

AB

EX 10391
SCW 9410

Neural Networks in High Energy Physics: From Pattern Recognition to Exploratory Data Analysis ¹

Vicens Gaitan Alcalde
Universitat Autònoma de Barcelona
Institut de Física d'Altes Energies
E-08193 Bellaterra (Barcelona) Spain

November 1993

CERN LIBRARIES, GENEVA



CM-P00080936

Thesis-1993-Alcalde

¹Thesis Dissertation

Neural Networks in High Energy Physics: From Pattern Recognition to Exploratory Data Analysis ¹

Vicens Gaitan Alcalde
Universitat Autònoma de Barcelona
Institut de Física d'Altes Energies
E-08193 Bellaterra (Barcelona) Spain

November 1993

¹Thesis Dissertation

En Lluís Garrido Bertran, Professor Titular de Física de la Universitat de Barcelona,

CERTIFICA: Que la present memòria, que porta per títol "*Neural Networks in High Energy Physics: From Pattern Recognition to Exploratory Data Analysis*", ha estat realitzada sota la seva direcció per en Vicens Gaitan Alcalde i que constitueix la seva Tesi per a optar al Grau de Doctor en Ciències, Secció Física, per la Universitat Autònoma de Barcelona.

Lluís Garrido Bertran

Bellaterra, 29 d'Octubre de 1993.

Acknowledgements

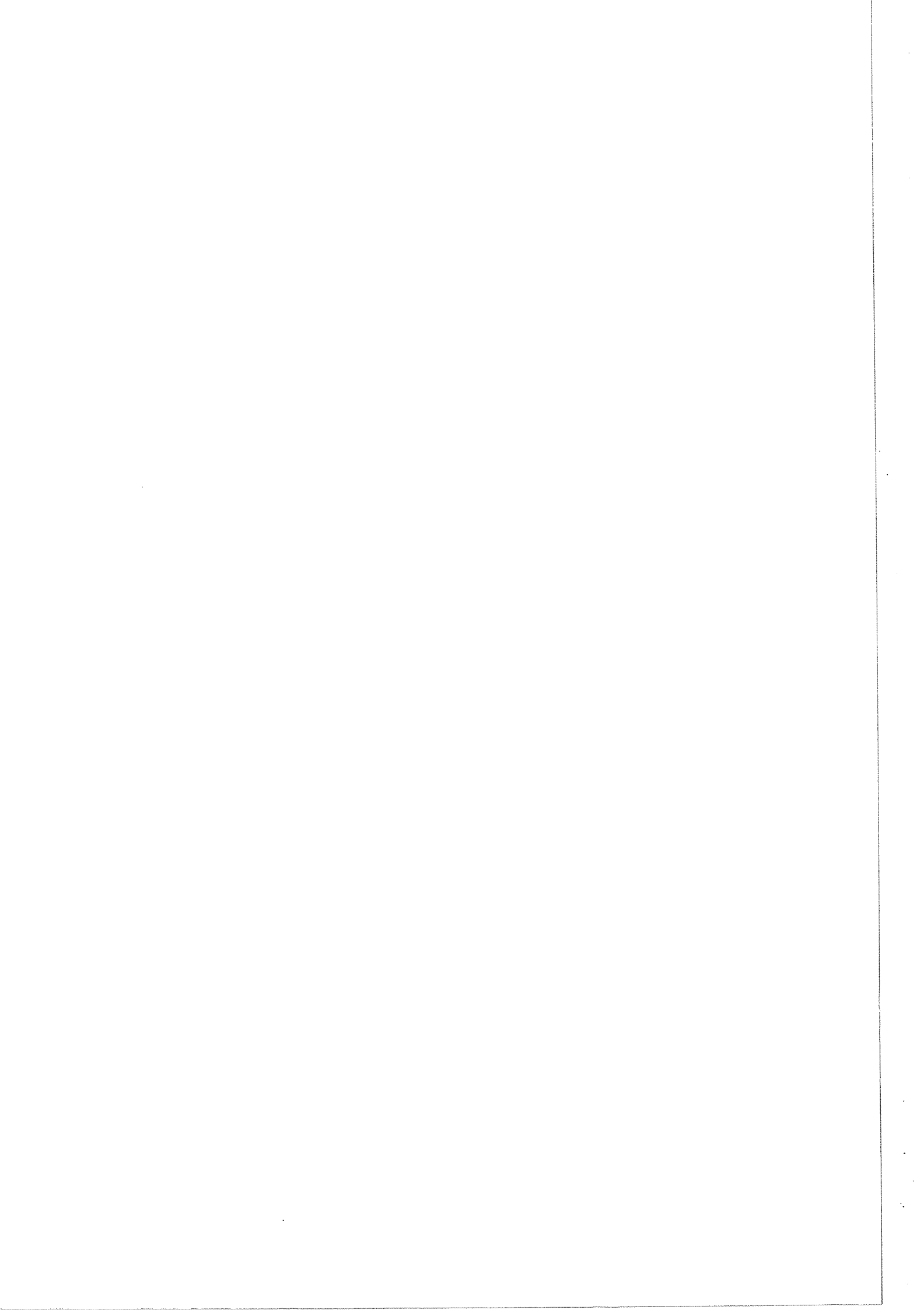
I would like to thank all people that has contributed to this work with their comments, suggestions, discussions, and specially with their patienty.

Abstract

The application of Neural Networks to High Energy Physics is studied in this work.

A set of statistical tools for data analysis based on Neural Networks has been developed, and their optimality properties have been proven. These tools have been used for measuring simultaneously the branching ratios and polarization of τ particles coming from Z^0 decays measured at the ALEPH detector at LEP.

Previous work on track finding using Neural Networks has been extended for on-line purposes using dedicated hardware that makes the method usable for triggering applications.



Resum

En aquest treball han estat estudiades les aplicacions de les Xarxes Neuronals en Física d'Altes Energies.

S'ha desenvolupat un conjunt d'eines estadístiques per a anàlisis de dades basades en Xarxes Neuronals i s'han demostrat les seves propietats d'optimalitat. Amb l'ajut d'aquestes eines s'han mesurat les probabilitats de desintegració de partícules τ en cadascun dels possibles canals així com la seva assimetria de polarització per successos provinents del detector ALEPH al LEP.

Treballs anteriors en reconeixement de traces en detectors fent servir Xarxes Neuronals han estat ampliat en la direcció de fer-los viables per aplicacions en temps reals fent servir un circuit integrat especialment dissenyat per a aquest propòsit.

Contents

1	Introduction	3
1.1	Motivation	3
1.2	Neural Network formalism	6
1.3	History of Neurocomputing	6
2	Multilayer Perceptrons: Supervised learning	9
2.1	Layered Feed Forward Networks	9
2.1.1	BackPropagation (Mean Square approach)	10
2.2	Analytical interpretation of the net output	12
2.2.1	A real example: Tau polarization	14
2.2.2	Finding the helicity of both τ in the process: $e^+e^- \rightarrow Z^0, \gamma \rightarrow \tau^+\tau^- \rightarrow (\rho\nu)(\rho\nu)$	23
2.3	From Network output to results	25
2.3.1	Matrix deconvolution method	26
2.3.2	Fit method	27
2.3.3	Optimal choice of the fitting space	27
2.4	Testing the agreement between multidimensional distributions.	34
2.4.1	Introduction	34
2.4.2	The method	34
2.4.3	Neural Net implementation	37
2.4.4	Examples and results	38
2.4.5	High Energy Physics example	44

3	Unsupervised learning	46
3.1	Introduction	46
3.2	The method	47
3.3	Examples and results	50
4	Hopfield Networks	53
4.1	Neural nets with constraints	53
4.1.1	General considerations	53
4.2	MFT	53
4.3	Breaking NP-hard problems with Hopfield networks.	55
5	Data analysis	59
5.1	Introduction	59
5.2	ALEPH	59
5.3	Tau Physics	60
5.3.1	Introduction	60
5.3.2	A new method to determine the τ polarization for a single channel	62
5.3.3	Simultaneous measurement of the τ polarization and branching ratios with neural networks: Raw input approach	63
5.3.4	Simultaneous measurement of the τ polarization and branching ratios with neural networks: Energy Flow approach	69
5.4	Unsupervised τ classification	74
6	Pattern recognition	77
6.1	Track Finding	77
6.1.1	Introduction	77
6.2	Track finding with Neural Networks	78
6.2.1	Track finding in the ALEPH TPC: Conventional method	79
6.2.2	Definition of the neurons	79
6.2.3	Definition of the weight matrix	81
6.2.4	Update of the neurons	81
6.2.5	Reconstruction of events and results	82
6.3	Discrete track finding	84

6.3.1	Simulation	87
6.4	Hardware implementation	89
6.4.1	New Hardware developements	93
7	Some ANN applications in HEP	97
7.1	Current developements on NN in HEP	97
7.1.1	Triggering and event reconstrucction	97
7.1.2	Off-line analysis	103
8	Conclusions	107
A	Backpropagation algorithm	109
B	Heuristics for designing and training multilayer perceptrons	111
B.1	Input preprocessing	111
B.2	Network topology	112
B.3	Weight initialization	112
B.4	Weight update.	113
C	Mean Field equations for a Hopfield Network.	114
D	Sources of information about NN	116
D.1	Good introductory literature about Neural Networks	116
D.1.1	Books for the beginner:	116
D.1.2	The classics:	118
D.1.3	The best:	118
D.1.4	Introductory journal articles:	118
D.1.5	Not-quite-so-introductory literature:	119
D.2	Any journals and magazines about Neural Networks	120
D.3	The most important conferences concerned with Neural Networks	124
D.4	Other sources of information about NNs	124
D.5	Freely available software packages for NN simulation	126
D.6	Commercial software packages for NN simulation	131
D.7	Neural Network hardware	132

