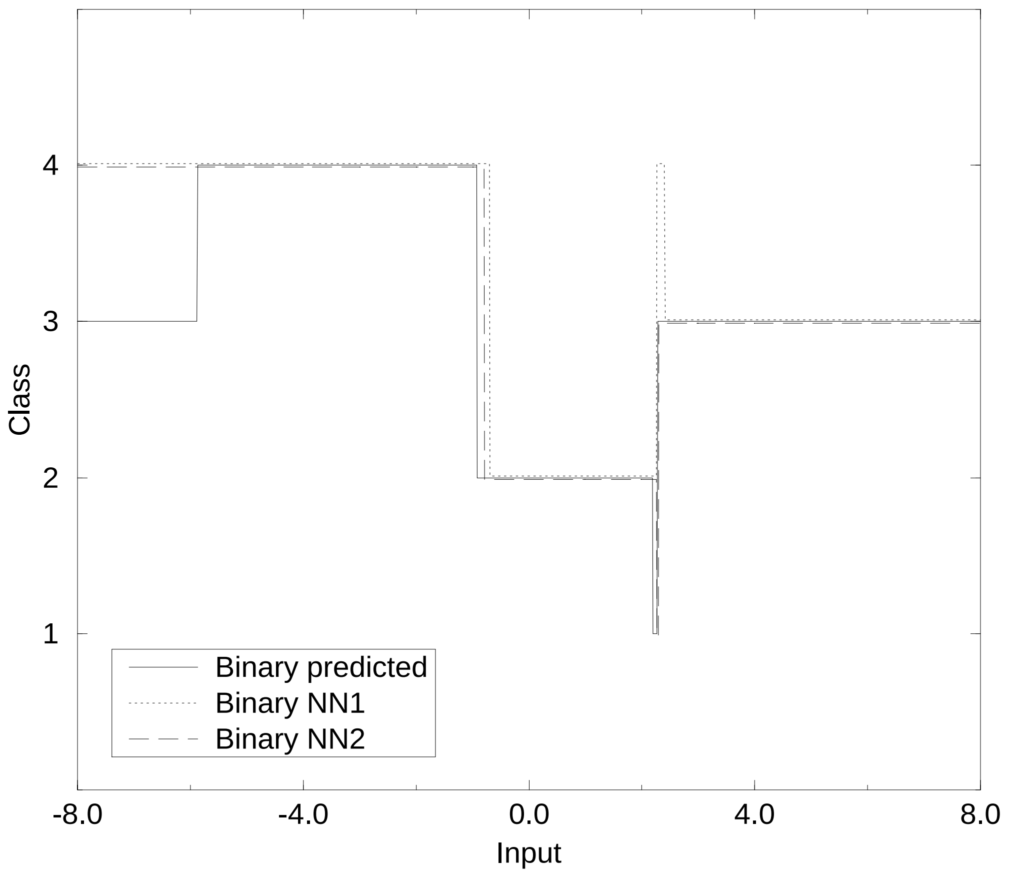

Figure 4.8: Predicted and neural network classifications for the four gaussians problem using binary output representations.

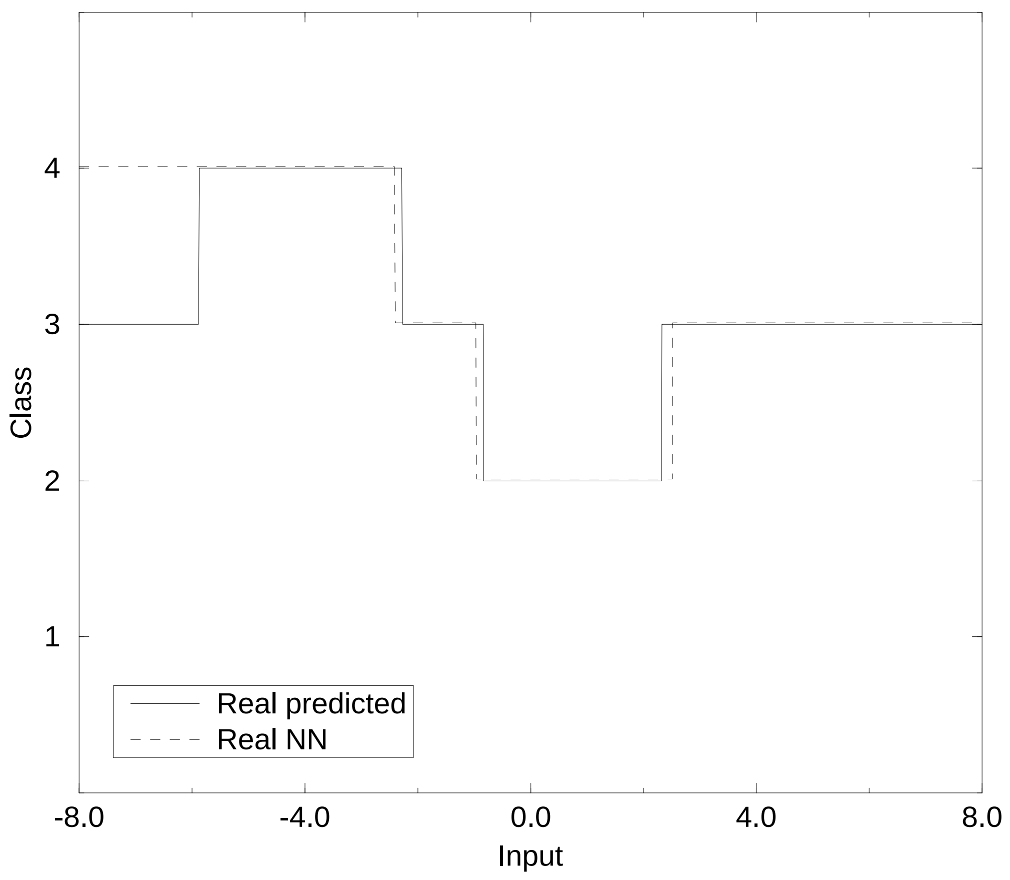

Figure 4.9: Predicted and neural network classifications for the four gaussians problem using real output representations.

1). In most of the input interval the three curves are very similar. However, the sigmoidal one fails to produce the peak located at about -0.5. This is in favour of recent results [76] which show that sinusoidal activations can solve difficult task which sigmoidal cannot, or can but with much more epochs of training.

4.2.6 Study of other error criterions

In all the previous subsections the study has been concentrated in the minimization of the quadratic error function eq. (4.38). However, other quantities may serve for the same purpose, such as

$$E_q[\boldsymbol{o}] \equiv \frac{1}{q\,p} \sum_{\mu=1}^{p} \sum_{i=1}^{m} |o_i(\boldsymbol{x}^\mu) - z_i(\boldsymbol{x}^\mu)|^q \ . \tag{4.50}$$

For instance, in [42] Karayiannis and Venetsanopoulos modify the error measure during the learning in order to accelerate its convergence, changing in a continuous way from $E_2[\boldsymbol{o}]$ to $E_1[\boldsymbol{o}]$.

Repeating the scheme of Subsect. 4.2.1, the large p limit of eq. (4.50) is

$$E_q[\boldsymbol{o}] = \frac{1}{q} \sum_{i=1}^{m} \int_{\mathbb{R}^n} d\boldsymbol{x}\, p(\boldsymbol{x}) \int_{\mathbb{R}^m} d\boldsymbol{z}\, p(\boldsymbol{z}|\boldsymbol{x})\ |o_i(\boldsymbol{x}) - z_i|^q \ , \tag{4.51}$$

and its unconstrained minimum $\boldsymbol{o}^*(\boldsymbol{x};q)$ is found by annulling the first functional derivative. The solutions for the different values of q satisfy the following equations:

$$\begin{aligned}
&\sum_{k=0}^{q-1} (-1)^k \binom{q-1}{k} o_j^*(\boldsymbol{x};q)^{q-k-1} \left\langle (z_j)^k \right\rangle_{\boldsymbol{x}} = 0 && \text{if } q \text{ even,}\\
&\sum_{k=0}^{q-1} (-1)^k \binom{q-1}{k} o_j^*(\boldsymbol{x};q)^{q-k-1} \left\langle \mathrm{sign}(o_j^*(\boldsymbol{x};q) - z_j)\,(z_j)^k \right\rangle_{\boldsymbol{x}} = 0 && \text{if } q \text{ odd.}
\end{aligned} \tag{4.52}$$

The most interesting case is when $q = 1$ due to the fact that eq. (4.52) acquires the simplest form

$$\left\langle \mathrm{sign}(o_j^*(\boldsymbol{x};1) - z_j) \right\rangle_{\boldsymbol{x}} = 0\,, \tag{4.53}$$

which may be written as

$$\int_{-\infty}^{o_j^*(\boldsymbol{x};1)} dz_j\, p_{(j)}(z_j|\boldsymbol{x}) = \int_{o_j^*(\boldsymbol{x};1)}^{\infty} dz_j\, p_{(j)}(z_j|\boldsymbol{x})\ ,\ j = 1, \ldots, m, \tag{4.54}$$

where $p_{(j)}(z_j, \boldsymbol{x})$ is the j-th marginal distribution of $p(\boldsymbol{z}, \boldsymbol{x})$. Therefore, the minimization of $E_1[\boldsymbol{o}]$ has as solution the function that assigns, for each input $\boldsymbol{x}$, the *median* of its corresponding outputs in the training set.

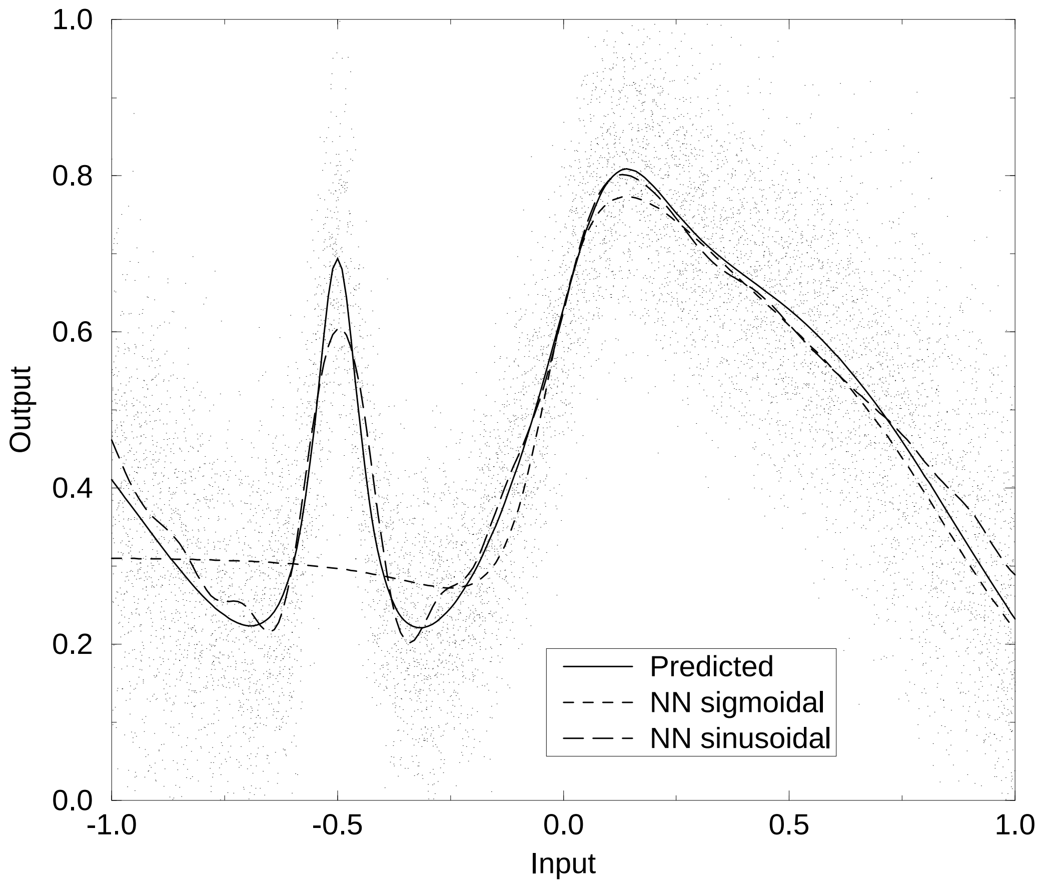

Figure 4.10: Predicted and neural network outputs. Two different nets have been tested, one with ordinary sigmoidal activation functions and the other with sinusoidal ones.

4.3 Applications

4.3.1 Reconstruction of images from noisy data

A typical problem which appears when one wants to make use of images for scientific purposes is the existence of different kinds of noise which reduce their quality. For instance, astronomical photographs from the Earth are usually contaminated by the atmosphere, while the Hubble space telescope has suffered from a disappointing spherical aberration which has severely damaged the results obtained during a large period of time. The same happens with all sorts of images, since 'perfect' cameras do not exist.

Taking for granted the presence of noise, several techniques have been proposed for the *reconstruction* of the original images from the noisy ones. In particular, *Bayesian*, *maximum entropy* and *maximum likelihood estimators* are among the most successful methods (see e.g. [56] for a particular iterative algorithm called FMAPE). Most of them share the characteristic of being very time consuming, by involving several Fast Fourier Transforms over the whole image at each iteration. In this subsection we are going to show how neural networks can be easily applied for image reconstruction, giving rise to a method which, once the learning has been done, it is almost instantaneous.

Let us consider the 500×500 aerial image of Fig. 4.11, having 256 grey levels. The noise has made it look blurred, avoiding the possibility of distinguishing the details. Fig. 4.12 shows the same image but reconstructed with the aid of the FMAPE algorithm. Now the whole image looks sharper, and some details previously hidden become visible. Using the standard online back-propagation method we have trained a multilayer neural network, with architecture 49:10:3:1, in the following way: the input patterns were subarrays (or cells) of 7×7 pixels of the noisy image of Fig. 4.11, chosen randomly, whose corresponding desired outputs were the central pixel of each subarray, but read in the reconstructed image of Fig. 4.12. Sigmoidal activation functions were used, and the pixels were linearly scaled between 0 and 1. Thus, it may be stated that we have 'trained the net to eliminate the noise'.

Fig. 4.13 shows the result of applying the learnt network to the noisy image of Fig. 4.11. It looks like very much to Fig. 4.12, which has been used as the prototype of noiseless image. However, the key of this method consists in the realization that we can apply this network to any other image, without having to perform a new learning process. For instance, the reconstruction of the image of Fig. 4.14 with the previous net is given in Fig. 4.16, which compares favourably with the FMAPE solution shown in Fig. 4.15. Therefore, a single standard image reconstruction plus a single back-propagation learning generate a multilayer neural network capable of reconstructing a large number of images. Since the learnt network has been trained to eliminate the noise of a particular image, good results are expected if the new images presented to the net have similar characteristics to it and, in particular, if the structure of the noise is not too different.

Several improvements could be introduced to our method. For instance, if we have a family of images, all taken with the same camera and in similar circumstances, the training could be done not with a single and whole image but with a selection of the most significative parts of several images. Another possiblity is the pre-selection of the training patterns within a single image so as to concentrate the learning on the structures we are more interested in. This happens, for instance, when the image to be reconstructed is too big that the standard methods cannot process the whole image, so they are applied only to smaller subimages. Finally, in the case of color images the procedure would be the same, but reconstructing separately each of the three color channels.

4.3.2 Image compression

The increasing amount of information which has to be stored by informatic means (e.g. in data bases) and the need of faster data transmissions has risen the interest towards data compression. Among the different existing methods it is possible to distinguish two big classes: those which are completely reversible, and those which may result in a certain information loss. Usually, the modification or loss of a single byte is unacceptable. However, there are situations in which a certain loss of 'quality' of the item to be stored is greatly compensated by the achievement of a large enough compression rate. This is the case, for instance, of digitalized images, since each one spends a lot of disk space (a typical size is 1Mb), and the reversible compression methods cannot decrease their size in, roughly speaking, more than a half.