D.8 Databases for experimentation with NNs	138
--	-----

List of Figures

1.1	Schematic drawing of a nervous cell (neuron)	4
1.2	Neurons from the visual cortex of a rat (Ramón y Cajal, 1888) . .	5
1.3	NN paradigms classification: unsupervised vs. supervised and feed-forward vs feedback.	8
2.1	Typical architecture of a layered feed-forward network with two hidden layers (neurons are represented by circles and the connection weights w_{ij} by arrows pointing in the direction of the information flow).	10
2.2	Angular distribution for a τ^- with positive helicity (or τ^+ with negative helicity). The area of the boxes is proportional to the weight distribution in the bin.	16
2.3	Angular distribution for a τ^- with negative helicity. (or τ^+ with positive helicity). The area of the boxes is proportional to the weight distribution in the bin.	17
2.4	Activation of the output unit after convergence for the learning example $\tau^- \rightarrow \rho^- \nu$	18
2.5	Efficiency in the classification found by the neural net as a function of the cut in the output unit.	19
2.6	Boundary between the two zones corresponding to different helicities found by the neural net (dashed line) and the Bayesian method (solid line).	20
2.7	Net output predicted by equation (5) ($o(x)$) and learned by a 2-3-1 net ($n(x)$) vs. $\cos \varphi$ and $\cos \psi$ for three different α in the training sample.	22
2.8	State of the net after convergence. The widths of the lines are proportional to $ w_{ij} $	23

- 2.9 Top: Bidimensional probability distribution functions for three classes. Center: Function learned by a 2-X-2 neural net trained to distinguish the three classes. The third output neuron (O_3) is redundant. Bottom: bidimensional neuron output distributions for each class. 31
- 2.10 Top: One-dimensional reference distribution for each class corresponding to output neuron one. Center: One-dimensional reference distribution for each class corresponding to output neuron two. Bottom left: two dimensional output distribution for a mixture sample with proportions 0.2,0,3,0.5. The other two figures at the bottom show the projections of this distribution over the two output neurons (solid line e), and its composition in terms of the three single classes (Dashed line: class 1, Dotted line: class 2, Dotted-Dashed line: class 3) obtained by making a fit of these two projections to a linear combination of the reference unidimensional distributions. The proportions obtained are in good agreement with the original ones. 33
- 2.11 a) Bidimensional projection (x_1 vs x_2) for sample 1. b) Bidimensional projection (x_1 vs x_2) for sample 2. c) $\langle \chi^2 \rangle$ as function of the number of events "n" between the output of the net for sample 1 and sample 2. d) Correlation between the output neuron and each input variables for sample 1 versus the same quantity for sample 2. The three points out of the diagonal corresponds to the variables with correlation at the original space. 39
- 2.12 Plot of $\langle \chi^2 \rangle$ vs the number of events n of each sample, using the known probabilities distributions a) at the input space. The slope of the fitted line gives $I = 0.049$. b) at the κ space. The slope of the fitted line gives $I = 0.051$ 40
- 2.13 Distribution of events in the projected κ space (full line corresponds to sample 1 and dashed line to sample 2) a) Calculated from the analytical expressions ($\kappa = P_1/(P_1 + P_2)$). b) Estimated from the output of the neural network. 41
- 2.14 View of the 3-dimensional generated samples (The two triangles plotted corresponds to the intersection between the cube and the planes $x + y + z = 1$ and $x + y + z = 2$ respectively). a) view of sample 1. b) view of sample 2. 42
- 2.15 a) Output neuron distribution (full line corresponds to sample 1 and dashed line to sample 2). b) Correlation between the output neuron and each input variables for sample 1 versus the same quantity for sample 2. c) Points in the cube which neuron output is in the interval: [0.49, 0.495] d) Points in the cube which neuron output is in the interval: [0.505, 0.51]. 43

3.1	Network topology for the encoder task	48
3.2	Top: we show the learned mapping f_2 (a one-dimensional surface, the continuous line), superposed to the input two-dimensional points, yielded by a 2:5:1:5:2 neural network using a sigmoid function for the neuron response, after the MSBP was applied. Bottom: the activation of the neck unit versus the angle θ is plotted.	51
3.3	Results yielded by the PCA and the NNA (bottom) when we attempt to separate 4 Gaussian distribution contained in the two-dimensional input space (top), when a projection onto one-dimension is made. .	52
4.1	Network topology for the Hopfield model.	54
4.2	Two possibilities on the problem complexity classification: In the first case if a NP problem is solved in polynomial time, then there exists a polynomial algorithm for any problem in the NP class. Although this cannot be excluded, the most likely situation is shown in bottom.	57
5.1	Distribution for the two input variables for data and Monte Carlo and the resulting neuron output for the τ decaying into ρ polarization analysis from the 90 ALEPH data.	64
5.2	NN topologies for the polarization analysis.	65
5.3	Variables used in the polarization analysis.	66
5.4	Efficiency for distinguish real and Monte Carlo data using the NNAC analysis output neuron to classify them.	68
5.5	Efficiency matrix.	71
5.6	Event distribution for the first three output neurons (a,b,c) and the polarization neuron (d) for data and the best Monte Carlo fit. The dashed line corresponds to Monte Carlo, errors bars to real data . .	72
5.7	Top: Correlation matrix. Black pixels shows bin to bin correlations different from 0. Next to the X-Y axes the correlation values are shown. Top right and bottom left :correlation values of a bin with all the others	73
5.8	Unsupervised tau decay clusterization.	75
5.9	Unsupervised tau decay clusterization.	76
6.1	Flow chart for the standard ALEPH track finding algorithm.	80
6.2	Display of all generated lines for a real $Z^0 \rightarrow$ hadrons (X-Y projection). .	83
6.3	Display of all generated lines for a real $Z^0 \rightarrow$ hadrons (R-Z projection). .	84

6.4	Display of the activated lines after convergence for a real $Z^0 \rightarrow$ hadrons (X-Y projection).	85
6.5	Display of the activated lines after convergence for a real $Z^0 \rightarrow$ hadrons (R-Z projection).	86
6.6	Execution time as a function of the number of tracks.	87
6.7	Display of the generated lines for a Monte Carlo event with 100 tracks (X-Y projection).	88
6.8	Display of the activated lines after convergence for a Monte Carlo event with 100 tracks (X-Y projection).	89
6.9	Schematic view of the detector segmentation. Solid lines: excitatory connections; dashed lines: inhibitory connections. The dotted area corresponds to the pads with hits caused by the track.	90
6.10	Display of all possible neurons using fixed architecture (X-Y projection).	91
6.11	Display of a reconstructed event with 50 tracks using fixed architecture (X-Y projection).	92
6.12	Weights obtained after clipping. Dashed (dotted) line corresponds to a +1 (-1) weight in the connection with the solid neuron.	93
6.13	Basic schematic of the neuron cluster.	95
6.14	Physical implementation of synaptic connections.	96

List of Tables

2.1	Input variables. The charged track with the highest momentum corresponds to the particle p_1	45
5.1	Input variables for the polarization analysis. The charged track with greater momentum corresponds to the particle p_1	67
5.2	Input variables for the polarization analysis using energy flow. . . .	70
5.3	Branching ratios and Polarization results for the 1992 data.	71
8.1	Branching ratios and Polarization results for the 1992 data.	108

Motivation of the present work

In 1988, a paper about a method for track finding [1] introduced some words unknown to the people working in the High Energy Physics field : Neural Networks (NNs).

At that time people were interested in using expert systems for automatic data classification [2]. The aim was to build a system able to "learn" from Monte Carlo (MC) a set of rules (cuts) in order to separate two kind of events. The approach was correct : a human expert makes this task in the same fashion. But the method does not work properly. A system based on logical (or even probabilistic) reasoning needs a lot of a priori knowledge in order to learn from examples (as human have to do).

That summer, my thesis advisor applied NNs for reconstruction of real tracks from the ALEPH detector at LEP [3]. And as a logical step forward on pattern recognition he was also trying to use NNs for classification. The problem: Identify the helicity of a τ particle decaying into a ρ . This is a theoretically well known problem, and the optimal solution cannot be found using a single linear cut. He used a Multilayer Perceptron (MLP) trained with Backpropagation [4]. The NN was able to learn the right answer. This was impressive for me. At that time I decided to understand the "mystery" of NN.

Now, several people are using similar technics in HEP, from DAQ to off-line data analysis.

Chapter 1

Introduction

1.1 Motivation

The human brain has capabilities that outperform the most advanced Artificial Intelligence (AI) systems, even running on the fastest supercomputer: Visual and speech information processing, adaptative motor task control and so on. The brain has, also, many other features that can be desirable from a computational point of view:

- It is robust and fault tolerant.
- It is flexible. It can easily adjusted to a new environment by *learning*.
- It can deal with information that is fuzzy, probabilistic, noisy or inconsistent.
- It is highly parallel.

Nevertheless in tasks based principally on numerical computation or logical inference, a digital sequential computer can easily outperform a human in speed and accuracy. This apparent paradox motivates the search for an alternative computational paradigm to the usual one (based on a programmed instruction sequence), which was introduced by von Neumann and has been used as the basis of almost all machine computation to date.

Neurosciences provide an inspiration of new models of computation based primarily on the collective behaviour of adaptative single cells (neurons). The brain is composed of about 10^{11} neurons (nerve cells) of many different types. Fig 1.1 is a schematic drawing of a single neuron. Tree-like networks of nerve fiber called dendrites are connected to the cell body or soma, where the cell nucleus is located. Extending from the cell body is a single long fiber called the axon, which eventually branches into strands and substrands. At the ends of these are the transmitting

ends of the synaptic junctions, or synapses, to other neurons. (Fig 1.2). The receiving ends of these junctions on other cells can be found both on the dendrites and on the cell bodies themselves. The axon of a typical neuron makes a few thousand synapses with other neurons.

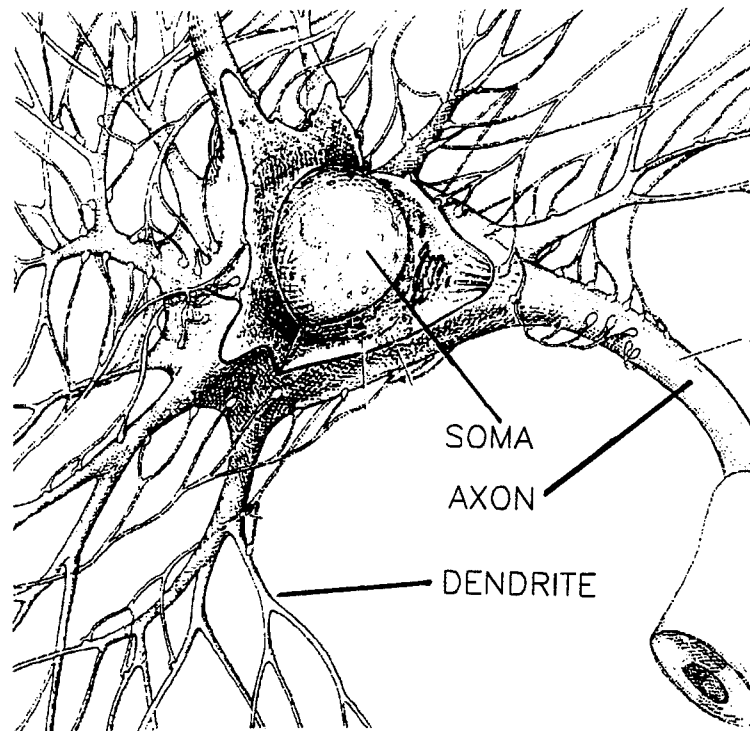


Figure 1.1: Schematic drawing of a nervous cell (neuron)

The transmission of a signal from one cell to another at a synapse is a complex chemical process in which specific transmitter substances are released from the sending side of the junction. The effect is to raise or lower the electrical potential inside the body of the receiving cell. If this potential reaches a threshold, a pulse of action potential of fixed strength and duration is sent down the axon. The most powerful characteristic of this system is that the synapse strength is adaptatively varied in response to the behavior of the neighbor cells. This will be identified with the process of learning. The memory of a biological brain is stored in these synapse strengths.

The collective behavior of networks of single cells like the previously described leads to complex sequences of information transmission and processing capable of performing sophisticated computational tasks. Just as most of the details of the

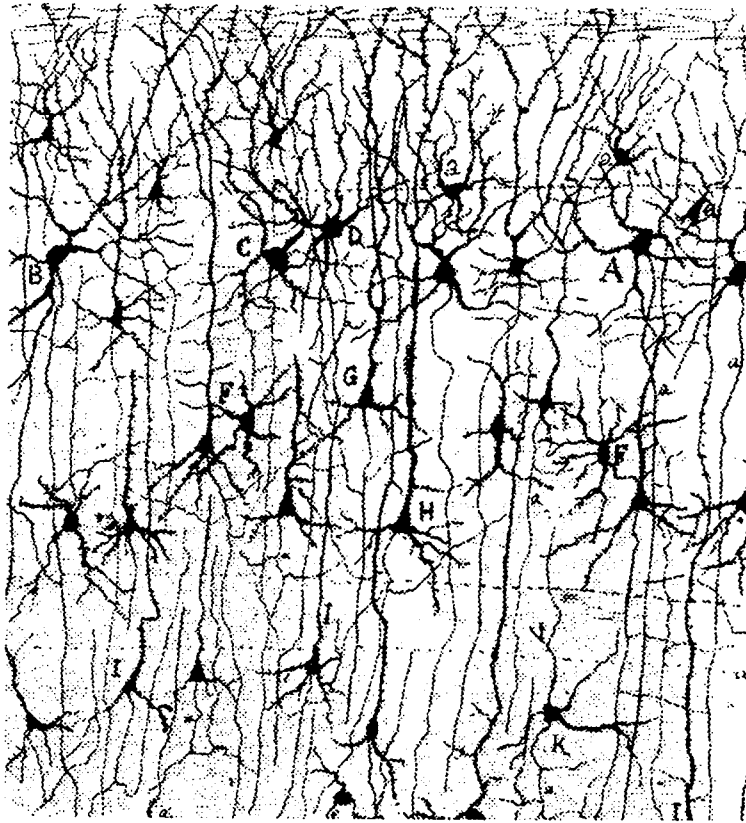


Figure 1.2: Neurons from the visual cortex of a rat (Ramón y Cajal, 1888)

separate parts of a large computer are unimportant in understanding the behavior of the whole machine, so many details of single nervous cells may be unimportant in understanding the collective behavior of networks of cells. This fact suggests to work in a more abstract layer in order to emulate the capabilities of the human brain. The model will be described in the next section.

1.2 Neural Network formalism

To systematize all possible implementations of neural networks on software or hardware it is clear that we need a formal definition of a neural network (NN).

A formal and very general definition of a NN is (from [5]):

- A NN is a parallel, distributed information processing structure consisting of single processing elements (called neurons) interconnected by input/output channels (synapses). The information processing that goes on within each neuron can be defined arbitrarily with the restriction that it must be completely local; i.e. it must depend only on the current values of the input signals coming from the adjacent synapses.

In order to define a NN one must specify several parameters:

- Activation function for the neuron: which kind of mathematical operation must be performed in order to obtain the neuron output from the input signals.
- Topology: which is the pattern of connections between neurons, the information flow, and which neurons must be considered as input or output processing elements.
- Learning algorithm: how to adapt the connection strength in response to the environment.

1.3 History of Neurocomputing

The beginning of neurocomputing is often taken to be the 1943 paper of McCulloch and Pitts [6]. This paper shows that even simple types of neural networks, could in principle compute some arithmetic or logical function. In 1949, Donald Hebb [7], described the first learning rule, based on classical psychological conditioning: the synapsis between two neurons is reinforced when both are activated at the same time, and inhibited otherwise.

The first successful neurocomputer (the Mark I Perceptron) [8] was developed by Rosenblatt, Wightman and others in 1957. It was capable to perform visual pattern recognition with its 400 pixels retina with hard limiting units, using a Hebbian learning rule. Shortly afterwards, Widrow developed a different type of processing element called ADALINE (ADaptative LINEar Element)[9], which was equipped with a powerful learning law, still in widespread use.

By the mid 1960's the neurocomputing field was drawing to a close, mainly due to the non-rigorous approach to the subject and the lack of the promised results.

The final episode of this era was the book *Perceptrons* [10] by Minsky and Papert, written with the aim to discredit neural network research. The implicit thesis of *Perceptrons* was that essentially, all neural networks suffer from the same drawback than perceptrons: the inability to compute non-linearly separable predicates like the XOR (In fact this is only true for single layer perceptrons). This stopped the interest in the field.

In the 1970's Grossberg developed his Adaptive Resonance Theory (ART) [11], introducing novel hypotheses about the underlying principles governing biological neural systems. Another important theory on self organizing systems was pioneered by Kohonen with his work on feature maps [12].

In the 1980's two facts revitalize the field: the established physicist John Hopfield [13] introduced outer products learning laws as well as an equivalent approach to the hebbian learning for fully- connected networks with feedback, using the tools coming from the Statistical Mechanics field. The other point was the publication of the Backpropagation algorithm by Rumelhart [4], which broke the single layer perceptron limitations by providing a learning rule for training nonlinear multilayer perceptrons. The Backpropagation algorithm was invented in fact by Werbos [14] in 1971, but unfortunately, his work remained almost unknown in the scientific community.

Today, the field is in a state of fast growth, focusing the interest in both theoretical developments and practical applications. Figure 1.3 shows a classification of neural paradigms developed at this moment. The main division is related to the learning method. In unsupervised learning there is no teacher and the network self-organizes the information processing. In supervised learning instead, input-output pairs are supplied as examples to the network. The other way to classify the neural network models takes into account the information flow through the network: with feedback connections or without them (feedforward).

In the next chapters the two most used types of networks will be revised in detail: Multilayer Perceptrons and Hopfield Networks. The reason for using the Multilayer Perceptron is related to the existence of a very general learning rule (Backpropagation) which makes possible real medium-scale problem solving. In the case of Hopfield Networks, the reason is that there exist a well understood physical model (spin glasses) which provides several tools for studying these networks. Nevertheless, either of these neuron models matches a biological plausible model for the brain. In this case, the local learning law, and the non-stationary dynamics (due to the asymmetry in the synapses) makes it difficult to design affordable networks for practical purposes. In this work the emphasis is on applications, and not in plausible models of the human brain.