Self-supervised back-propagation

Suppose that we want to compress a n-dimensional pattern distribution. We start by choosing a multilayer neural network whose input and output layers have n units, and with a hidden layer of m units, with $m < n$, thus known as a *bottle-neck* layer. Using a *self-supervised back-propagation* [70], which consists in a standard back-propagation with a training set formed by pairs for which the desired output members are chosen to be equal to the corresponding input patterns, two functions f_1 and f_2 are found:

$$\text{Input} \in \mathbb{R}^n \xrightarrow{f_1} \text{Compressed} \in \mathbb{R}^m \xrightarrow{f_2} \text{Output} \in \mathbb{R}^n \, .$$

The first function, f_1, transforms the n-dimensional input data into *compressed patterns*, the neck units activation, with a lower dimension ($m < n$). Thus, f_1 may be considered a projector of the input space into a smaller intermediate subspace. The second function, f_2, transforms the compressed data into the output data, which has the same dimension as the input (n). Therefore, f_2 may

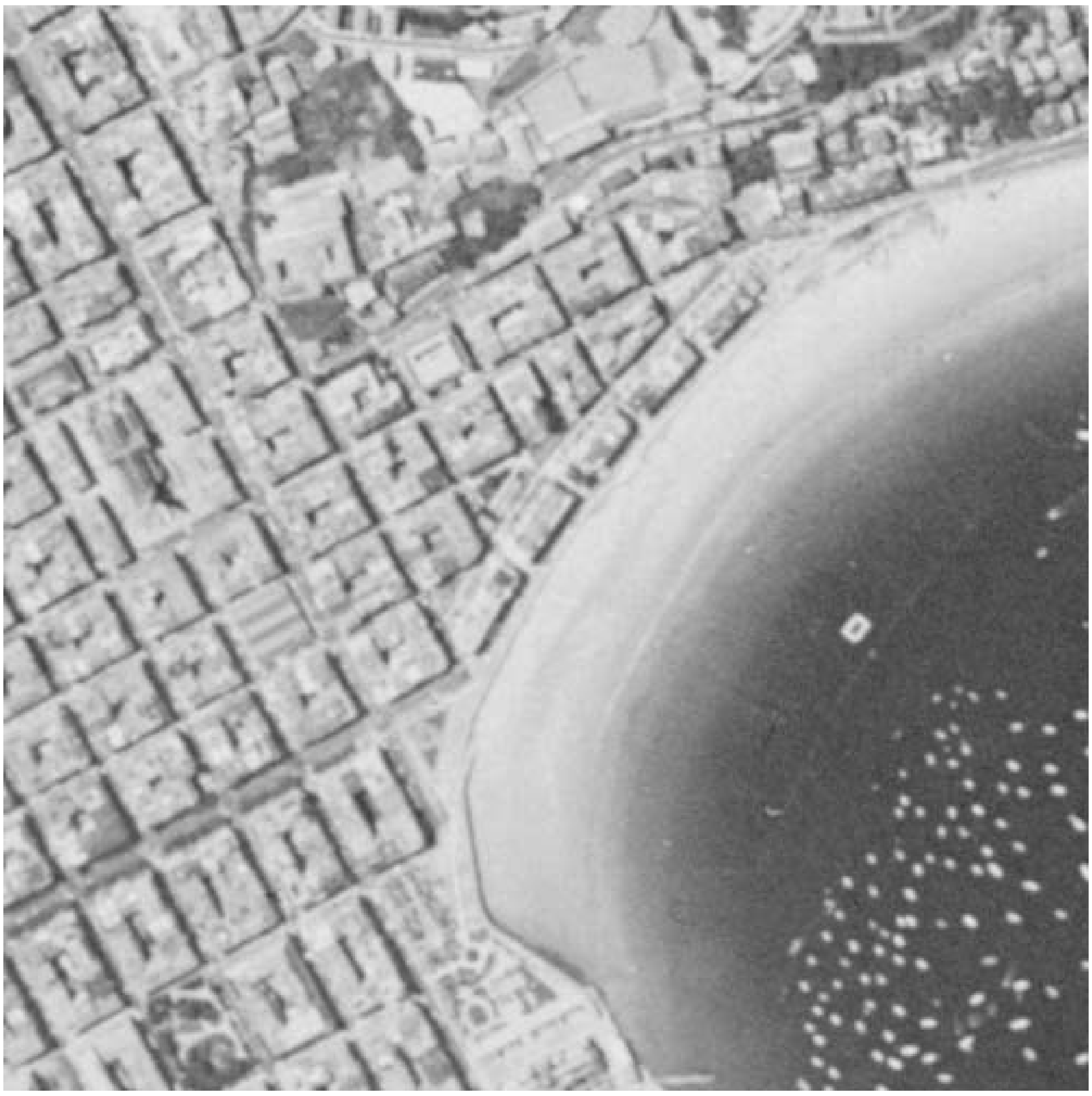

Figure 4.11: A noisy image.

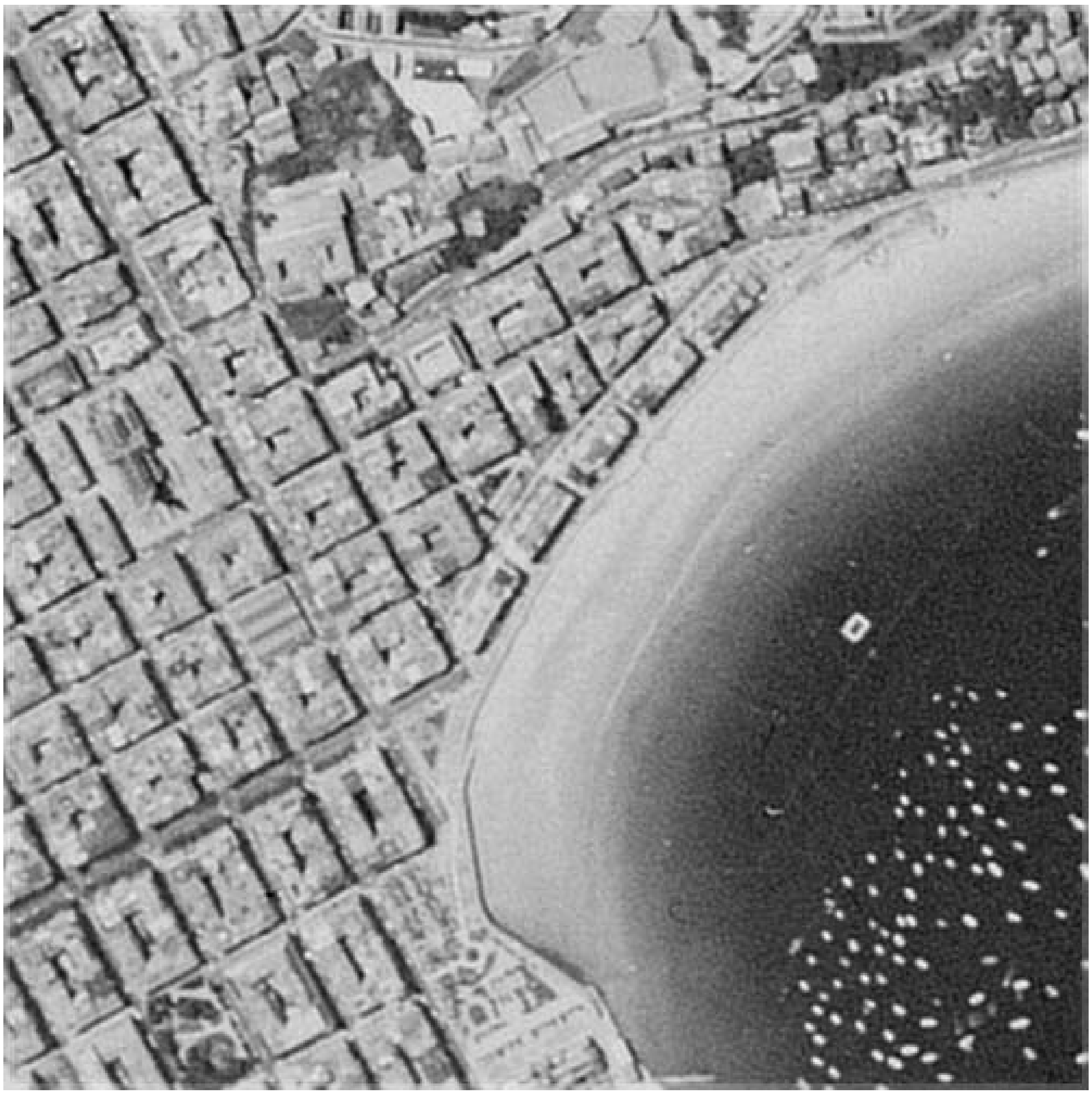

Figure 4.12: Reconstruction of the image of Fig. 4.11 using the FMAPE algorithm.

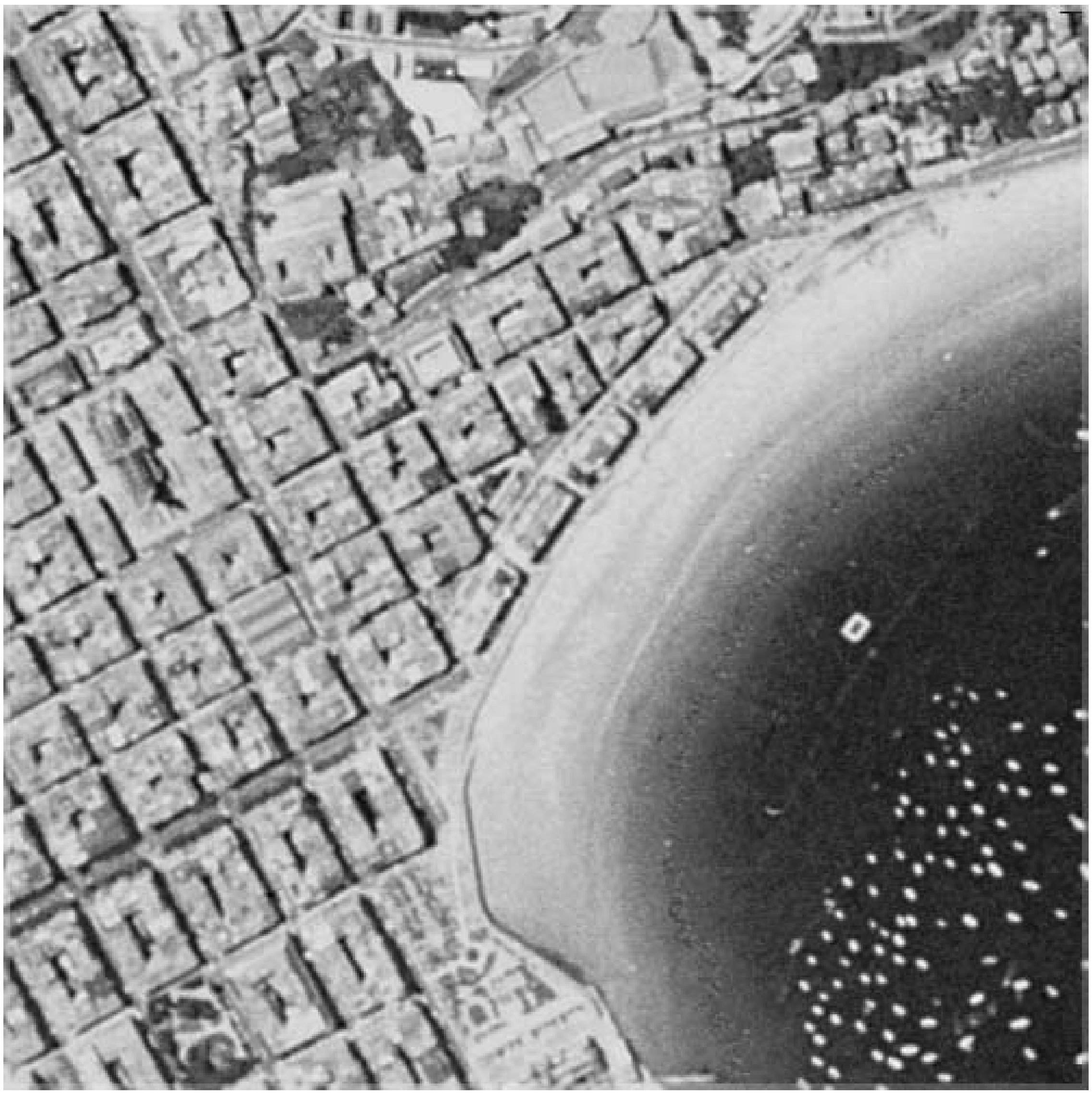

Figure 4.13: Reconstruction of the image of Fig. 4.11 using a neural network trained with the reconstructed image of Fig. 4.12.

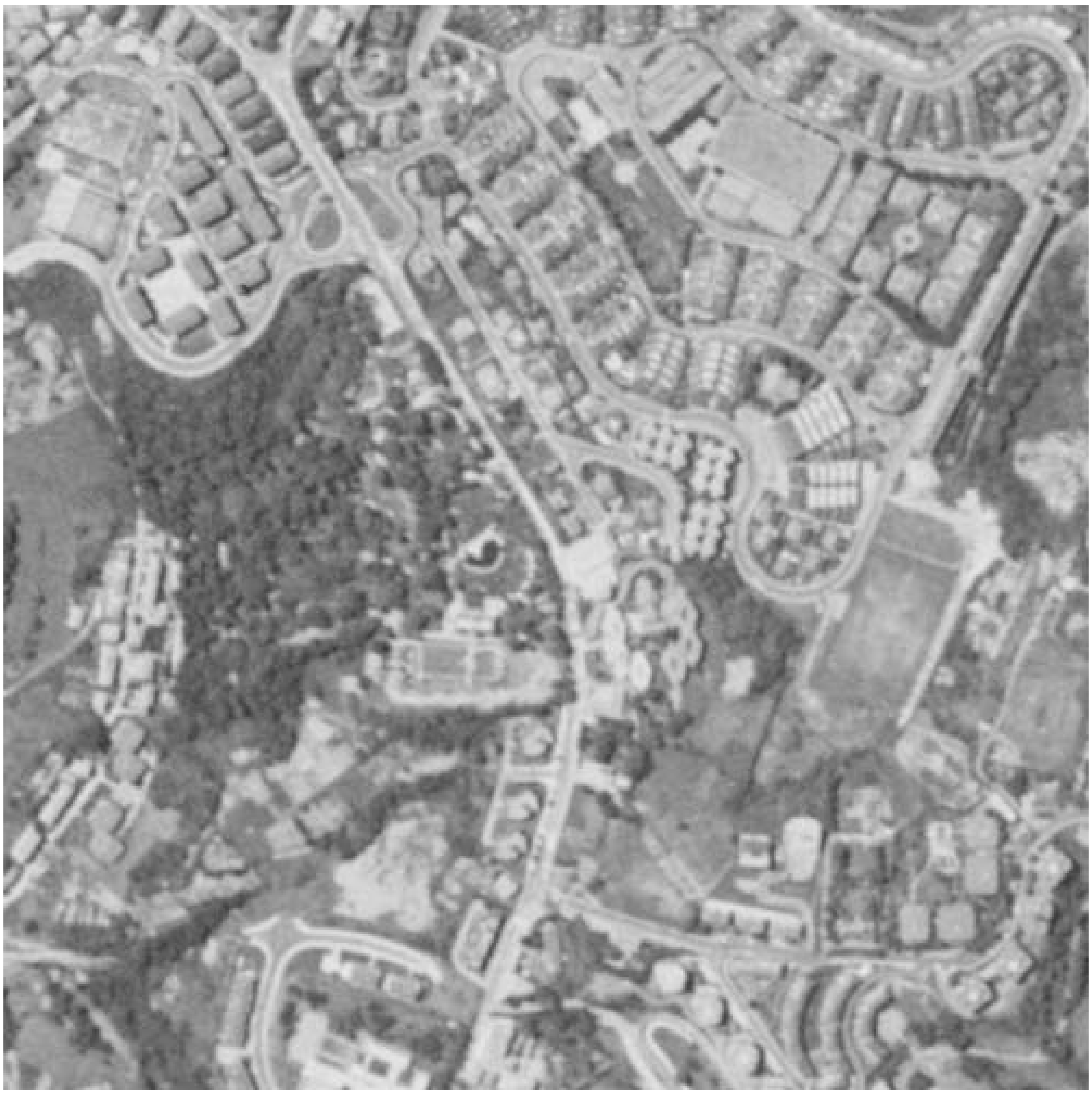

Figure 4.14: Another noisy image.