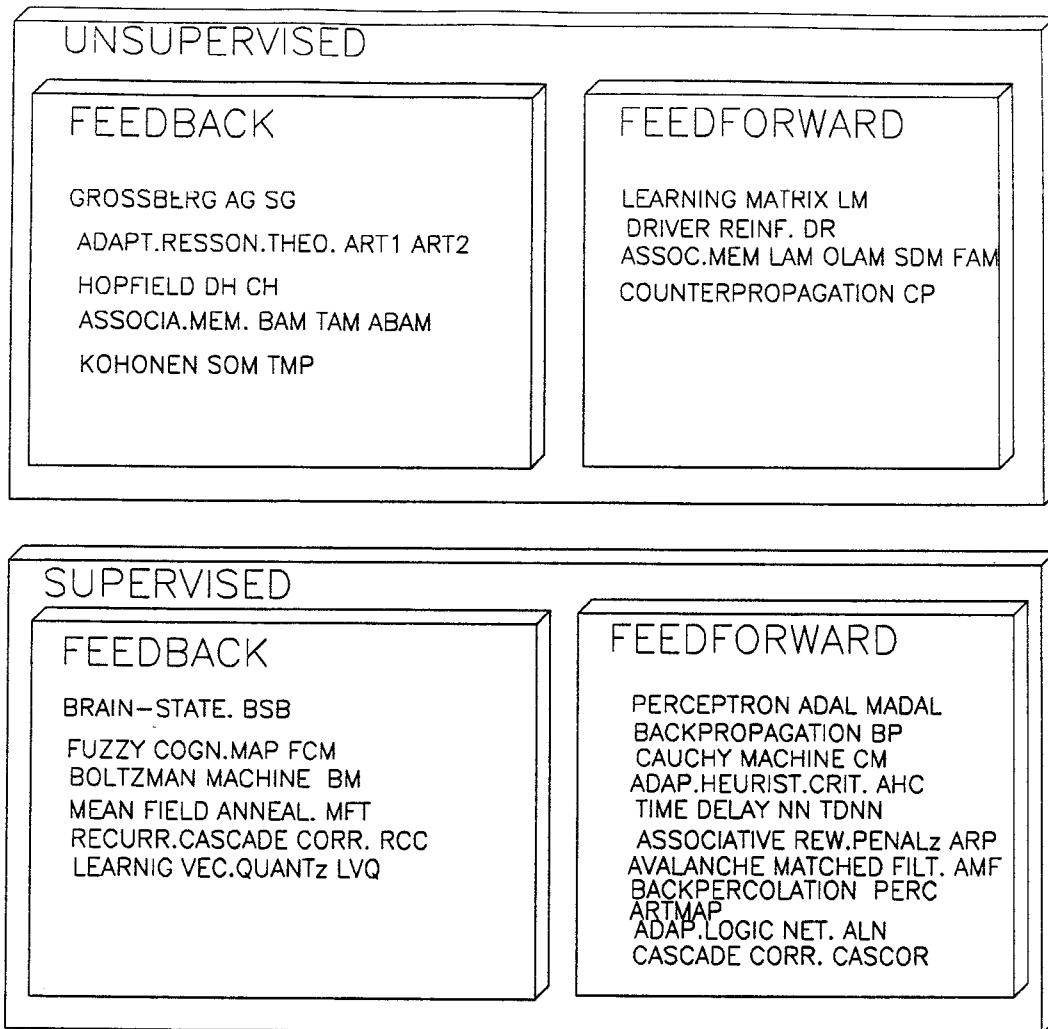


Figure 1.3: NN paradigms classification: unsupervised vs. supervised and feedforward vs feedback.

Chapter 2

Multilayer Perceptrons: Supervised learning

2.1 Layered Feed Forward Networks

A typical architecture for a layered feed-forward network is shown in Figure 2.1. The neurons are grouped into layers. There is one input layer where the information comes in, one or several hidden layers where the information is processed, and one output layer which gives the answer of the net.

The input to neuron i in layer $l+1$ is given by:

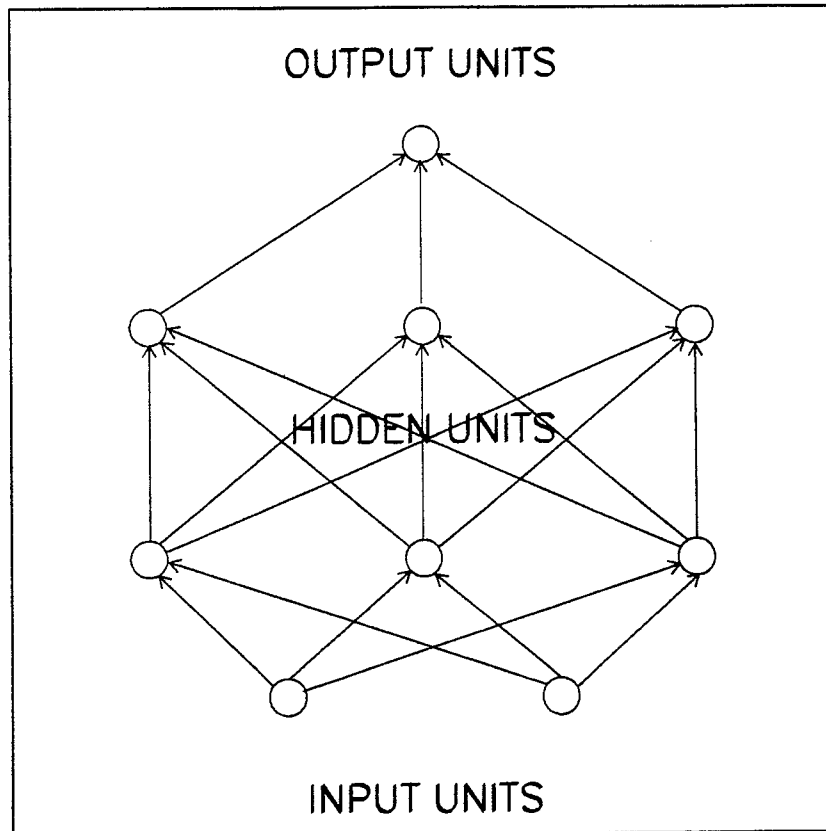
$$I_i^{l+1} = \sum_j w_{ij}^{l+1} S_j^l + B_i^{l+1}, \quad (2.1)$$

where the sum runs over the neurons of the preceding layer (l), S_j^l is the state of the neuron j at layer l , w_{ij}^{l+1} is the connection weight between the neuron i and the neuron j , and B_i^{l+1} is a bias input to neuron i .

The state of the neuron is a function of its input, $S_j^l = f(I_j^l)$. In this work we will use as a function $f(I)$ the 'logistic function': $f(I) = 1/(1 + \exp(-I/T))$, where T is a parameter which plays the role of the temperature. The choice of this function constrains the state of the neuron to a value between 0 and 1.

The activity of the neurons in the input layer is propagated forward to the output layer by equation (2.1). By interpreting the activity of the output units as a measure of the probability that the input pattern belongs to a certain category, the network can be used for classification, as we will explain in the next section. In general the network must be trained. This is done by applying learning algorithms [4] to the net to modify the weights and the bias term in such a way that the difference between the actual and the expected output is minimized in the training sample.

Figure 2.1: Typical architecture of a layered feed-forward network with two hidden layers (neurons are represented by circles and the connection weights w_{ij} by arrows pointing in the direction of the information flow).



2.1.1 BackPropagation (Mean Square approach)

A very popular learning algorithm is the so called error back-propagation. Back-propagation (BP) is a supervised learning algorithm where the rule to be learned is defined in advance by a set of input desired-output pairs. The main objective of

back propagation is to minimize an error function (also called energy)

$$E = E(I_i^{(p)}, d_j^{(p)}, w_{kl}) = \frac{1}{2} \sum_{jp} (o_j^{(p)} - d_j^{(p)})^2 \quad (2.2)$$

by adjusting the w_{kl} parameters. $I_i^{(p)}$ is the state of neuron i in the input layer, $o_j^{(p)}$ the state of neuron j in the output layer, $d_j^{(p)}$ its desired state, and p runs over the patterns of the training sample.

The error calculated in the last layer is propagated backwards using an ordered chain rule to obtain $\frac{\partial E}{\partial w_{ij}}$ (Appendix A). Steepest descent in the opposite direction of this error gradient takes the function to a minimum if we use

$$\Delta w_{ij}(n) = -\epsilon \frac{\partial E}{\partial w_{ij}} + \alpha \Delta w_{ij}(n-1) \quad (2.3)$$

The parameter ϵ is the learning rate. Choosing $\epsilon \approx 1$, the error goes to a minimum faster but losing precision on the position of the global minimum. The parameter α is the momentum rate. It speeds up the convergence of the algorithm by increasing the step in a direction averaged over the sample. A good choice is to update the weights after the presentation of each couple input-output using a learning rate ϵ of the order of 0.1 and a momentum rate of $\alpha = 0.9$. Normally, to reach the convergence in the learning process, several passes over all the training sample (training iterations) are needed.

It is known that the BP algorithm has two major problems [15]. The first one is that the energy surface may have local minima, therefore finding the optimal solution is not guaranteed; the second one is slowness of the convergence process. To minimize these problems we applied the Multi-Seed Backpropagation method (MSBP) [16]. It consists of using a wide range of random number generator seeds to find the initial weights of the network, and calculating, for every seed, the final minimum energy with classical BP. After the MSBP was applied, an energy versus seed sample was obtained in order to easily distinguish local minima points from global minima one.

More sophisticated techniques can be used to minimize E , like conjugate gradient descent [17], or quasi-Newtonian methods [18]. With these methods the number of iterations over the training sample is lower, but due to the complexity of the calculations involved (linear search and pseudo-inverse calculation respectively) the total CPU time needed to reach the minimum is most likely to be of the same order of magnitude.

Instead, a method based on local adaptation of the learning rate produces a big improvement in the learning speed and the stability of the solution when new examples are used. In Appendix B the heuristics implemented in the BP algorithm are explained.

2.2 Analytical interpretation of the net output

At this moment, there is no full theory of multilayer perceptrons. Several questions, like the optimal network topology for a given problem, or what are the general properties of functions mapped with a MLP (MultiLayer Perceptron), are still open. Nevertheless, at least for classifications task, the network output can be related to the well known theory of optimal Bayesian classifiers. This study was motivated by the fact that when a MLP is trained to distinguish between two classes using two output neurons, (demanding $0 - 1$ for one class and $1 - 0$ for the other) their sum gives 1 with great accuracy. The first step was to understand why the network, or the objective function, impose this fancy constraint. The results of this work is explained in the following.

Let us assume that we have two classes of patterns defined by two probability distributions $P_1(\vec{x})$ and $P_2(\vec{x})$, both normalized to 1 ($\int P_1(\vec{x})d\vec{x} = \int P_2(\vec{x})d\vec{x} = 1$). We want to distinguish the two classes using a feed-forward layered neural net trained with back propagation over a sample with an α_i proportion of class i . The input layer will have as many neurons as the dimension of vector $\vec{x}$ and their activation will be $\vec{x}$. The output layer will consist of only one neuron with the desired output of 1 for the first class and 0 for the second one.

The error function defined by equation (2.2) can be written as a functional of $o(\vec{x})$ in terms of the probability distributions of the different categories in the training sample

$$E[o(\vec{x})] = \int d\vec{x} (\alpha_1 P_1(\vec{x})(o(\vec{x}) - 1)^2 + \alpha_2 P_2(\vec{x})(o(\vec{x}))^2) \quad (2.4)$$

where $o(\vec{x})$ is the net output and obviously $\alpha_1 + \alpha_2 = 1$. The functional minimum of $E[o(\vec{x})]$ is reached when

$$\frac{\delta E[o(\vec{x})]}{\delta o(\vec{x}')} = 2 \int d\vec{x} (\alpha_1 P_1(\vec{x})(o(\vec{x}) - 1) + \alpha_2 P_2(\vec{x})o(\vec{x})) \delta(\vec{x} - \vec{x}') = 0 \quad (2.5)$$

which implies

$$o(\vec{x}) = \frac{\alpha_1 P_1(\vec{x})}{\alpha_1 P_1(\vec{x}) + \alpha_2 P_2(\vec{x})} \quad (2.6)$$

Notice that this equation gives the probability for a given $\vec{x}$ to belong to class 1 (Bayes Theorem). Assuming that we want to distinguish the two classes applying a cut on the value of $o(\vec{x})$ this will be just a Bayesian Decision Rule [19] (for example this cut will be 0.5 if the test sample has the same α_i as the training sample).

In the case that the test sample has a different proportion α'_i from the training sample, a Bayesian Decision Rule is still possible by applying a cut β on $o(\vec{x})$, where

β is given by the equation

$$\frac{1 - \beta}{\beta} = \frac{\alpha_2 \alpha_1'}{\alpha_1 \alpha_2'} \quad (2.7)$$

This last equation has been obtained from (2.6) and the condition valid in the cut region $\alpha_1' P_1 = \alpha_2' P_2$. In the case that $\alpha_i = \alpha_i'$ we recover the cut mentioned above of $\beta = 0.5$. In the case that $\alpha_i' = 0.5$ the cut must be $\beta = \alpha_1$.

The learning procedure maps the input vector $\vec{x}$ to the output unit by a function $n(\vec{x})$. The network learns by adjusting its weights in order to approximate $n(\vec{x})$ to $o(\vec{x})$. The functional form of $n(\vec{x})$ and its similarity to $o(\vec{x})$ depends only on the net architecture, its degrees of freedom (w_{ij}) and the existence of adequate data. This shows that this type of neural net methods is an approximation to a Bayesian Decision Rule, but with the important fact that these approximations can be built with a minimum of a priori information based on the generalization power of the neural nets [47]. This means that it is not necessary to know the full information on the $P_i(\vec{x})$.

The generalization for n classes is straightforward. We have studied two cases: unary and binary representation of the class number in the output layer. In the unary case the number of the output neurons is n , and class i is represented with a 1 in the i output neuron and 0 in the others. The best net output $o_i(\vec{x})$ of neuron i is given by:

$$o_i(\vec{x}) = \frac{\alpha_i P_i(\vec{x})}{\sum_{j=1}^n \alpha_j P_j(\vec{x})} \quad (2.8)$$

In the first case the $o_i(\vec{x})$ can be used for Bayesian Decision Rule. For instance when $\alpha_i = \alpha_i', \forall i$ the decision rule consists in choosing the class i for which $o_i(\vec{x})$ is the maximum.

In the binary case the number of the output neurons is $\log_2 n$. Class i is encoded mapping the binary representation of i to the output layer. The best net output $o_i(\vec{x})$ of neuron i is given by:

$$o_i(\vec{x}) = \frac{\sum_{(i)} \alpha_i P_i(\vec{x})}{\sum_{j=1}^n \alpha_j P_j(\vec{x})}, \quad (2.9)$$

where (i) denotes the sum over the classes whose binary number class has a 1 in position i . Again the learning procedure will try to adjust the weights in order to approximate $n_i(\vec{x})$ to $o_i(\vec{x}), \forall i$. In this case there is a loss of information that does not allow to build a Bayesian decision rule.

An interesting application is the estimation of the underlying probability distribution function (p.d.f.) of a sample of data. If the network is trained to classify as 1 the events belonging to the sample from which the unknown p.d.f ($p(\vec{x})$) will be estimated, and to classify as 0 for a sample with known p.d.f $p_0(\vec{x})$, the output will

be (assuming equal proportions for both samples)

$$o(\vec{x}) = \frac{p}{p + p_0}$$

thus

$$p(\vec{x}) = p_0 \frac{o}{1 - o}$$

The accuracy of the approximation depends on the similarity between p_0 and p because the network has to learn a smoother function.

A final remark is that in this development the cost function used is a Kroneker's delta (penalty 1 if the classification is correct, 0 otherwise). Nevertheless once the a posteriori probabilities are know (the network output) it is possible to build the decision boundary corresponding to any cost function [21].

In the next subsection this concepts will be demonstrated by applying a Feed Forward Neural Network (FFNN) to the problem of identifying the helicity of a τ decaying into $\rho\nu$.

2.2.1 A real example: Tau polarization

In Particle Physics, one assigns an observable called helicity to a given system. This quantity is defined as the projection of the spin over the motion direction. The study of the helicity decay-products in e^+e^- scattering was considered an important point in precision tests of the Standard Electroweak Model [22]. In this section we discuss the determination of the helicity of a τ^- that decays into $\rho^- + \nu$ assuming that the τ^- is in one of the two possible helicity states $h = \pm 1$. As the ρ^- decays almost 100% of the time in $\pi^- + \pi^0$ the signature for such a τ^- will be a charged track (π^-), neutral energy (π^0) and missing energy (ν). We define φ as the τ^- decay angle: the angle between the ρ^- and the τ laboratory line of flight with the τ^- at rest. Similarly we define ψ as the ρ^- decay angle.

The angular distribution as function of φ and ψ of this process is for $h = 1$ [23]:

$$\begin{aligned} \frac{1}{C} \frac{\partial^2 W}{\partial \cos \varphi \partial \cos \psi} &= \frac{1}{C} \left[\cos^2 \psi \left(\cos \eta \cos \frac{\varphi}{2} + \frac{m_\rho}{m_\tau} \sin \eta \sin \frac{\varphi}{2} \right)^2 + \right. \\ &\left. \frac{\sin^2 \psi}{2} \left[\left(\sin \eta \cos \frac{\varphi}{2} - \frac{m_\rho}{m_\tau} \cos \eta \sin \frac{\varphi}{2} \right)^2 + \frac{m_\rho^2}{m_\tau^2} \sin^2 \frac{\varphi}{2} \right] \right] \equiv P_+(\vec{x}) \end{aligned} \quad (2.10)$$

and for $h = -1$

$$\begin{aligned} \frac{1}{C} \frac{\partial^2 W}{\partial \cos \varphi \partial \cos \psi} &= \frac{1}{C} \left[\cos^2 \psi \left(\cos \eta \sin \frac{\varphi}{2} - \frac{m_\rho}{m_\tau} \sin \eta \cos \frac{\varphi}{2} \right)^2 + \right. \\ &\left. \frac{\sin^2 \psi}{2} \left[\left(\sin \eta \sin \frac{\varphi}{2} - \frac{m_\rho}{m_\tau} \cos \eta \cos \frac{\varphi}{2} \right)^2 + \frac{m_\rho^2}{m_\tau^2} \cos^2 \frac{\varphi}{2} \right] \right] \equiv P_-(\vec{x}) \end{aligned} \quad (2.11)$$

m_ρ is the ρ mass, m_τ is the τ mass, $\vec{x} = (\cos \varphi, \cos \psi)$, C is a normalization constant, and $\cos \eta$ is given by (assuming that the τ energy is much bigger than m_τ)

$$\cos \eta = \frac{m_\tau^2 - m_\rho^2 + (m_\tau^2 + m_\rho^2) \cos \varphi}{m_\tau^2 + m_\rho^2 + (m_\tau^2 - m_\rho^2) \cos \varphi} \quad (2.12)$$

The formulae corresponding to τ^+ are the same as those of τ^- with the exchange of h by $-h$.

The distribution for a τ^- with positive helicity is plotted in figure 2.2 and with negative helicity in figure 2.3.