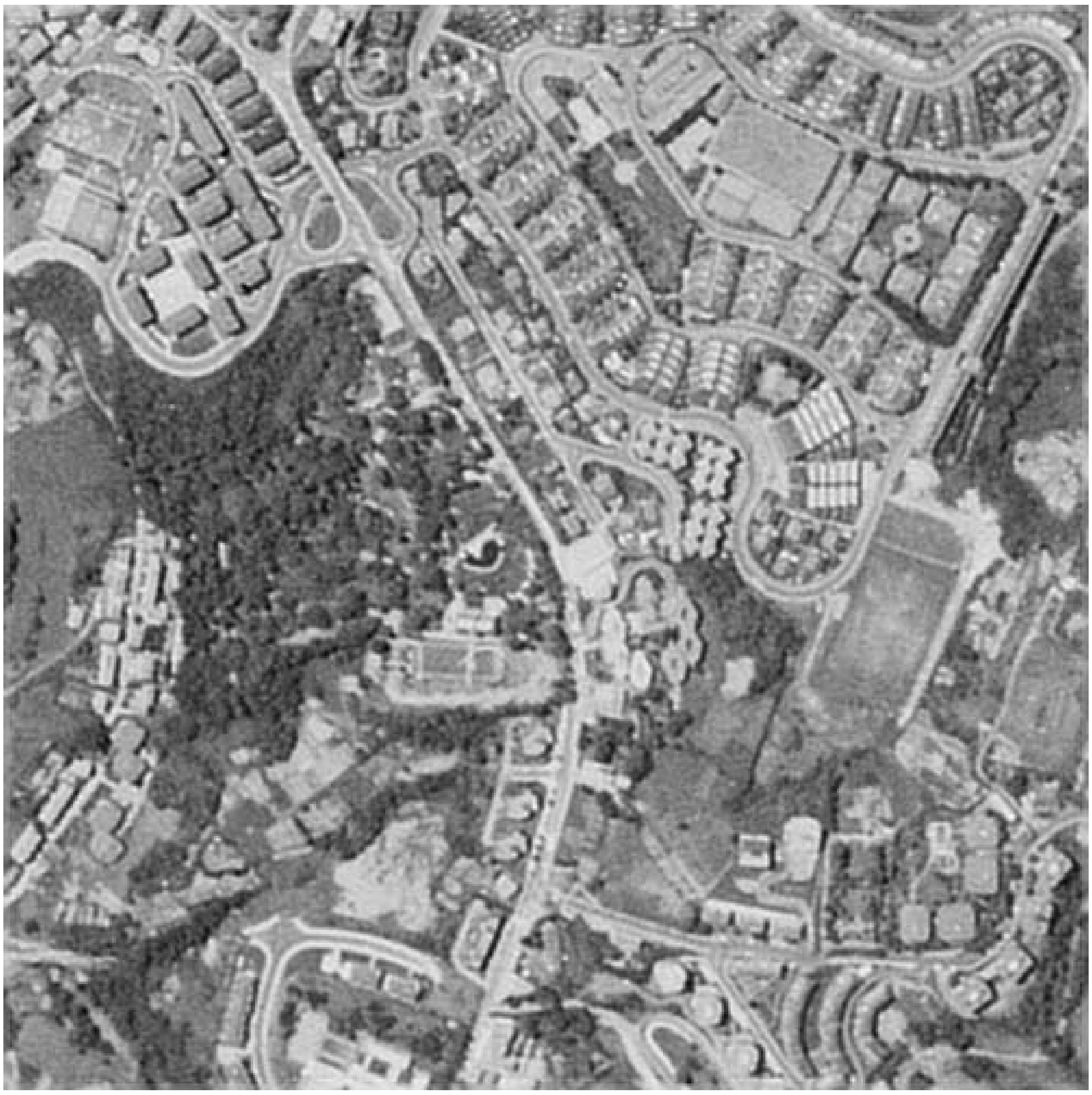

Figure 4.15: Reconstruction of the image of Fig. 4.14 using the FMAPE algorithm.

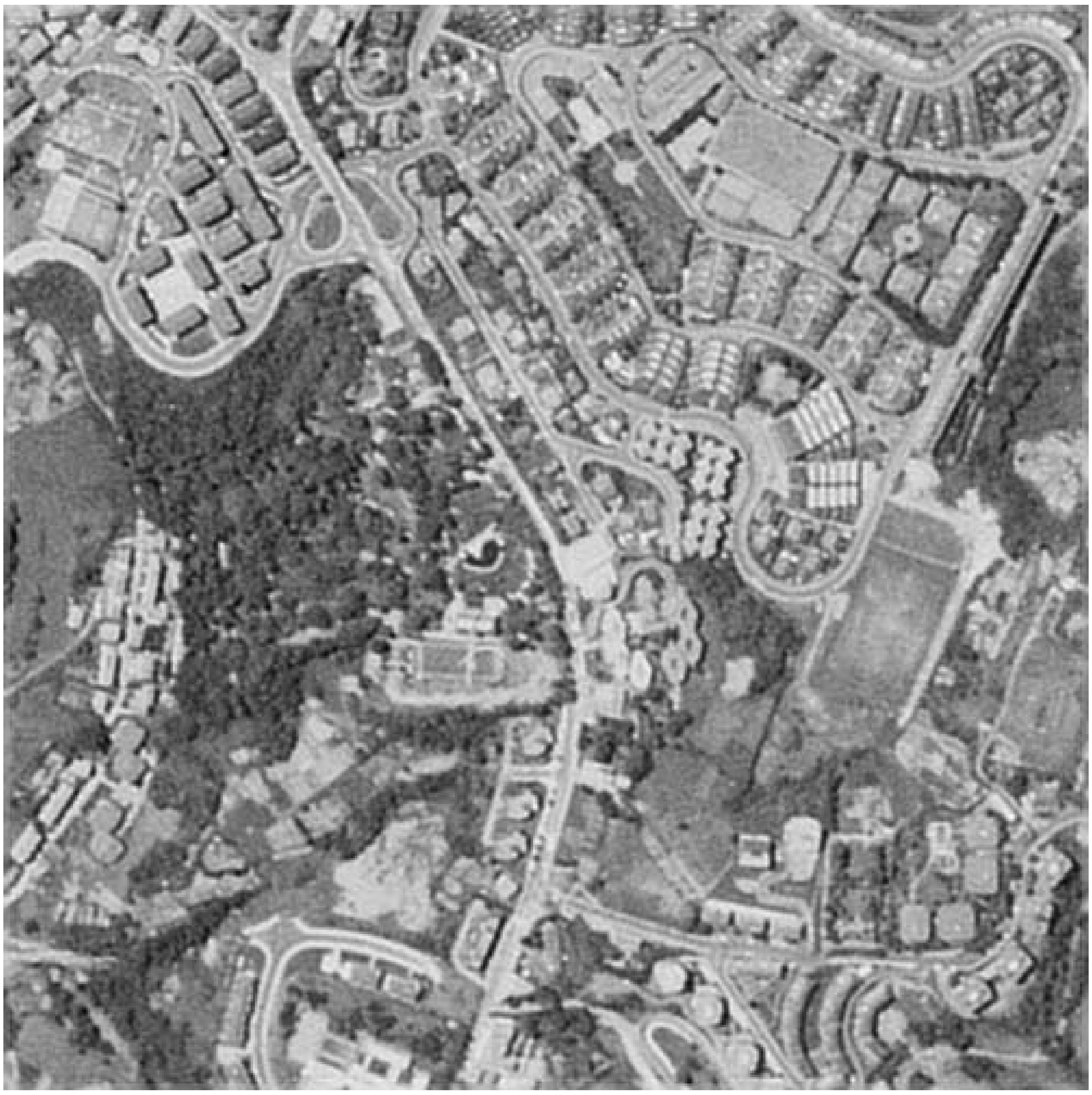

Figure 4.16: Reconstruction of the image of Fig. 4.14 using a neural network trained with the reconstructed image of Fig. 4.12.

be viewed as an embedding operator of the previous subspace into the initial input space. Since self-supervised back-propagation requires these functions to minimize the euclidean distance between each input point and its output, what we are doing is to approximate the identity function by the composite map $f_2 \circ f_1$, where f_1 makes the compression and f_2 the decompression. The quality of the whole process depends basically on the freedom given to these functions during the training of the network, the ability to find a good minimum of the error function, and the difference of dimensions between the input distribution and the bottle-neck layer.

In the simplest case, i.e. with just a hidden layer and linear activation functions, it is possible to show that self-supervised back-propagation is equivalent to *principal component analysis* [70]. This means that the solution found is the best possible one, if the search is restricted to linear compressions and decompressions. One step forward consists in the use of sigmoidal activation functions, thus introducing non-linearities [71]. However, the best way of improving the results is through the addition of new hidden layers before and after the bottle-neck layer, thus giving more freedom to the functions f_1 and f_2 [57].

The compression method

The application of self-supervised back-propagation for *image compression* could be done, for instance, in the following way (we will suppose, for simplicity, that the image is 1024×1024 with 256 grey levels):

1. A 16:25:12:2:12:25:16 multilayer neural network is initialized. In this case, the bottle-neck layer is the fourth one.

2. A cell of 4×4 pixels is chosen at random on the image, then being introduced to the net as an input pattern.

3. Next, the error between the output and the input patterns is back-propagated through the net.

4. The last two steps are repeated until enough iterations have been performed.

5. The f_1 is read as the half of the network from the input layer to the two neck units, and the f_2 as the other half from the bottle-neck layer to the output units. Thus, the compression transforms a 16-dimensional distribution into a simpler 2-dimensional one.

6. The original image is divided in its 65 536 cells of 4×4 pixels each, and f_1 is applied to all these subarrays. The two outputs per cell (the neck states) are stored, constituting the compressed image. Thus, 16 pixels will have been replaced by two numbers, which if they are stored with a precision of a byte each, the minimum compression rate achieved is of 8 (a further reversible

compression could be applied to this compressed image, increasing in some amount the final compression rate).

7. Once the image has been compressed, the f_1 is discarded or used for the compression of other similar images, likewise what was done for the reconstruction of images in Subsect. 4.3.1. Of course, the decompression function f_2 has to be stored together with the compressed image.

Similar improvements to those mentioned in Subsect. 4.3.1 could be proposed, such as the training using more than one images, or the preprocessing of the patterns to get rid of the dependence on the contrast and brightness of the image. Another possibility is the substitution of the f_2 function by a new decompressing function which would take into account the neighbouring cells (in its compressed format). That is, a new 10:12:25:16 network could be trained to decompress the image, using as input the neck-states of a cell plus those of their four neighbours, and as outputs the pixel values of the central cell itself.

Maximum information preservation and examples

In Fig. 4.17 we have plotted the 2-dimensional compressed distribution which corresponds to a 16-dimensional data distribution (obtained from four radiographies of the sort described above) after a large enough number of training steps. It is clear that this result is not optimal since most of the space available is free, and the distribution is practically 1-dimensional. The reason why this distribution acquires this funny shape lies in the *saturation* of the output sigmoids of f_1, i.e. the signal which arrives at the neck units is far beyond the 'linear' regime of the sigmoids. Consequently, the loss of information is greater than desirable.

To accomplish a *maximum information preservation*, the compressed distribution should be uniform in the unit $[0,1]^2$ square. Our first proposal is the introduction of a *repulsive term* between pairs of compressed patterns, together with periodic boundary conditions. More precisely, calling $\boldsymbol{\xi}^{(\mathrm{neck})1}$ and $\boldsymbol{\xi}^{(\mathrm{neck})2}$ the states of the m-dimensional bottle-neck layer of two different cells, an external error of the form

$$E^{(\mathrm{neck})} = -\frac{\lambda}{2} \sum_{k=1}^{m} \min \left\{ \left| \xi_k^{(\mathrm{neck})1} - \xi_k^{(\mathrm{neck})2} \right|^a , \left(1 - \left| \xi_k^{(\mathrm{neck})1} - \xi_k^{(\mathrm{neck})2} \right| \right)^a \right\} \tag{4.55}$$

is added to the usual squared error function, being back-propagated during the training process (the only difference to standard back-propagation is that the $\Delta_k^{(\mathrm{neck})\mu}$ variables have an extra contribution). Since the state of the neck units only depends on f_1, this modification does not affect the decompression function f_2. The new λ parameter takes into account the relative importance of the repulsive term in front of the quadratic error. In computer simulations, we have

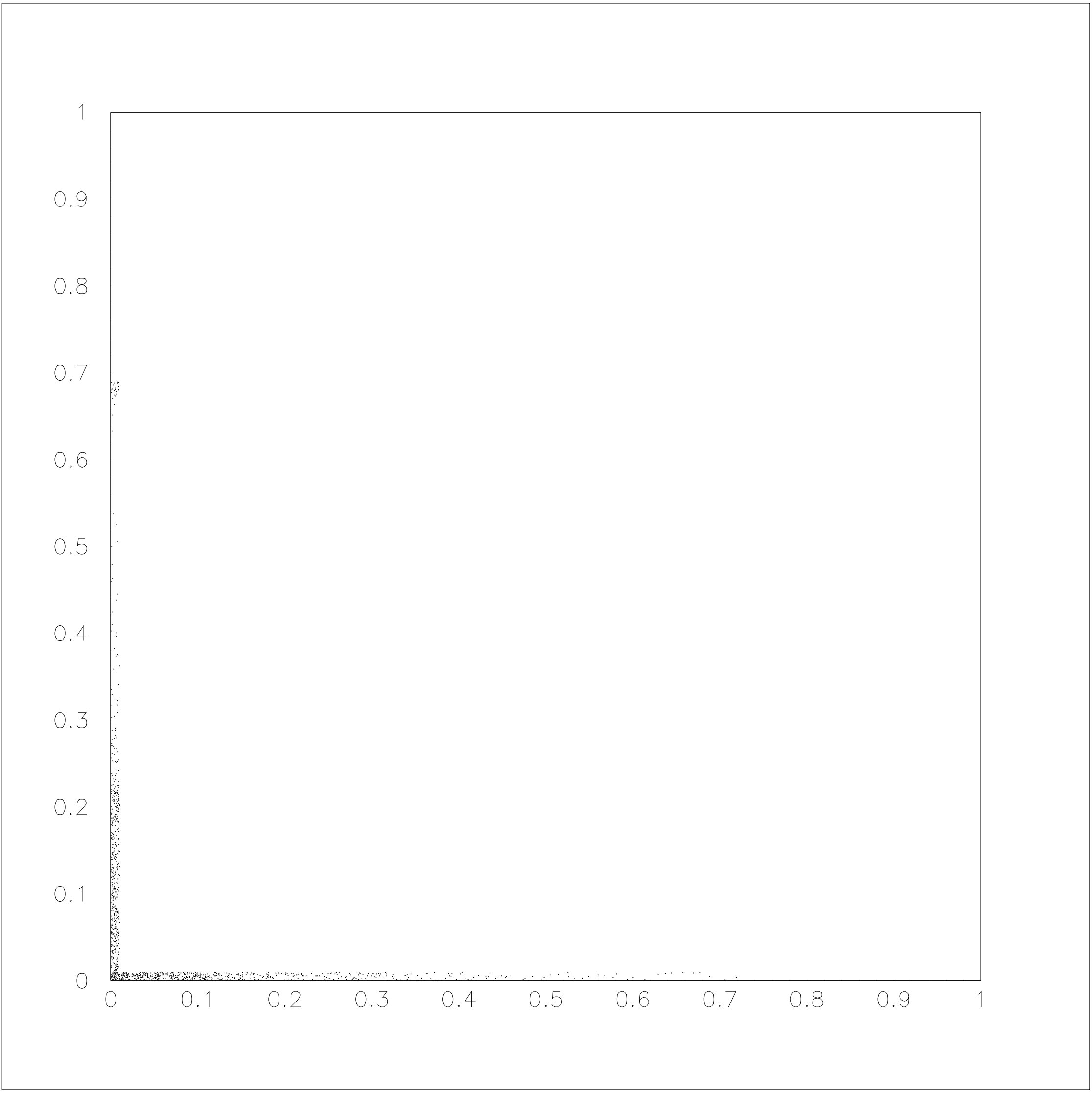

Figure 4.17: Compressed images distribution obtained with the self-supervised back-propagation.

found that good results are obtained if the λ is chosen such that both errors are of the same order of magnitude, and with $a = 1$.