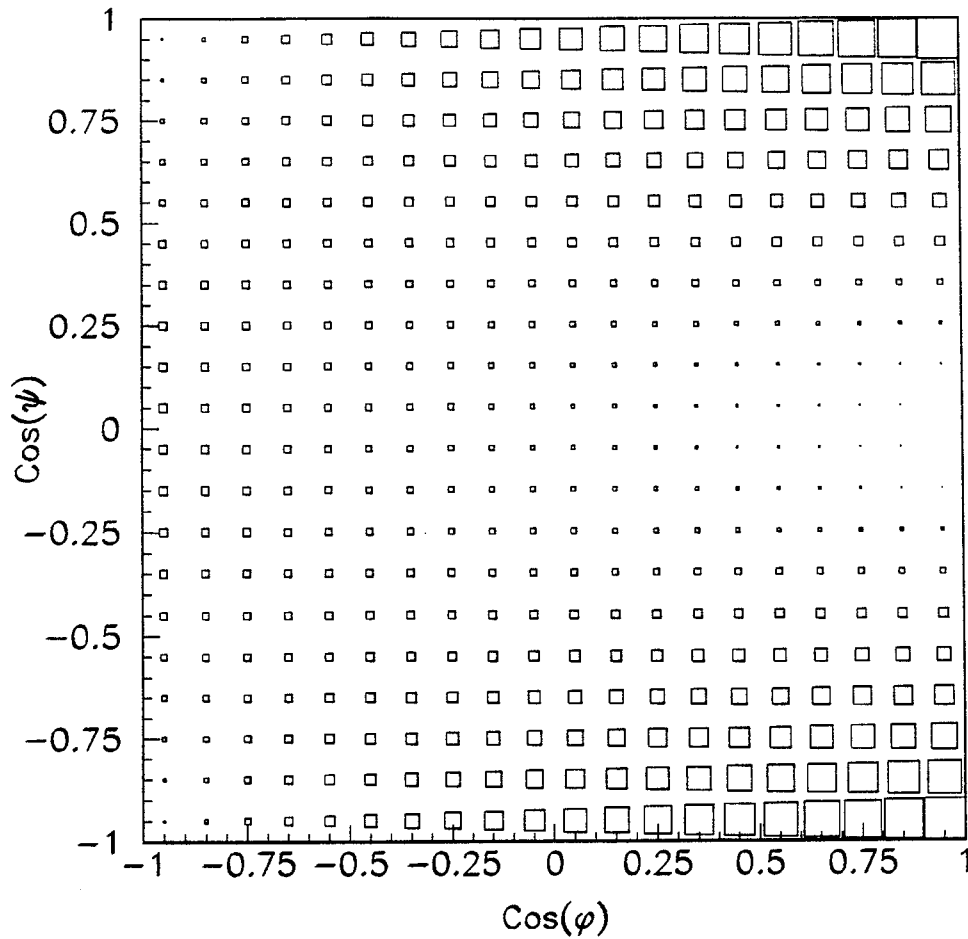


Figure 2.2: Angular distribution for a τ^- with positive helicity (or τ^+ with negative helicity). The area of the boxes is proportional to the weight distribution in the bin.

From these figures it is obvious that we have to exclude the regions where $\cos \varphi \approx 1$ and simultaneously $\cos \psi \approx \pm 1$ if we want to select τ^- with negative helicity.

To test now whether a simple feed-forward network can distinguish between the two helicity regions we have chosen a neural net with 2 units in the input layer, one hidden layer with 4 units, and 1 output unit. We have generated τ^- decays according to equations (2.10) and (2.11) with equal probability for each helicity

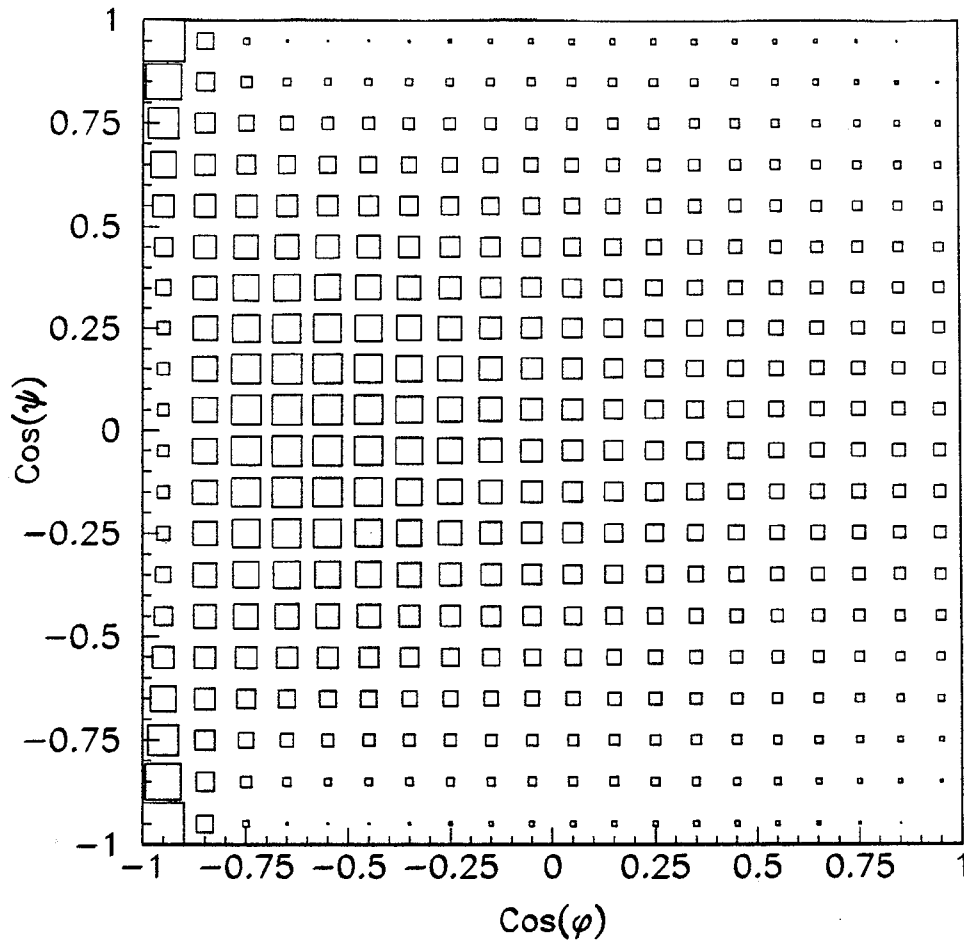


Figure 2.3: Angular distribution for a τ^- with negative helicity. (or τ^+ with positive helicity). The area of the boxes is proportional to the weight distribution in the bin.

(unpolarized sample: $\alpha_+ = \alpha_- = 0.5$). The 2 units of the input layer are activated by the quantities $(\cos \varphi + 1)/2$ and $|\cos \psi|$ for a given τ^- decay. Then we look at the activation of the output unit, and we want that this activation is 1 for a τ^- with positive helicity and 0 for a τ^- with negative helicity. When the activation of the output unit is not the desired one we modify the weights and the bias term by error back-propagation. τ^- decays are presented to the net until the weights are

stabilized. We have found that it is better to generate each time a new τ decay than to have a sample with a finite number of decays and go over and over the same sample during the learning procedure. This comes from the fact that if the sample does not contain enough events, statistical fluctuations produce a bias in the output of the net.

Figure 2.4 shows the activation of the output unit after convergence as a function of $\cos \varphi$ and $\cos \psi$.

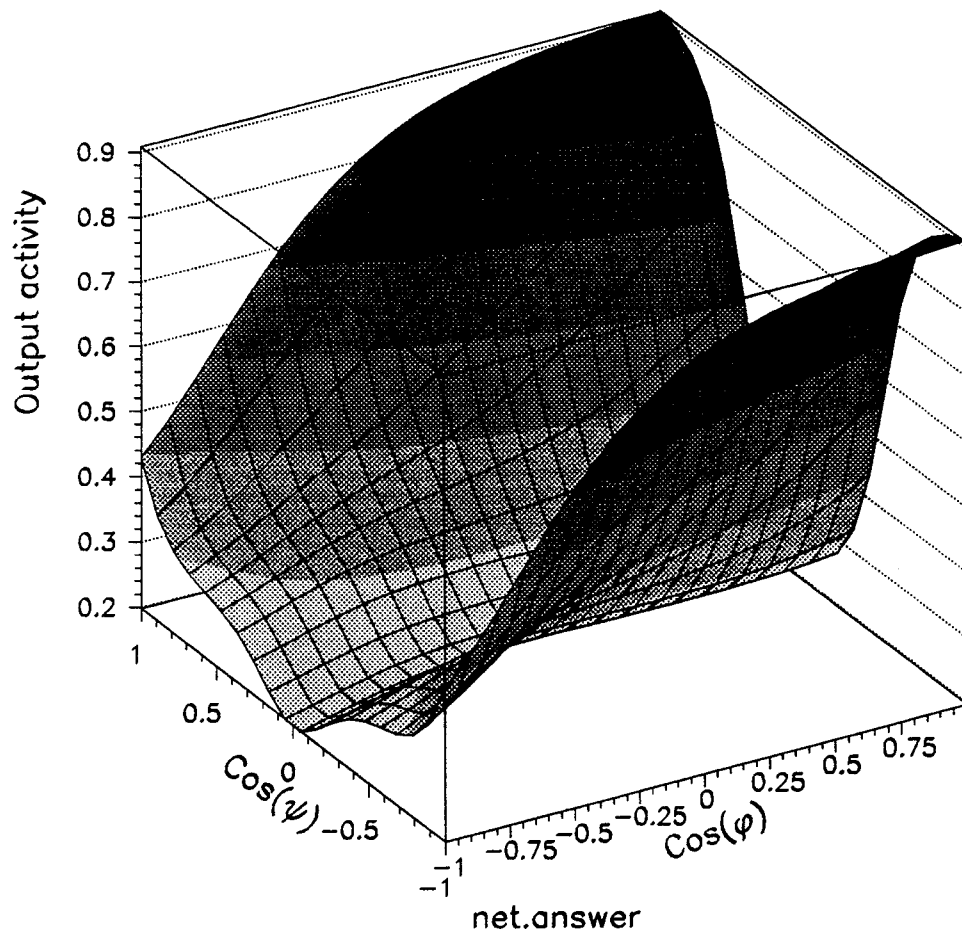


Figure 2.4: Activation of the output unit after convergence for the learning example $\tau^- \rightarrow \rho^- \nu$.

We found that the net has translated the input region to a smooth function between 0 and 1. Considering an output larger than β as the answer for positive helicity (expected is 1), and smaller than β as the answer for negative helicity (expected is 0), figure 2.5 shows the efficiency that the net has in finding the correct answer as function of β for an unpolarized sample ($\alpha'_+ = \alpha'_- = 0.5$).

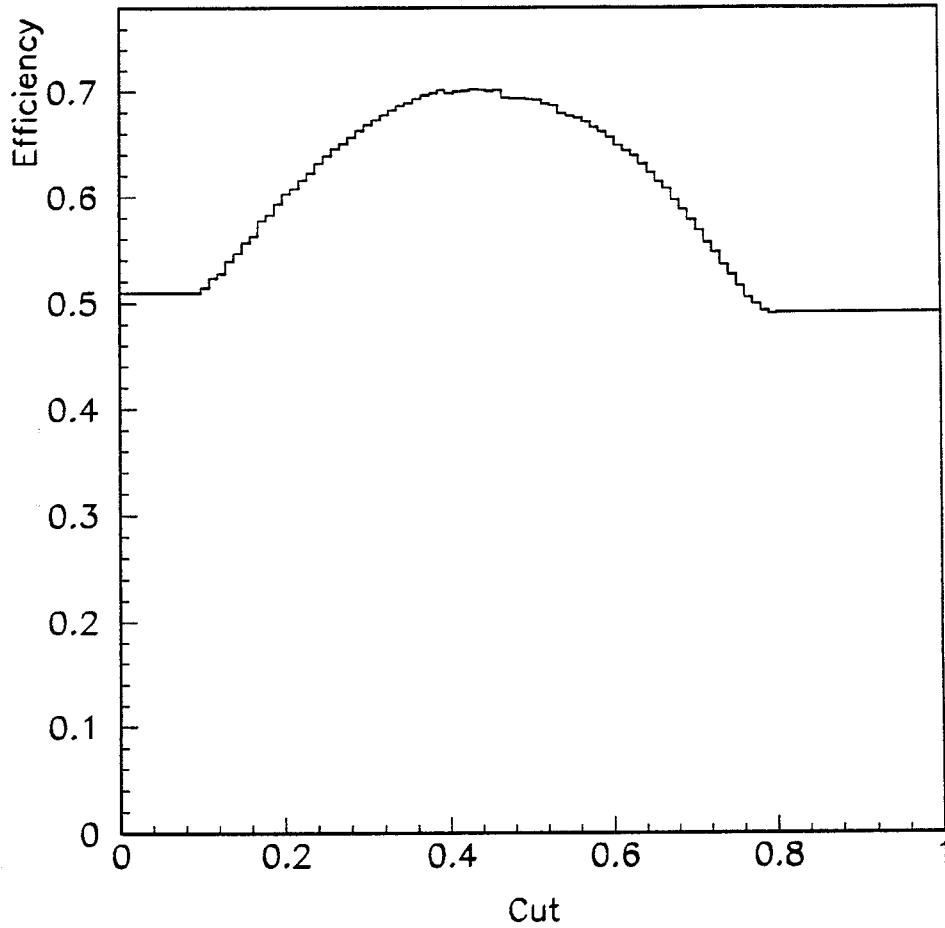


Figure 2.5: Efficiency in the classification found by the neural net as a function of the cut in the output unit.

We found that around β approximately 0.5 there is a plateau where the total efficiency is around 70% for both helicities. Choosing the cut exactly at $\beta = 0.5$

(as expected from last section when $\alpha_i = \alpha'_i$), figure 2.6 shows the dividing line between the two regions of different helicity.

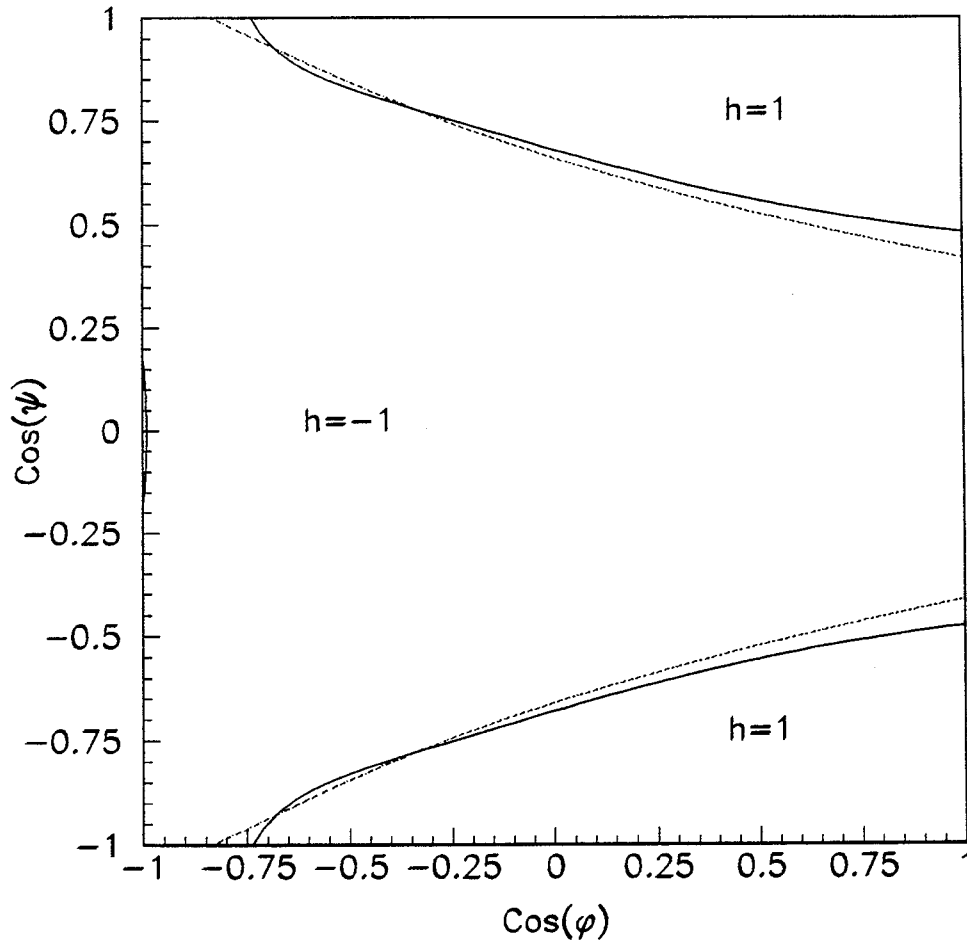


Figure 2.6: Boundary between the two zones corresponding to different helicities found by the neural net (dashed line) and the Bayesian method (solid line).

Let us now compare the efficiency found by the neural net method with a Bayesian discrimination: given $\cos \varphi$ and $\cos \psi$ for a τ^- decay we say that the τ^- has positive helicity if the probability that it has positive helicity, given by equation (2.10), is bigger than the probability of having negative helicity, given by equation (2.11). Using this method we obtain an efficiency of 70.2% which is essentially the

same as the neural net efficiency. In figure 2.6 we also show the separation contour of different helicities found by this statistical method. It agrees very well with the region found by the neural net method.

To show that really the answer of the net given in figure 2.4 is an approximation of the function $o(\vec{x})$ given by equation (2.6) we have performed the learning procedure over various samples with different α_i . Figure 2.7 shows the function $o(\vec{x})$ and the approximation given by a 2-3-1 network for three different α in the training sample. The topology of the network answer is simpler than the prediction of equation (2.6), but the shape and the variation with α is the same. Using a larger network the similarity between the two surfaces can be improved due to the existence of more degrees of freedom.

So far we have chosen the activation of the input units to be proportional to the variables we knew in advance were relevant to our problem. To test the ability of the net to recognize which variables are relevant to the problem and which are not, we have modified the configuration of our net by including a third unit in the input layer, whose activation is chosen at random for each τ decay. Figure 2.8 shows the state of the net after convergence. The width of the lines is proportional to $|w_{ij}|$.

As we see the third neuron of the input layer is almost disconnected from the net (connection weight very small), showing that the learning procedure has found that this neuron is irrelevant for the output.

We can also test the configuration dependence of the net. We choose a case similar to the original one: 2 units in the input layer, one hidden layer with 4 units and the output layer with 1 unit, but no bias input. With this configuration we reach 70%, very close to the limit of 70.2%. Also we have tried configurations with more than 1 hidden layer. As an example we have chosen a net with 2 units in the input layer, two hidden layers, the first one with 6 units and the second one with 3 units, and 1 unit in the output layer. The results are the same as in the first case.

So far we have been working with an ideal detector. To test how the distortions introduced by a real detector change the performance of the net, we have used a training sample of 10,000 Monte Carlo generated ρ 's reconstructed from the signals of a simulated tracking device with momentum resolution of 5% and a calorimeter with energy resolution of 6%. Using a net with the same configuration as the net in our previous example we get an efficiency of 70%, the same as in the ideal case. This shows that the network is able to distinguish the systematic distortions introduced by the detector and to extract the physical information available.