Fig. 4.18 shows the compressed distribution of the same data that in Fig. 4.17, but with our first modified version of the learning. Although Fig. 4.18 is not a true uniform distribution, at least its appearance is now really 2-dimensional. The benefits are evident: not only the error between inputs and outputs is reduced to less than a half (from $E = 115.0$ in Fig. 4.17 to $E = 56.9$ in Fig. 4.18) but also the quality of the decompressed image is clearly superior.

A second method of eliminating the dangerous saturations is much more simple: the replacement of the sigmoidal activation functions by *sinusoidal* ones. The sinus function cannot saturate because of its periodicity and the fact that it has no 'flat' zones, whereas the sigmoids are only not constant in a small interval around the origin. Fig. 4.19 shows the compressed distribution with a network made of sinusoidal units, producing a final error of $E = 36.2$, even better than that with the first method.

The lowering of the error means that our methods have been able to move the learning away from a local minimum. This is a quite remarkable achievement, since it opens the application of these methods to more general problems. In particular, they may be applied to any layer of a network which suffers from saturation problems.

In order to compare the performances of the different methods proposed in this subsection we have applied them to the thorax in Fig. 4.20. For instance, Fig. 4.21 shows the result of compressing it with the repulsive term and decompressing it taking into account the neighbours of the cells. Since the differences are hard to see, it is better to compute them (between each decompressed image and the original one) and show them directly. Thus, Fig. 4.22 corresponds to a standard self-supervised learning, Fig. 4.23 to a learning using the neighbours, Fig. 4.24 to a learning without neighbours but with the repulsive term, and Fig. 4.25 to a learning using both the neighbours and the repulsive terms. They make it clear that both modifications (neighbours and repulsive term) significantly improve the quality of the compressions. The comparison between the success of the repulsive term and of the sinusoidal activation functions is given in Figs. 4.26 and 4.27, the second one seeming to produce better results. Finally, the result obtained by the standard image compression method known as JPEG is given in Fig. 4.28. The compression rates in all these cases have been about 13.

4.3.3 Time series prediction

The proliferation of mathematical models for the description of our world reveals their success in the prediction of the behaviour of a huge variety of systems. However, there exist situations in which no valid model can be found, or in which they cannot give quantitative answers. For instance, the evolution of the stock exchange is basically unpredictable, since the number of influencing variables is

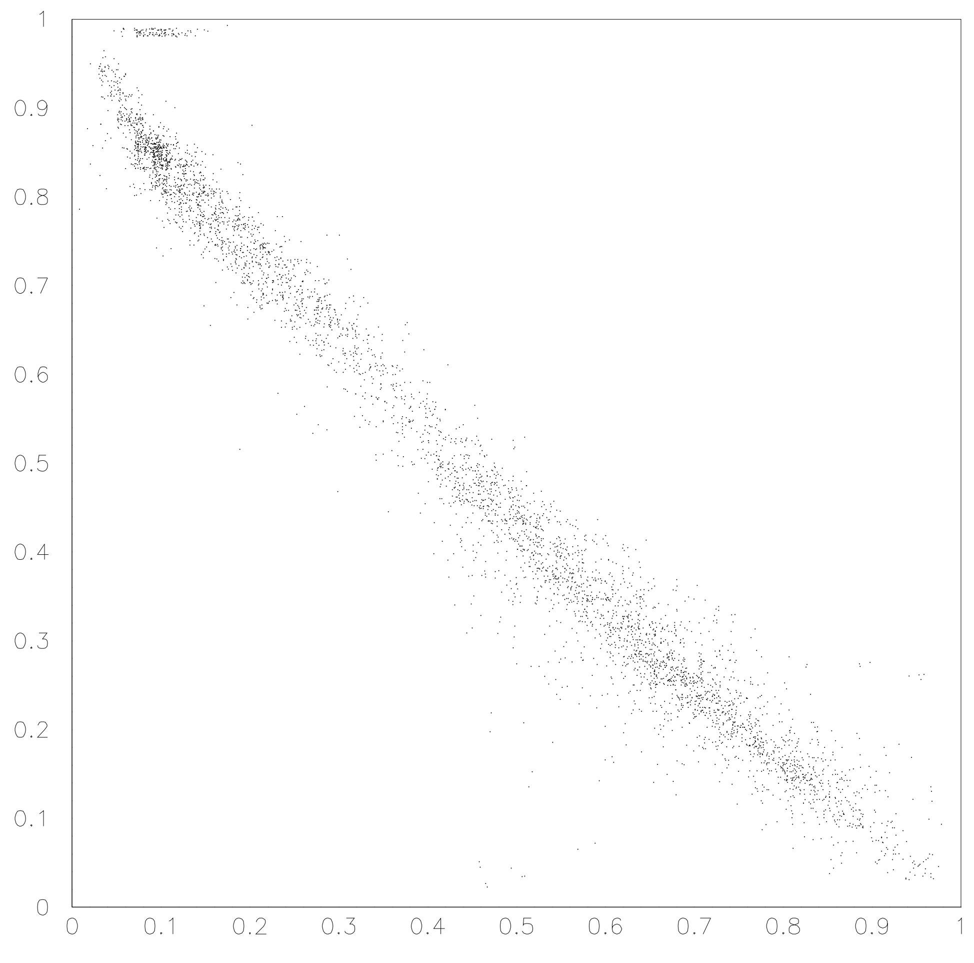

Figure 4.18: Compressed images distribution obtained with our self-supervised back-propagation with a repulsive term.

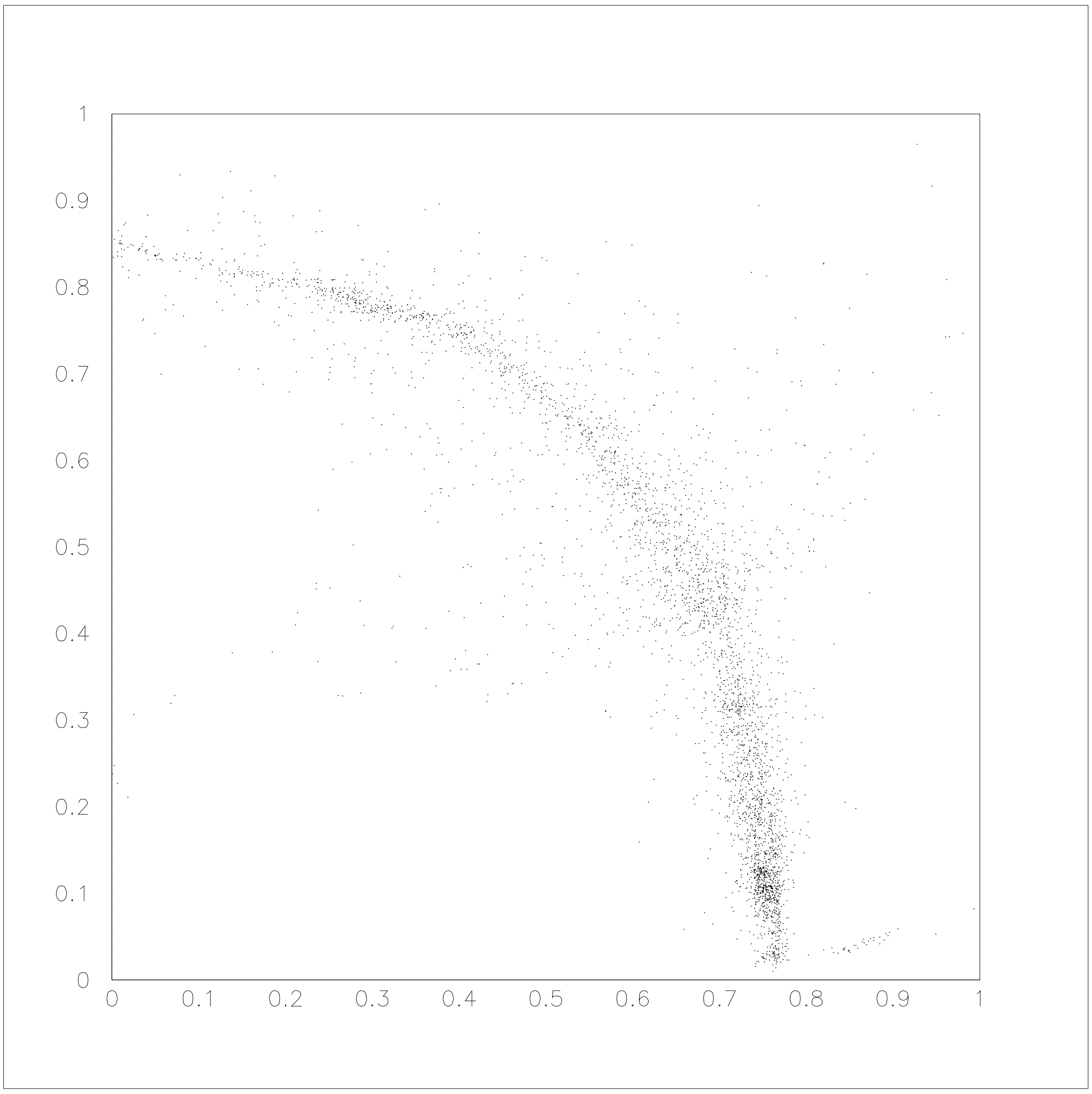

Figure 4.19: Compressed images distribution obtained with our self-supervised back-propagation with sinusoidal units.

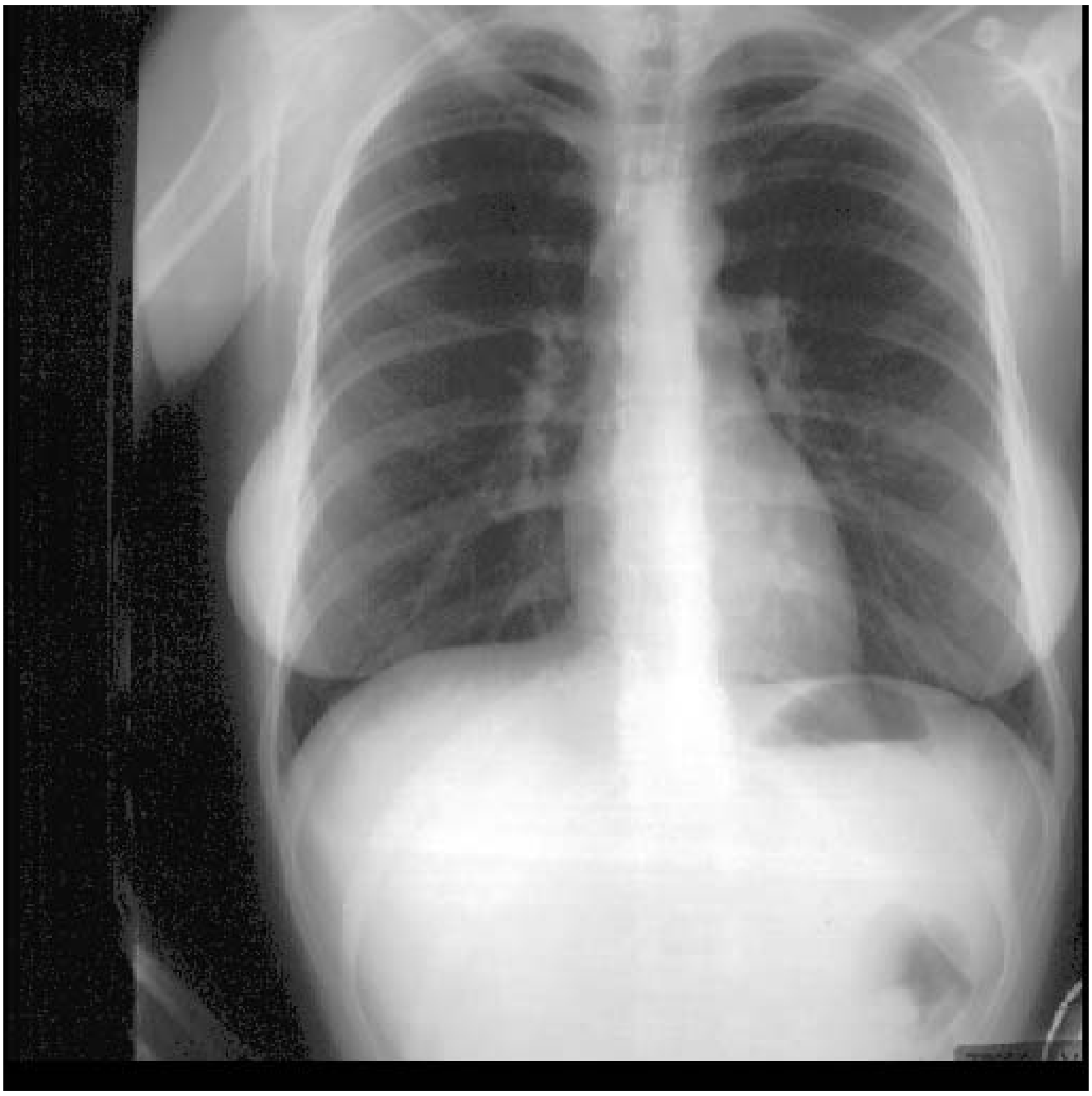

Figure 4.20: Thorax original.

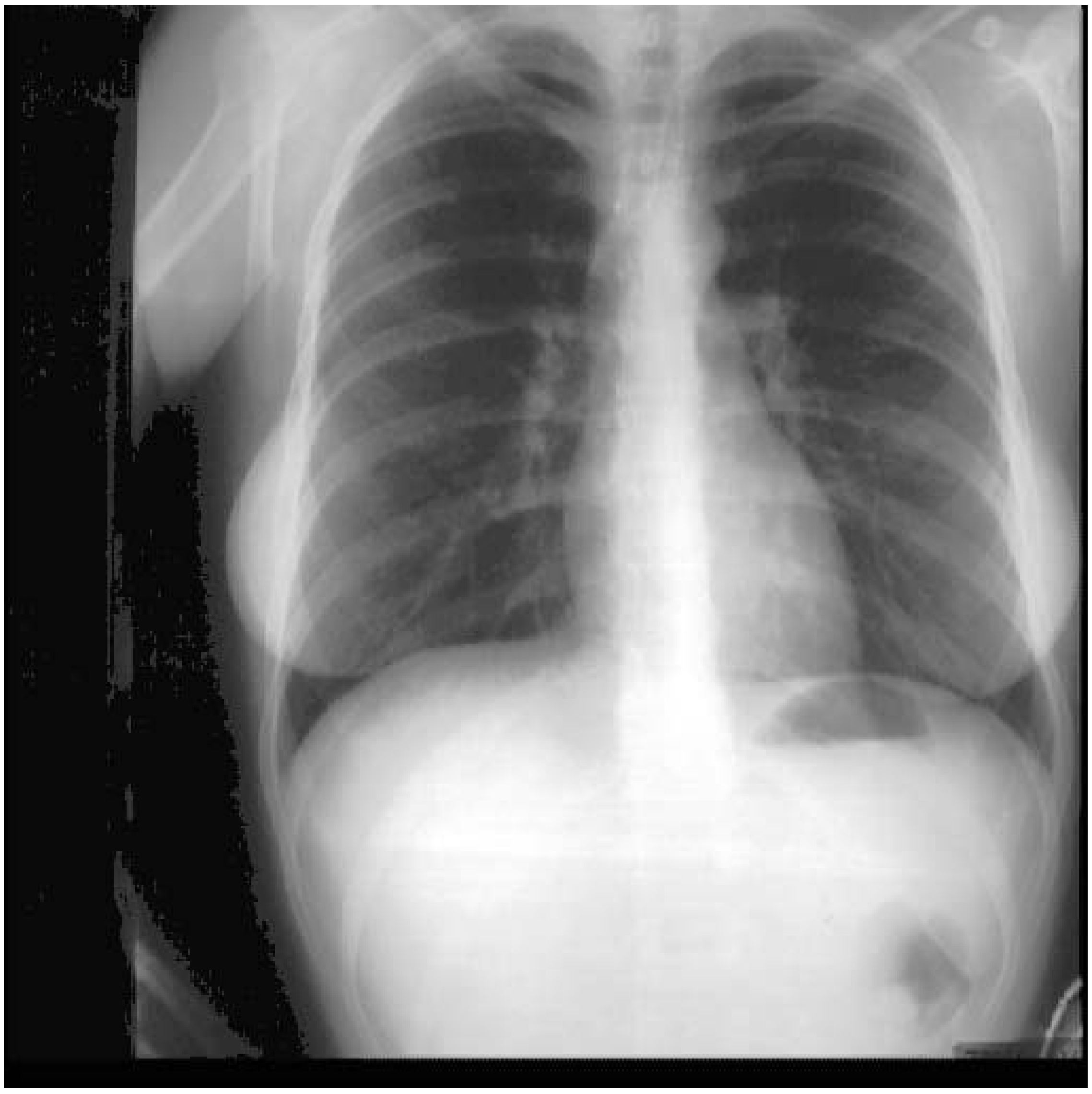

Figure 4.21: Thorax compressed with the repulsive term and decompressed using the neighbours.

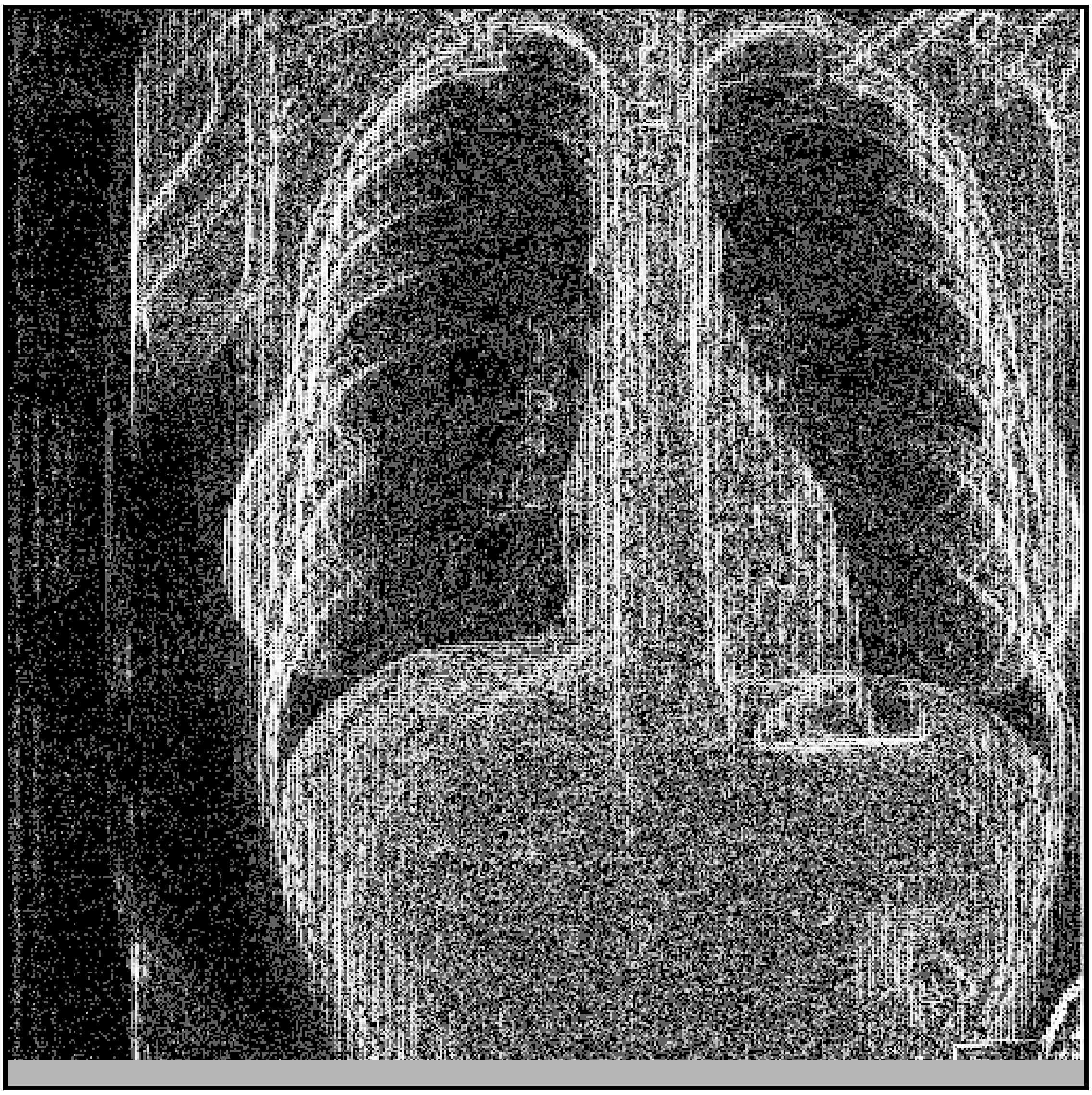

Figure 4.22: Difference between the original of the thorax and the compressed and decompressed with standard self-supervised back-propagation.

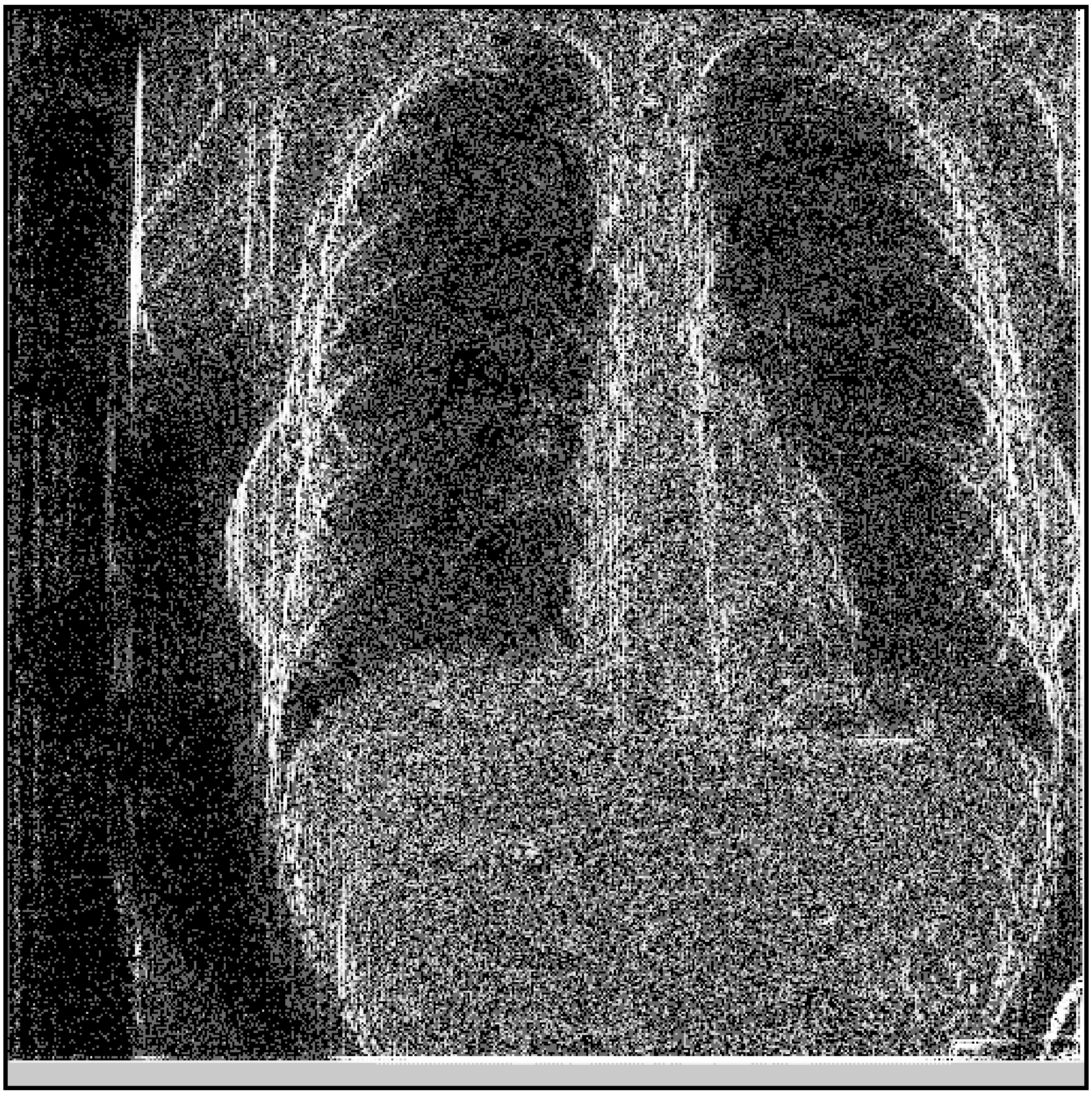

Figure 4.23: Difference between the original of the thorax and the decompressed making use of the neighbours.

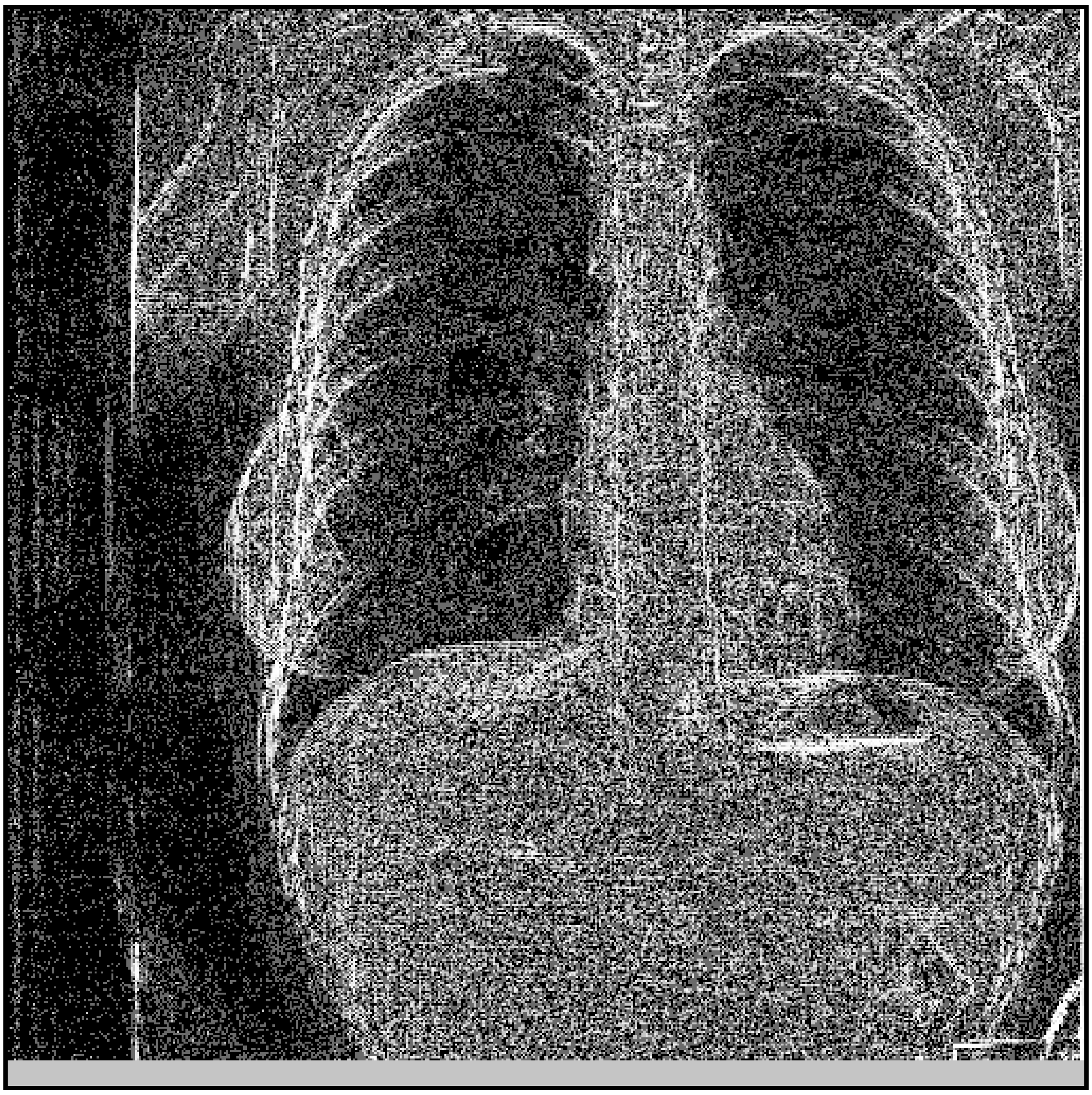

Figure 4.24: Difference between the original of the thorax and the compressed using the repulsive term.

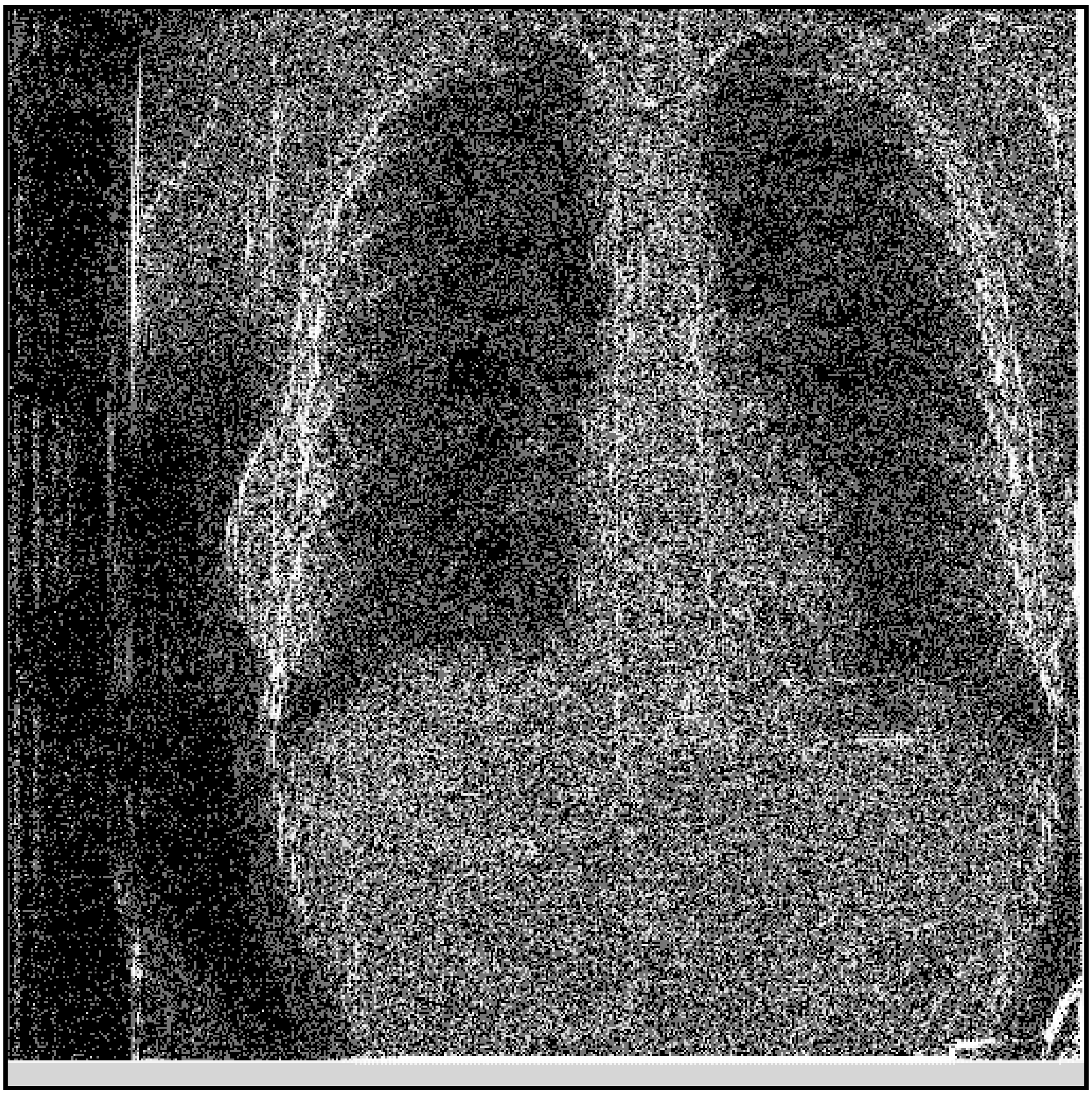

Figure 4.25: Difference between the original of the thorax and the learnt using the repulsive term and the neighbours.

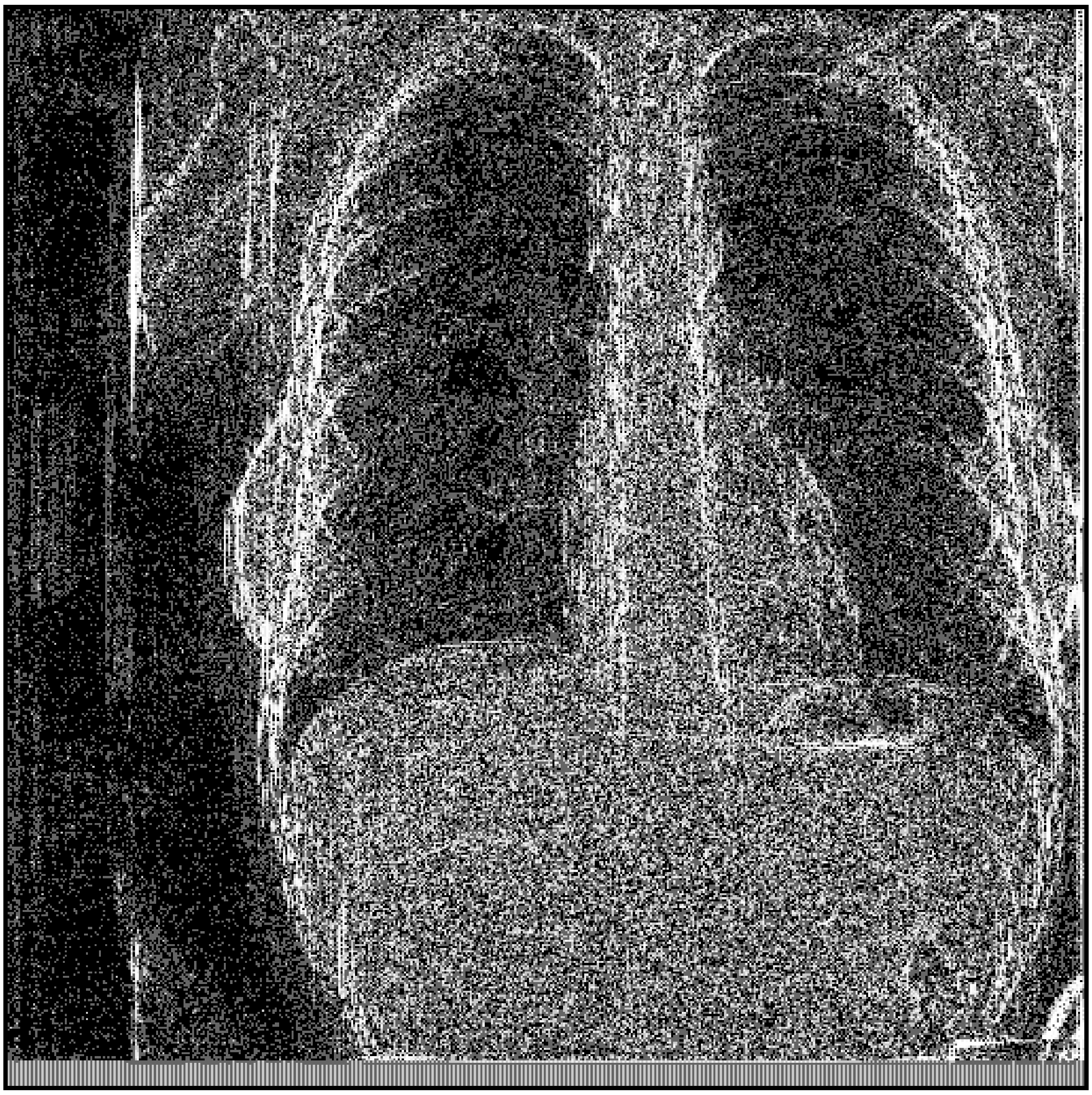

Figure 4.26: Difference between the original of the thorax and the learnt using four different images, the repulsive term and the neighbours.

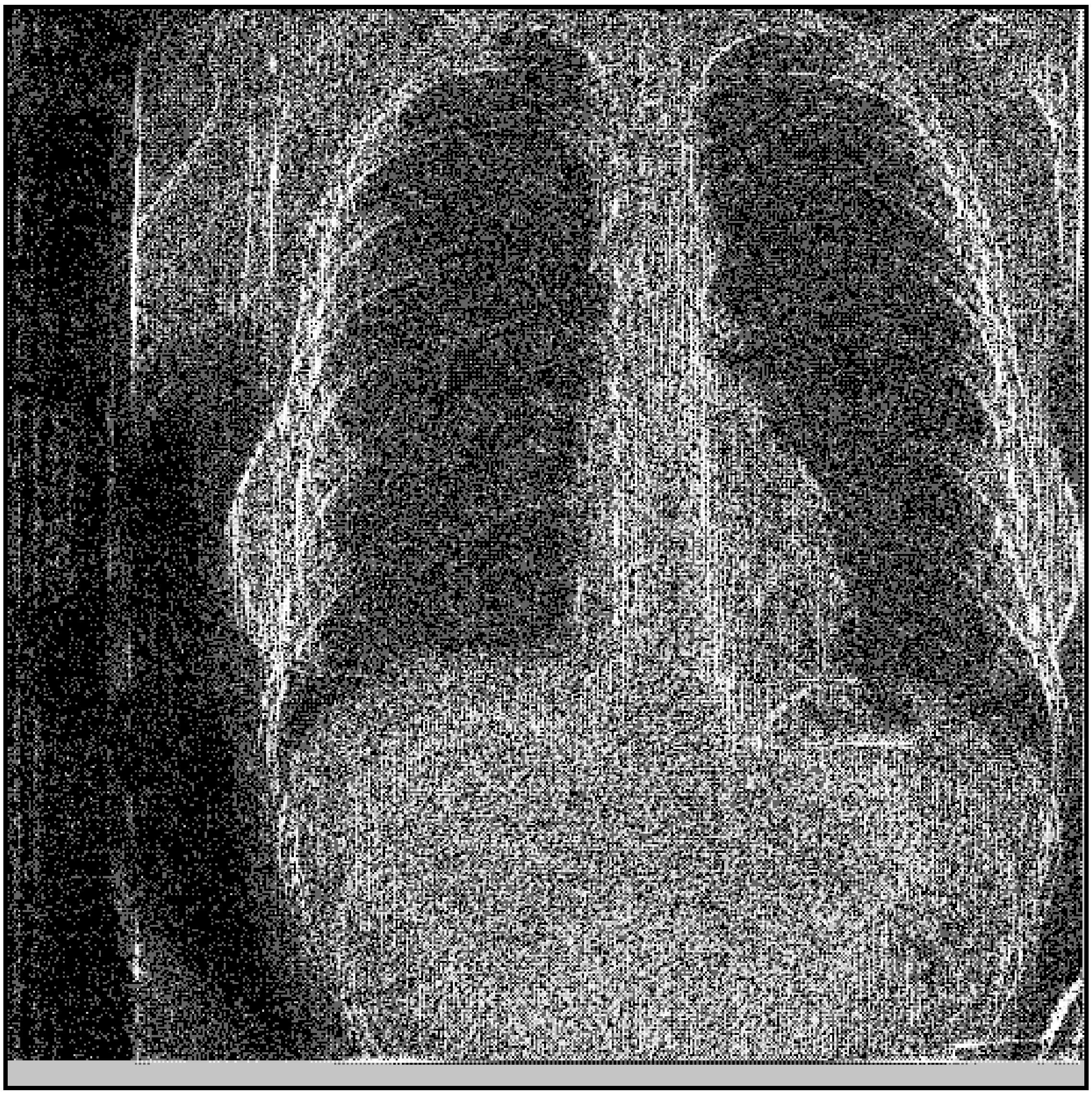

Figure 4.27: Difference between the original of the thorax and the learnt using four different images and sinusoidal activation functions.

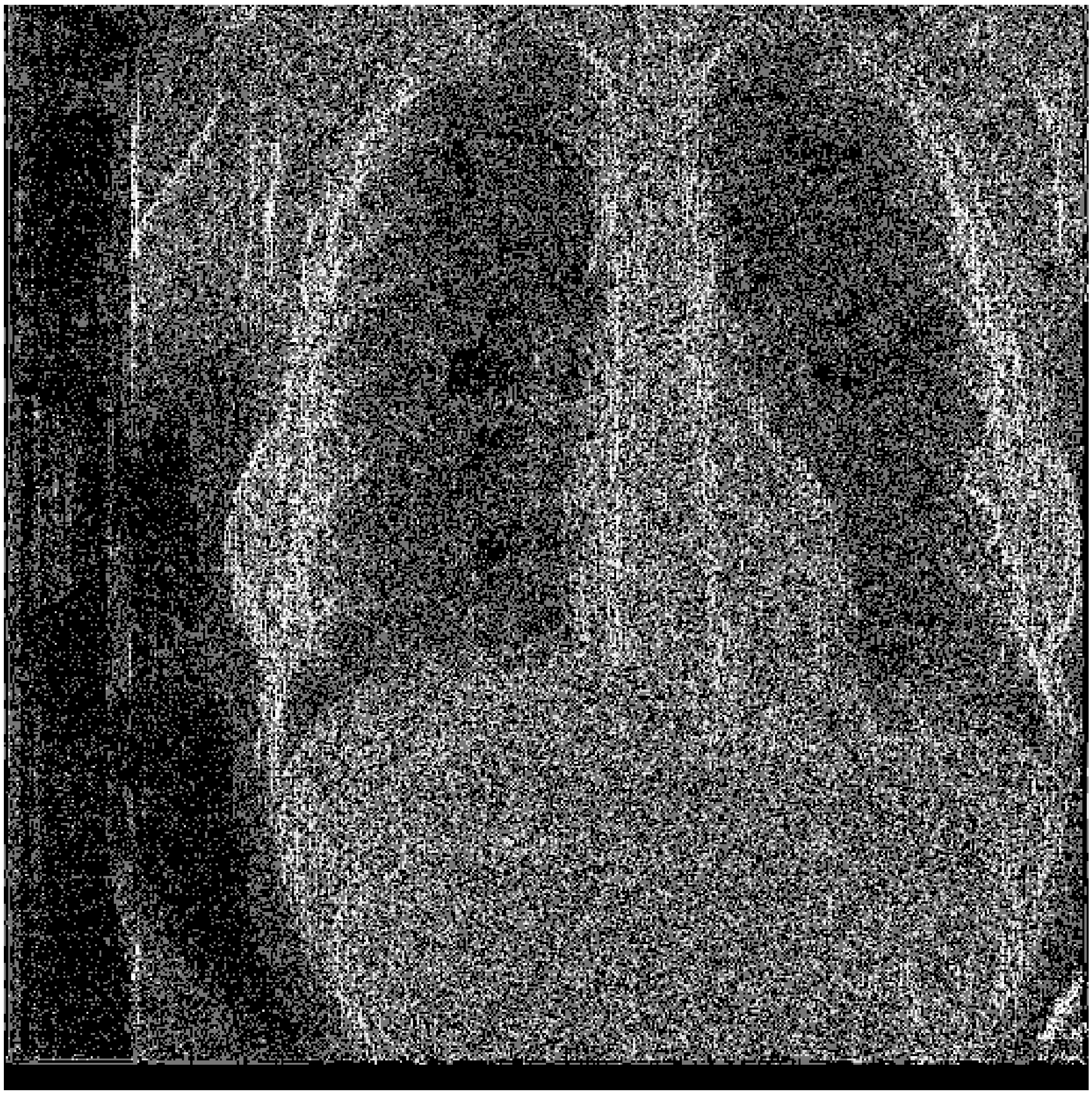

Figure 4.28: Difference between the original of the thorax and the compressed and decompressed with the JPEG algorithm.

too large and some of them are not numerical. A possible approach to these problems consists in the recording of some of the most relevant variables during a period of time large enough, and then to look for some regularities within these experimental data. Usually, the variable to be predicted is supposed to have a certain functional dependence on the last values of the series, and afterwards the function which best fits the data is calculated.

The *autoregressive integrated moving average* (ARIMA) models are among the most commonly used methods for *time series prediction*. They suppose that the dependence between a number of consecutive members of the series is linear, taking also into account that a random noise acts at each time step. The advantage of neural networks in front of these models is its non-linearity. In fact, in [79] Weigend, Rumelhart and Huberman show that their tapped-delay neural networks with weight-elimination are even better than other non-linear methods when applied to the bench mark *sunspots series*.

Although tapped-delay neural nets have proved to work well, it seems that recurrent networks should be more adequate than feed-forward ones to deal with time series. We have tested them using the same scheme that in [79], i.e. we have trained feed-forward and recurrent networks with the yearly sunspots data from 1700 to 1920, and the data from 1921 to 1955 has been used for the evaluation of the predictions.

In Fig. 4.29 we show a comparison between the predictions of feed-forward nets obtained with the Weigend, Rumelhart and Huberman architecture 12:8:1 (solid lines), and two different sorts of trainings with a recurrent architecture 1:3:4d:3:1 ('4d' means a four units layer with additional one time delayed weights between all of them), using the back-propagation through time of Subsect. 4.1.3. In the case termed as 'next' the error function has contributions from all the times steps, since at each time step the desired output is the sunspots number which follows that of the input (dashed lines), whereas in the 'last' case the desired output is shown only after the previous 12 sunspots number have been introduced to the net (dotted lines). In the horizontal axis we have put the number of steps-ahead of the prediction, and in the vertical the average relative variance. Each line is the result with lowest error among five different repetitions of the training, once the learning parameters have been properly adjusted. The plot shows that our recurrent nets perform better either at short and long-term predictions, and that the 'next' method is preferable for one-step-ahead predictions. The same applies to Fig. 4.30, obtained with the architecture 1:3d:4:3d:1, which has two recurrent layers. Finally, a typical example of the run of a recurrent solution over the whole sunspots series is given in Fig. 4.31.

It should be emphasized that recurrent nets do not learn when sigmoidal activation functions are used. Thus, taking advantage of the results of Sopena et al (see e.g. [76]), we have substituted them by linear ones in the recurrent units, and by sinusoidal ones in the rest of the neurons.

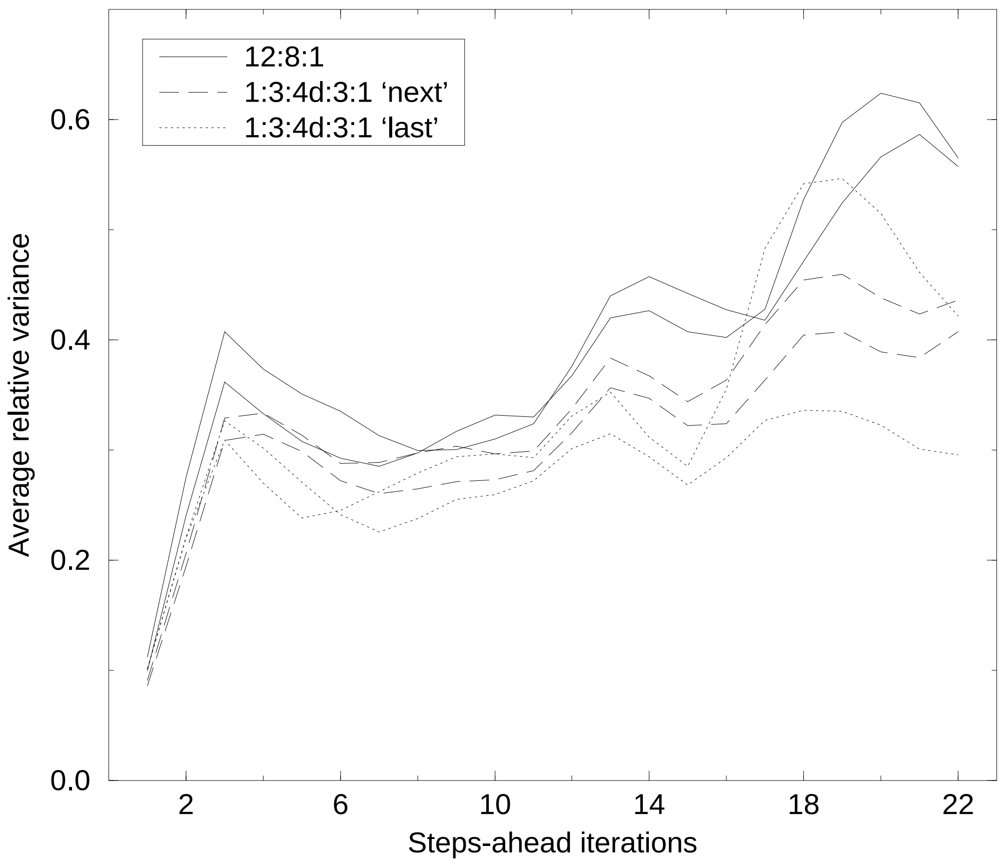

Figure 4.29: Average relative variance of the predictions of the sunspots series at different numbers of step-ahead iterations.

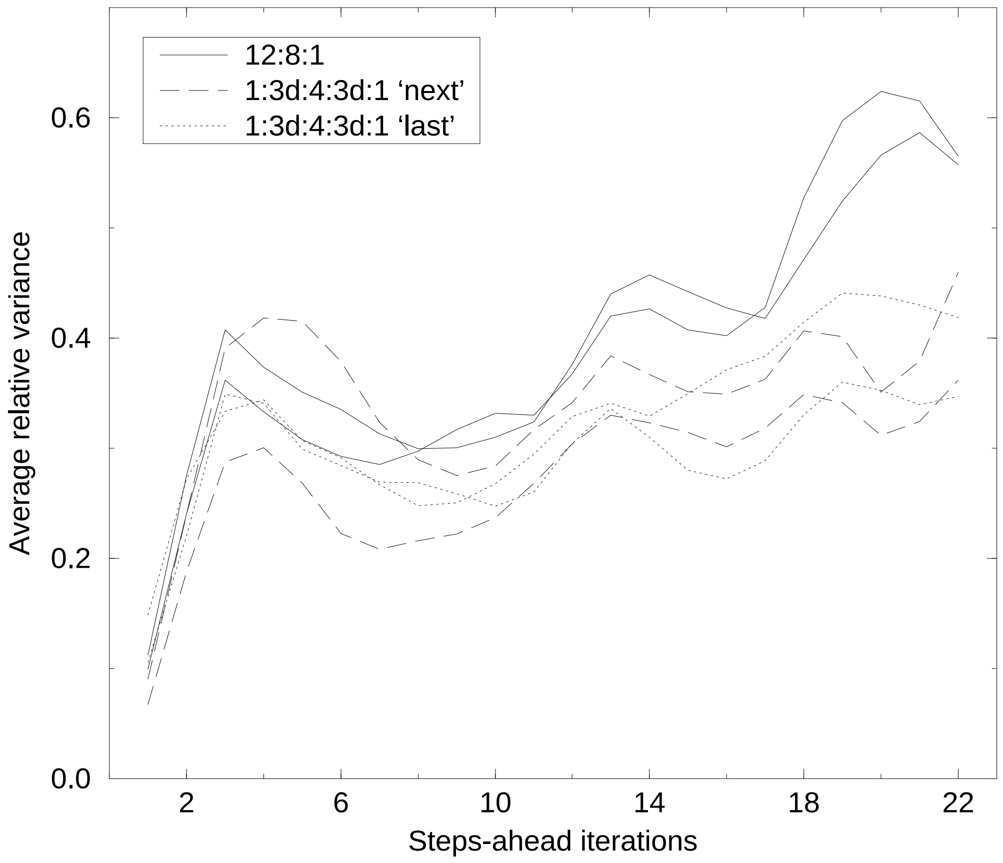

Figure 4.30: Average relative variance at different numbers of step-ahead iterations.

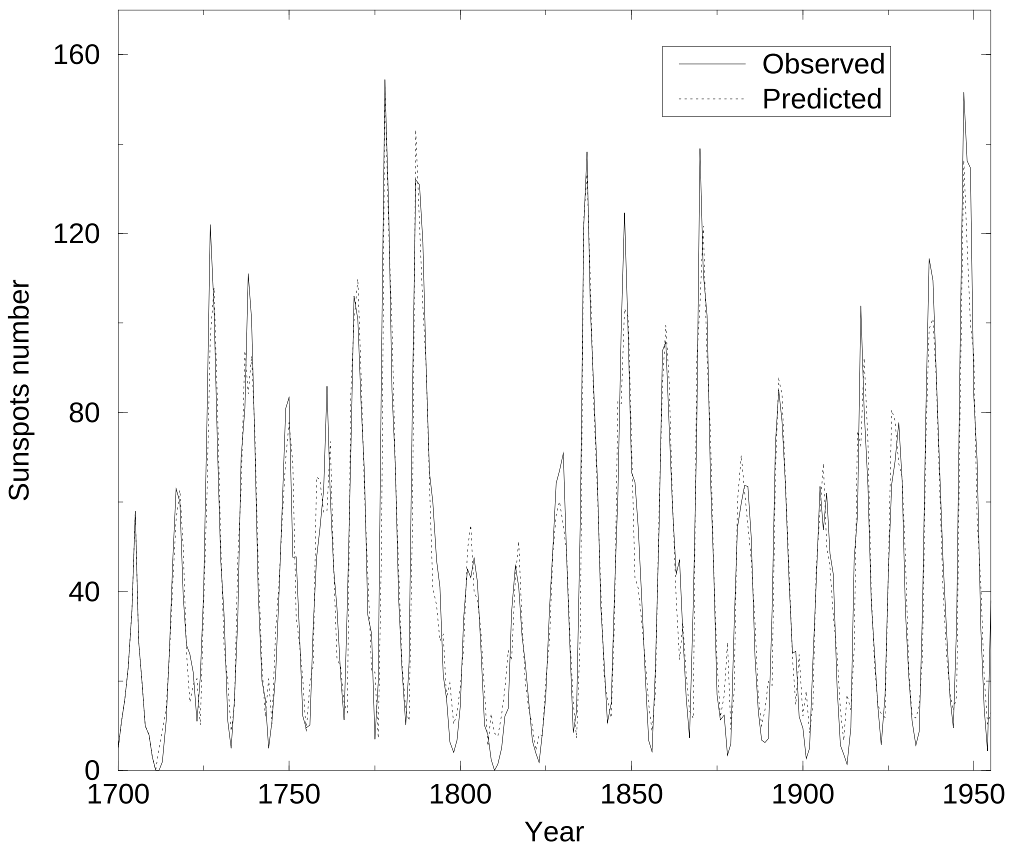

Figure 4.31: A single step-ahead prediction of the sunspots series.

Chapter 5

Conclusions

The main results that we have obtained and which are presented in this work are the following:

- Three different multilayer solutions to the problem of associative memory have been constructed, all of them sharing unlimited storage capacity, perfect recall and the removal of spurious minima and unstable states. Their retrieval power is optimal in the sense that the network's answer is selected by the maximal overlap criterion. The original contribution of these solutions has been the realization of such designs without introducing types of units different from those currently used in most neural network architectures.

- Neural network techniques for encoding-decoding processes have been developed. We have proved that it is not possible to encode arbitrary patterns with the minimal architecture, thus other non-optimal set-ups have been studied. In the simplest case of unary patterns, the accessibilities of the outputs have been calculated in two different situations: with and without thermal noise.

- A new perceptron learning rule which can be used with perceptrons made of multi-state units has been derived, and its corresponding convergence theorem has been proved. The definition of a perceptron of maximal stability has been enlarged in order to include these new multi-state perceptrons, and a proof of the existence and uniqueness of such optimal solutions has been outlined.

- The importance of the first hidden layer when multilayer neural networks with discrete activation functions are considered has been explained. As a consequence, several enhancements to the tiling algorithm have been proposed so as to increase the generalization ability of the trained nets.

- The unconstrained global minimum of the squared error criterion used in the back-propagation algorithm has been found using functional derivatives. The role played by the representation of the output patterns has been studied, showing that only certain output representations allow the achievement of the optimal Bayesian decision in classification tasks. Moreover, other error criterions have been analyzed.

- A method for the reconstruction of images from noisy data has been introduced and applied to two aerial images, showing that the results have a very competitive quality.

- Several methods based on self-supervised back-propagation have been devised for the compression of images. The new strategies admit more general applications, specifically to the diminishing of the loss of information produced by the saturation of the sigmoids.

- The performances of multi-layer feed-forward and recurrent networks have been compared when applied to time series prediction, showing that the second ones give, if proper activation functions are chosen, better predictions.

Appendix A

Accessibilities in terms of orthogonalities

Substituting (3.79) into (3.77) we obtain

$$\begin{aligned}
&f(h_1 \neq 0, \dots, h_R \neq 0) \\
&= 2^N - \sum_{j=1}^{R} \binom{R}{j} f(h_1 = 0, \dots, h_j = 0) + (-1)^2 \sum_{j=1}^{R} \sum_{k=1}^{R-j} \binom{R}{j} \binom{R-j}{k} \\
&\quad \times f(h_1 = 0, \dots, h_{j+k} = 0, h_{j+k+1} \neq 0, \dots h_R \neq 0)\,. \qquad \text{(A.1)}
\end{aligned}$$

By recurrent iterations of this sort of substitution in the last term each time, we finally end up with

$$\begin{aligned}
&f(h_1 \neq 0, \dots, h_R \neq 0) \\
&= 2^N + \sum_{l=1}^{R} (-1)^l \sum_{k_1=1}^{R} \sum_{k_2=1}^{R-k_1} \sum_{k_3=1}^{R-k_1-k_2} \cdots \sum_{k_l=1}^{R-k_1-\cdots-k_{l-1}} \binom{R}{k_1} \binom{R-k_1}{k_2} \\
&\quad \times \binom{R-k_1-k_2}{k_3} \cdots \binom{R-k_1-\cdots-k_{l-1}}{k_l} f(h_1 = 0, \dots, h_{k_1+\cdots+k_l} = 0) \\
&\equiv 2^N + \sum_{l=1}^{R} (-1)^l S_l\,, \qquad \text{(A.2)}
\end{aligned}$$

where we have introduced S_l as a shorthand for each l-dependent term in the multiple summatory. Defining the new indices

$$\begin{cases}
j_1 & \equiv & k_1 + \cdots + k_l \\
j_2 & \equiv & k_1 + \cdots + k_{l-1} \\
j_3 & \equiv & k_1 + \cdots + k_{l-2} \\
 & \vdots & \\
j_l & \equiv & k_1
\end{cases}$$

and observing that their respective ranges are

$$\begin{cases} l & \leq & j_1 & \leq & R \\ l-1 & \leq & j_2 & \leq & j_1 - 1 \\ l-2 & \leq & j_3 & \leq & j_2 - 1 \\ & & \vdots & & \\ 1 & \leq & j_l & \leq & j_{l-1} \end{cases}$$

we can put S_l as

$$\begin{aligned} S_l \; = \; & \sum_{j_1=l}^{R} \sum_{j_2=l-1}^{j_1-1} \sum_{j_3=l-2}^{j_2-1} \cdots \sum_{j_1=l}^{j_{l-1}-1} \binom{R}{j_l} \binom{R-j_l}{j_{l-1}-j_l} \binom{R-j_{l-1}}{j_{l-2}-j_{l-1}} \cdots \binom{R-j_2}{j_1-j_2} \\ & \times f(h_1 = 0, \ldots, h_{j_1} = 0) \, . \end{aligned} \qquad (A.3)$$

Multiplying and dividing each term by $j_1! \, j_2! \cdots j_{l-1}!$, this becomes

$$S_l = \sum_{j_1=l}^{R} \binom{R}{j_1} f(h_1 = 0, \ldots, h_{j_1} = 0) \sum_{j_2=l-1}^{j_1-1} \binom{j_1}{j_2} \sum_{j_3=l-2}^{j_2-1} \binom{j_2}{j_3} \cdots \sum_{j_l=1}^{j_{l-1}-1} \binom{j_{l-1}}{j_l} \, . \qquad (A.4)$$

Next, successively recalling that $\sum_{j=0}^{k} \binom{k}{j} = 2^k$ and exercising due care with the missing terms in each of the sums occurring, we get

$$\begin{aligned} S_l \; = \; & \sum_{j_1=l}^{R} \binom{R}{j_1} f(h_1 = 0, \ldots, h_{j_1} = 0) \sum_{j_2=l-1}^{j_1-1} \binom{j_1}{j_2} \cdots \sum_{j_{l-1}=2}^{j_{l-2}-1} \binom{j_{l-2}}{j_{l-1}} \\ & \times (2^{j_{l-1}} - 2) \\ = \; & \sum_{j_1=l}^{R} \binom{R}{j_1} f(h_1 = 0, \ldots, h_{j_1} = 0) \sum_{j_2=l-1}^{j_1-1} \binom{j_1}{j_2} \cdots \sum_{j_{l-2}=3}^{j_{l-3}-1} \binom{j_{l-3}}{j_{l-2}} \\ & \times (3^{j_{l-2}} - 3 \times 2^{j_{l-2}} + 3) \\ = \; & \sum_{j_1=l}^{R} \binom{R}{j_1} f(h_1 = 0, \ldots, h_{j_1} = 0) \sum_{j_2=l-1}^{j_1-1} \binom{j_1}{j_2} \cdots \sum_{j_{l-3}=4}^{j_{l-4}-1} \binom{j_{l-4}}{j_{l-3}} \\ & \times (4^{j_{l-3}} - 4 \times 3^{j_{l-3}} + 6 \times 2^{j_{l-3}} - 4) \\ \vdots \; & \\ = \; & \sum_{j_1=l}^{R} \binom{R}{j_1} f(h_1 = 0, \ldots, h_{j_1} = 0) \left[(-1)^l \sum_{k=1}^{l} (-1)^k \binom{l}{k} k^{j_1} \right] \, . \end{aligned} \qquad (A.5)$$

Now, let us focus on the quantity in square brackets. Using the notation

$$S(l, j) \equiv \sum_{k=1}^{l} (-1)^k \binom{l}{k} k^j \, , \qquad (A.6)$$

one can check the quite remarkable properties

$$\begin{aligned} S(l,j) &= 0\,,\ \text{for } 1 \le j < l\,, & \text{(A.7)}\\ \sum_{l=1}^{j} S(l,j) &= (-1)^j\,,\ \text{for } 1 \le j\,. & \text{(A.8)} \end{aligned}$$

whose proofs are given in Appendix B. Then, in terms of $S(l,j)$,

$$\sum_{l=1}^{R}(-1)^l S_l = \sum_{l=1}^{R}\sum_{j=1}^{R}\binom{R}{j} f(h_1=0,\dots,h_j=0)S(l,j)\,, \qquad \text{(A.9)}$$

where, by the first property, the range of the sum over j has been extended from $j=1$ to R changing nothing. As a result we can write

$$\sum_{l=1}^{R}(-1)^l S_l = \sum_{j=1}^{R}\binom{R}{j} f(h_1=0,\dots,h_j=0)\sum_{l=1}^{R} S(l,j)\,. \qquad \text{(A.10)}$$

Moreover, by virtue of the same property the sum over l can be restricted to the range from 1 to j, because the remaining terms give zero contribution, and then, applying the second one,

$$\sum_{l=1}^{R}(-1)^l S_l = \sum_{j=1}^{R}\binom{R}{j} f(h_1=0,\dots,h_j=0)(-1)^j\,. \qquad \text{(A.11)}$$

Consequently,

$$f(h_1\neq 0,\dots,h_R\neq 0) = 2^N + \sum_{j=1}^{R}(-1)^j\binom{R}{j} f(h_1=0,\dots,h_j=0)\,, \qquad \text{(A.12)}$$

which is (3.80).

Appendix B

Proof of two properties

B.1 Proof of $S(l,j) = 0\,,\ 1 \le j < l$

We start by considering the function

$$y_{(l,0)} \equiv (1-x)^l = \sum_{k=1}^{l} (-1)^k \begin{pmatrix} l \\ k \end{pmatrix} x^k \,. \tag{B.1}$$

Then, we make the definitions

$$y_{(l,1)} \equiv \frac{d}{dx}\, y_{(l,0)}(x) = \sum_{k=1}^{l} (-1)^k \begin{pmatrix} l \\ k \end{pmatrix} k x^{k-1} \,, \tag{B.2}$$

$$y_{(l,j+1)} \equiv \frac{d}{dx}\left(x y_{(l,j)}(x)\right) \ j \ge 1 \,, \tag{B.3}$$

the second one being a recurrent, constructive rule. In terms of these functions, the $S(l,j)$ of (A.6) is given by

$$S(l,0) = y_{(l,0)}(1) - 1\,,\ l \ge 1\,, \tag{B.4}$$

$$S(l,j) = y_{(l,j)}(1)\,,\ j \ge 1\,,\ l \ge 1\,. \tag{B.5}$$

Since $y_{(l,0)} = 0$, one realizes that

$$S(l,0) = -1\,,\ l \ge 1\,. \tag{B.6}$$

The next step is to show that $S(l,j) = 0$ for $1 \le j < l$. By taking sucessive derivatives, it is not difficult to notice that $y_{(l,k)}(x)$ is a sum of terms proportional to $(1-x)^{l-k}$, with $1 \le k < l$. Therefore

$$y_{(l,j)}(1) = 0 = S(l,j)\,,\ \text{for } 1 \le j < l\,. \tag{B.7}$$

B.2 Proof of $\sum_{l=1}^{j} S(l,j) = (-1)^j\,,\ j \geq 1$

We want to know the value of

$$\sum_{l=1}^{j} S(l,j) = \sum_{l=1}^{j}\sum_{k=1}^{l}(-1)^k \binom{l}{k} k^j\,. \tag{B.8}$$

Given that the binomial coefficients vanish for $l < k$, the second sum can be extended to the range from $k = 1$ to j and interchanged with the first afterwards

$$\sum_{l=1}^{j} S(l,j) = \sum_{k=1}^{j}(-1)^k k^j \sum_{l=1}^{j}\binom{l}{k}\,. \tag{B.9}$$

Further, by the same reasoning the l-summatory may now be restricted to $k \leq l$. Then, we obtain

$$\sum_{l=k}^{j}\binom{l}{k} = \binom{j+1}{k+1}\,. \tag{B.10}$$

Replacing this into the previous expression and making the index renaming

$$\left\{\begin{array}{rcl} N & \equiv & j+1 \\ r & \equiv & k+1, \end{array}\right. \tag{B.11}$$

we arrive at

$$\begin{aligned}\sum_{l=1}^{j} S(l,j) &= \sum_{r=2}^{N}(-1)^{r-1}(r-1)^{N-1}\binom{N}{r} \\ &= -\sum_{r=0}^{N}(-1)^{r}(r-1)^{N-1}\binom{N}{r} + (-1)^{N-1}\binom{N}{0}\,. \end{aligned} \tag{B.12}$$

The first term vanishes by a known property ([30] formula [0.154(6)]) and what remains reads

$$\sum_{l=1}^{j} S(l,j) = (-1)^j\,. \tag{B.13}$$

Bibliography

[1] D.J. Amit, H. Gutfreund and H. Sompolinsky, Statistical mechanics of neural networks near saturation, *Ann. Phys.* (N.Y.) **173** (1987) 30.

[2] J.K. Anlauf and M. Biehl, The AdaTron: an adaptive perceptron algorithm, *Europhys. Lett.* **10** (1989) 687.

[3] S. Bacci, G. Mato and N. Parga, Dynamics of a neural network with hierarchically stored patterns, *J. Phys. A: Math. Gen.* **23** (1990) 1801.

[4] A.R. Barron, Universal approximation bounds for superpositions of a sigmoidal function, *IEEE Trans. Information Theory* **39** (1993) 930.

[5] E.R. Caianiello, Outline of a theory of thought processes and thinking machines, *J. Theor. Biol.* **1** (1961) 204.

[6] G.A. Carpenter and S. Grossberg, A massively parallel architecture for a self-organizing neural pattern recognition machine, *Computer vision, graphics and image processing* **37** (1987) 54.

[7] G.A. Carpenter and S. Grossberg, ART2: self-organization of stable category recognition codes for analog input patterns, *Appl. Optics* **26** (1987) 4919.

[8] G. Cybenko, Approximations by superpositions of a sigmoidal function, *Math. Contr. Signals, Syst.* **2** (1989) 303.

[9] B. Denby, Neural networks for high energy physicists, Fermilab preprint Conf-90/94.

[10] B. Denby, M. Campbell, F. Bedeschi, N. Chris, C. Bowers and F. Nesti, Neural networks for triggering, Fermilab preprint Conf-90/20.

[11] J.S. Denker, D. Schwartz, B. Wittner, S. Solla, R. Howard, L. Jackel and J. Hopfield, Large automatic learning, rule extraction, and generalization, *Complex Systems* **1** (1987) 877.

[12] B. Derrida, E. Gardner and A. Zippelius, An exactly solvable asymmetric neural network model, *Europhys. Lett.* **4** (1987) 167.

[13] S. Diederich and M. Opper, Learning of correlated patterns in spin-glass networks by local learning rules, *Phys. Rev. Lett.* **58** (1987) 949.

[14] E. Domany and H. Orland, A maximum overlap neural network for pattern recognition, *Phys. Lett. A* **125** (1987) 32.

[15] R.O. Duda and P.E. Hart, *Pattern classification and scene analysis*, Wiley, New York (1973).

[16] E. Elizalde and S. Gómez, Multistate perceptrons: learning rule and perceptron of maximal stability, *J. Phys. A: Math. Gen.* **25** (1992) 5039.

[17] E. Elizalde, S. Gómez and A. Romeo, Encoding strategies in multilayer neural networks, *J. Phys. A: Math. Gen.* **24** (1991) 5617.

[18] E. Elizalde, S. Gómez and A. Romeo, Methods for encoding in multilayer feed-forward neural networks. In *Artificial Neural Networks*, A. Prieto (ed.), Lecture Notes in Computer Science **540**, Springer-Verlag (1991) 136.

[19] E. Elizalde, S. Gómez and A. Romeo, Maximum overlap neural networks for associative memory, *Phys. Lett. A* **170** (1992) 95.

[20] S.E. Fahlman and Lebiere, The cascade-correlation learning architecture. In *Advances in neural information processing systems II*, D.S. Touretzky (ed.), Morgan Kaufmann, San Mateo (1990) 524.

[21] W. Feller, *An introduction to probability theory and its applications*, Wiley, New York (1971).

[22] M. Frean, The upstart algorithm: a method for constructing and training feedforward neural networks, *Neural Computation* **2** (1990) 198.

[23] S.I. Gallant, Perceptron-based learning algorithms, *IEEE Trans. Neural Networks* **1** (1990) 179.

[24] E. Gardner, The space of interactions in neural network models, *J. Phys. A: Math. Gen.* **21** (1988) 257.

[25] Ll. Garrido and V. Gaitan, Use of neural nets to measure the τ polarization and its Bayesian interpretation, *Int. J. of Neural Systems* **2** (1991) 221.

[26] Ll. Garrido and S. Gómez, Analytical interpretation of feed-forward nets outputs after training, preprint UB-ECM-PF 94/14, Barcelona (1994).

[27] B. Gnedenko, *The theory of probability*, Mir, Moscow (1978).

[28] M. Golea and M. Marchand, A growth algorithm for neural network decision trees, *Europhys. Lett.* **12** (1990) 205.

[29] S. Gómez and Ll. Garrido, Interpretation of BP-trained net outputs. In *Proceedings of the 1994 International Conference on Artificial Neural Networks*, M. Marinaro and P.G. Morasso (eds.), Springer-Verlag, Vol. 1, London (1994) 549.

[30] Gradshteyn and Ryzhik, *Table of Integrals, Series and Products*, Academic Press, New York (1980).

[31] D.O. Hebb, *The organization of behaviour: a neurophysiological theory*, Wiley, New York (1949).

[32] R. Hecht-Nielsen, *Neurocomputing*, Addison-Wesley, Reading MA (1991).

[33] A.V.M. Herz, Z. Li and J.L. van Hemmen, Statistical mechanics of temporal association in neural networks with transmission delays, Institue for Advanced Study preprint IASSNS-HEP-90/67, Princeton (1990).

[34] J.A. Hertz, A. Krogh and R.G. Palmer, *Introduction to the theory of neural computation*, Addison-Wesley, Redwood City, California (1991).

[35] J.J. Hopfield, Neural networks and physical systems with emergent collective computational abilities, *Proc. Nat. Acad. Sci. USA* **79** (1982) 2554.

[36] J.J. Hopfield, Neurons with graded response have collective computational properties like those of two-state neurons, *Proc. Nat. Acad. Sci. USA* **81** (1984) 3088.

[37] J.J. Hopfield, D.I. Feinstein and R.G. Palmer, 'Unlearning' has a stabilizing effect in collective memories, *Nature* **304** (1983) 158.

[38] J.J. Hopfield and D.W. Tank, 'Neural' computation of decisions in optimization problems, *Biol. Cybern.* **52** (1985) 141.

[39] K. Hornik, M. Stinchcombe and H. White, Multi-layer feedforward networks are universal approximators, *Neural Networks* **2** (1989) 359.

[40] D.H. Hubel and T.N. Wiesel, Receptive fields, binocular interaction and functional architecture in the cat's visual cortex, *J. Physiol.* **160** (1962) 106.

[41] I. Kanter and H. Sompolinsky, Associative recall of memories without errors, *Phys. Rev. A* **35** (1987) 380.

[42] N.B. Karayiannis and A.N. Venetsanopoulos, *Artificial neural networks: learning algorithms, performance evaluation and applications*, Kluwer Academic Publishers, Boston (1993).

[43] T. Kohonen, Self-organized formation of topologically correct feature maps, *Biol. Cybern.* **43** (1982) 59.

[44] W. Krauth and M. Mézard, Learning algorithms with optimal stability in neural networks, *J. Phys. A: Math. Gen.* **20** (1987) L745.

[45] R.P. Lippmann, An introduction to computing with neural nets, *IEEE ASSP Mag.*, april (1987), 4.

[46] W.A. Little, The existence of persistent states in the brain, *Math. Biosci.* **19** (1974) 101.

[47] M. Marchand, M. Golea and P. R.ján, A convergence theorem for sequential learning in two-layer perceptrons, *Europhys. Lett.* **11** (1990) 487.

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

[49] S. Mertens, H.M. Köhler and S. Bos, Learning grey-toned patterns in neural networks, *J. Phys. A: Math. Gen.* **24** (1991) 4941.

[50] M. Mézard and J.P. Nadal, Learning in feedforward layered networks: the tiling algorithm, *J. Phys. A: Math. Gen.* **22** (1989) 2191.

[51] M. Minsky and S. Papert, *Perceptrons*, MIT Press, Cambridge MA, USA (1969).

[52] B. Müller and J. Reinhardt, *Neural networks: an introduction*, Springer-Verlag, Berlin (1991).

[53] T. Nakamura and H. Nishimori, Sequential retrieval of non-random patterns in a neural network, *J. Phys. A: Math. Gen.* **23** (1990) 4627.

[54] J.P. Nadal, Study of a growth algorithm for a feedforward network, *Int. J. of Neural Systems* **1** (1989) 55.

[55] J.P. Nadal and A. Rau, Storage capacity of a Potts-perceptron, *J. Phys. I France* **1** (1991) 1109.

[56] J. Nuñez and J. Llacer, A general bayesian image reconstruction algorithm with entropy prior. Preliminary application to HST data, *Publ. Astronomical Soc. of the Pacific* **105** (1993) 1192.

[57] E. Oja, Artificial neural networks. In *Proceedings of the 1991 International Conference on Artificial Neural Networks*, T. Kohonen, K. Mäkisara, O. Simula and J. Kangas (eds.), North-Holland, Amsterdam (1991) 737.

[58] A. Papoulis, *Probability, random variables and stochastic processes*, McGraw-Hill, New York (1965).

[59] N. Parga and M.A. Virasoro, The ultrametric organization of memories in a neural network, *J. Phys. France* **47** (1986) 1857.

[60] D.B. Parker, Learning logic: casting the cortex of the human brain in silicon, MIT Tech. Rep. TR-47 (1985).

[61] L. Personnaz, I. Guyon and G. Dreyfus, Information storage and retrieval in spin-glass-like neural networks, *J. Phys. France* **46** (1985) L359.

[62] H. Rieger, Properties of neural networks with multi-state neurons. In *Statistical mechanics of neural networks*, L. Garrido (ed.), Lecture notes in Physics, Springer-Verlag **368** (1990).

[63] F. Rosenblatt, *Principles of neurodynamics*, Spartan, New York (1962).

[64] D.W. Ruck, S.K. Rogers, M. Kabrisky, M.E. Oxley and B.W. Suter, The multilayer perceptron as an approximation to a Bayes optimal discriminant function, *IEEE Trans. Neural Networks* **1** (1990) 296.

[65] P. Ruján, Learning in multilayer networks: a geometric computational approach. In *Statistical mechanics of neural networks*, L. Garrido (ed.), Lecture notes in Physics, Springer-Verlag **368** (1990).

[66] P. Ruján, A fast method for calculating the perceptron with maximal stability, preprint (1991).

[67] D.E. Rumelhart, G.E. Hinton and R.J. Williams, Learning representations by back-propagating errors, *Nature* **323** (1986) 533.

[68] D.E. Rumelhart, G.E. Hinton and R.J. Williams, Learning internal representations by error propagation. In *Parallel Distributed Processing*, D.E. Rumelhart and J.L. McClelland (eds.), MIT Press, Vol. 1, Cambridge, MA (1986) 318.

[69] D.E. Rumelhart and D. Zipser. Feature discovery by competitive learning. In *Parallel Distributed Processing*, D.E. Rumelhart and J.L. McClelland (eds.), MIT Press, Vol. 1, Cambridge, MA (1986) 151.

[70] T.D. Sanger, Optimal unsupervised learning in a single-layer linear feedforward neural network, *Neural Networks* **2** (1989) 459.

[71] E. Saund, Dimensionality-reduction using connectionist network, *IEEE Trans. Pattern Analysis and Machine Intelligence* **11** (1989) 304.

[72] W. Schempp, Radar ambiguity functions, the Heisenberg group, and holomorphic theta series, *Proc. of the AMS* **92** (1984) 345.

[73] W. Schempp, Neurocomputer architectures, *Results in Math.* **16** (1989) 103.

[74] W. Schiffmann, M. Joost and R. Werner, Comparison of optimized backpropagation algorithms. In *Prooceedings of the European Symposium on Artificial Neural Networks*, M. Verleysen (ed.), D Facto, Brussels (1993) 97.

[75] S.M. Silva and L.B. Almeida, Speeding up backpropagation. In *Advanced neural computers*, R. Eckmiller (ed.), (1990) 151.

[76] J.M. Sopena and R. Alquezar, Improvement of learning in recurrent networks by substituting the sigmoid activation function. In *Proceedings of the 1994 International Conference on Artificial Neural Networks*, M. Marinaro and P.G. Morasso (eds.), Springer-Verlag, Vol. 1, London (1994) 417.

[77] G. Stimpfl-Abele and L. Garrido, Fast track finding with neural nets, *Comp. Phys. Comm.* **64** (1991) 46.

[78] E.A. Wan, Neural network classification: a Bayesian interpretation, *IEEE Trans. Neural Networks* **1** (1990) 303.

[79] A.S. Weigend, D.E. Rumelhart and B.A. Huberman, Backpropagation, weight elimination and time series prediction. In *Connectionist Models*, R. Touretzky, J. Elman, T.J. Sejnowsky and G. Hinton (eds.), Proceedings of the 1990 Summer School, Morgan Kaufmann.

[80] P. Werbos, *Beyond regression: new tools for prediction and analysis in the behavioral sciences*, Ph.D. thesis, Harvard University (1974).