Another interesting question is the representation used for the input variables. As an example of two new representations we have given as an input to the net directly $\cos \varphi$ and $\cos \psi$ without normalization in the first case and the charged and neutral energies in the second. The efficiency obtained is exactly the same as before but the normalization of the energies helps the learning speed. This seems to show that we can choose a big number of representations for the input variables and only

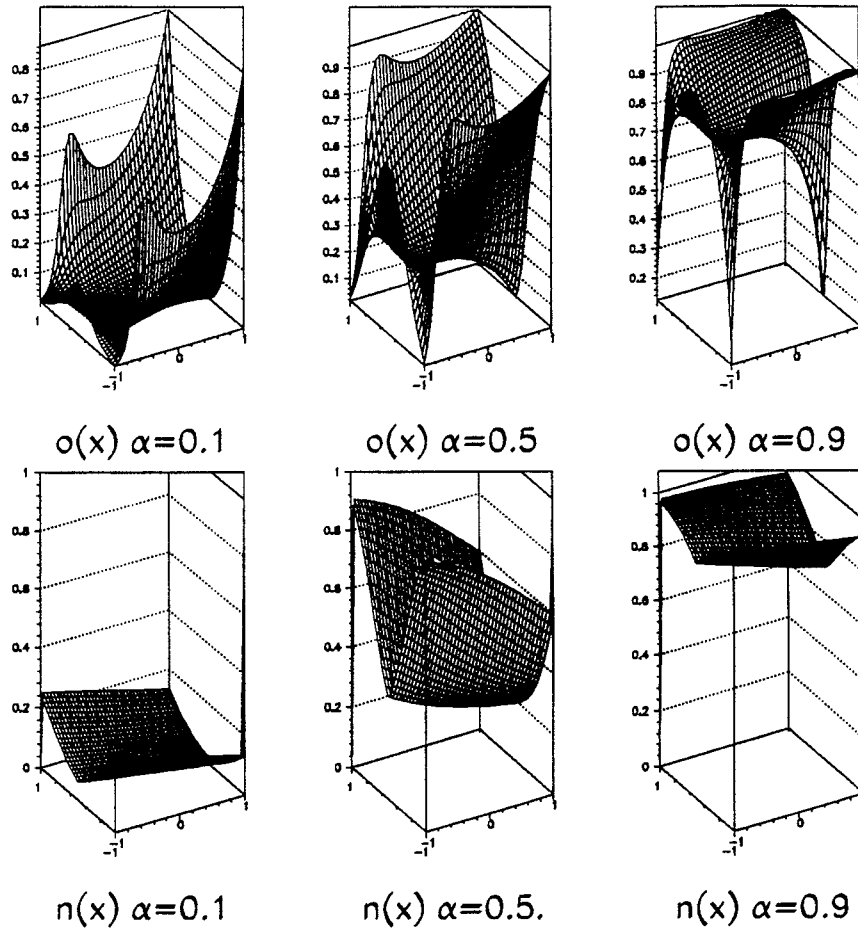


Figure 2.7: Net output predicted by equation (5) ($o(x)$) and learned by a 2-3-1 net ($n(x)$) vs. $\cos \varphi$ and $\cos \psi$ for three different α in the training sample.

the consideration of the learning speed can help us to decide about the best choice.

In this way, the normalized energies seem to be the most promising input in order to extend our method to other τ decays (a_1 , π , etc).

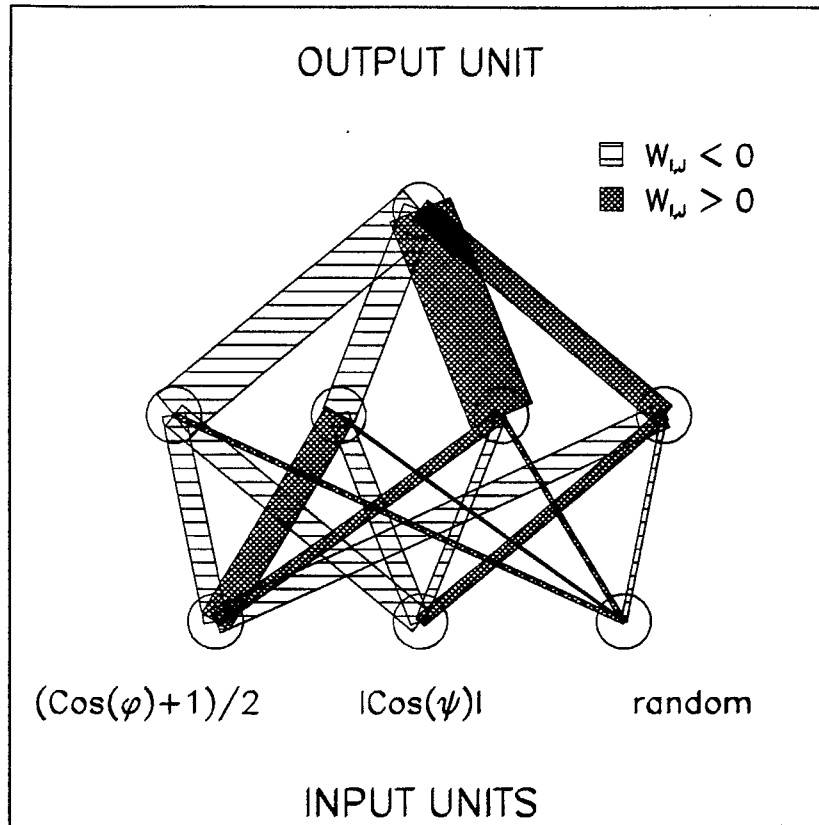


Figure 2.8: State of the net after convergence. The widths of the lines are proportional to $|w_{ij}|$.

2.2.2 Finding the helicity of both τ in the process: $e^+e^- \rightarrow Z^0, \gamma \rightarrow \tau^+\tau^- \rightarrow (\rho\nu)(\rho\nu)$

In the process $e^+e^- \rightarrow Z^0, \gamma \rightarrow \tau^+\tau^- \rightarrow (\rho\nu)(\rho\nu)$ at very high energy (energy of the τ much bigger than its mass) the two τ are produced with opposite helicity. 5% of the τ pairs produced in e^+e^- collisions have the above signature where both taus decay into $\rho + \nu$.

The neural net configuration that we have considered to attack the present problem has 4 units in the input layer, one hidden layer with 8 units, and 2 units in the output layer.

We have generated Monte Carlo events for the process $\tau^+\tau^- \rightarrow (\rho\nu)(\rho\nu)$ with the two τ having different helicities. The first 2 units of the input layer are activated by the quantities $(\cos\varphi + 1)/2$ and $|\cos\psi|$ of the τ^- decay and the next 2 units of the input layer are activated by the quantities $(\cos\varphi + 1)/2$ and $|\cos\psi|$ of the τ^+ decay. By convention we want the activation of the output 1,0 (0,1) when the τ^- has positive (negative) helicity.

After the learning procedure the net has been tested with a sample of new events. Taken the convention given in the last chapter of considering that an activation of an output unit bigger than 0.5 has to be interpreted as 1, and 0 for values less than 0.5, the efficiency that the net reaches is 77.2%. We see that this efficiency is higher than the one obtained before, showing that the net has been able to learn the existing correlation (the τ 's have opposite helicity). This efficiency is also similar to the one obtained using the Bayesian method (77.5%).

The comparison of the results of the net and the Bayesian method has been possible in the present case because we knew exactly the probability distributions. This will not be the case in a realistic situation where resolutions and detection cuts will make the exact probability distribution unknown. Nevertheless the neural net technique still works.

2.3 From Network output to results

It has been shown in the previous section that the NN output trained for classification to an unary codebook approximates the a posteriori bayesian probability to belong to a determinate class for a given input. There are several facts that must be clarified in order to correctly use this property.

Firstly, notice that bayesian a posteriori probability depends on the proportions used for its estimation. The probability computed by the NN is reliable only in the case that the test sample has the same proportions that the training sample. This can be overcome by the fact that knowing the a posteriori probability for a given proportion permits to find the a posteriori probability for any proportion. If $P_k^r = P(k|\vec{x}, \alpha^r)$ is the a posteriori probability when the training sample has an α_l^r proportion for class l , from Bayes theorem

$$P_k^r = \frac{\alpha_k^r p_k(\vec{x})}{\sum_l \alpha_l^r p_l(\vec{x})} \quad (2.13)$$

is easy to show that

$$\frac{p_k(\vec{x})}{p_l(\vec{x})} = \frac{P_k^r \alpha_l^r}{P_l^r \alpha_k^r} \quad (2.14)$$

Then, if P_j^s is the a posteriori probability assuming that the training sample has a proportion α_j^s for the class j

$$P_k^s = \left(\sum_l \frac{\alpha_l^s \alpha_k^r P_l^r}{\alpha_k^s \alpha_l^r P_k^r} \right)^{-1} \quad (2.15)$$

An important remark is that the network only gives an *approximation* to the probability. The goodness of this approximation depends on the NN degrees of freedom and sufficient statistics for the training sample.

It is clear from the above facts that it can be dangerous to classify the patterns making a cut in the network output. This approach should be biased to the proportions used in the training sample. This method should be used only in case that the different classes are well separated and the result of the classification does not strongly depend on the cut selection.

When there is overlap between classes the above approach does not work properly. In this case, the quality of the approximation can also play an important role.

In the next section a more robust use of the network output is presented that eliminates these drawbacks.

2.3.1 Matrix deconvolution method

In this method the fact that the NN output can be interpreted as a probability is not *explicitly* used. Let a sample of patterns belong to m different classes and a NN with m output neurons trained to a unary codebook. Let us assume that an input pattern is classified as k when the k -th output neuron has the highest activation (notice that this is a bayesian decision rule). Using a classified sample (the training sample for instance) it is possible to build an *efficiency matrix* E_{ij} defined as the quotient of the number of patterns belonging to class j classified as i , and the number of patterns j in the sample. In fact E_{ij} is the conditional probability to be classified as i when the pattern belong to the class j ($P(\hat{i}|j)$).

Clearly E_{ij} does not depend on the proportions used in the classified sample because E_{ij} is normalized to the number of patterns i in this sample. For a test sample, if N_j is the number of patterns classified as j by the network, C_i , the true number of patterns belonging to class i can be obtained using

$$C_i = \sum_j E_{ij}^{-1} N_j \quad (2.16)$$

The error in the C_i 's can be obtained by error propagation of the above formula. Notice that the function that maps the input space to the m -dimensional output neuron space can be any, but in general this will lead to a quasi-singular E matrix. A careful study of the above equation shows that the minimum error in the C_i is reached when E is the most diagonal possible. This fact is automatically verified when the mapping function implements a bayesian probability. This will be shown in the next section.

A common error is to use, instead of E , a matrix P_{ij} defined as the ratio between the number of patterns belonging to class i classified as j , and the total number of patterns classified as j , and use

$$C_i = \sum_j P_{ij} N_j \quad (2.17)$$

This formula is clearly biased to the proportions present in the classified sample. The reason is that

$$P_{ij} = P(i|\hat{j}) = \frac{\alpha_i P(\hat{j}|i)}{\sum_k \alpha_k P(\hat{j}|k)} \quad (2.18)$$

depends explicitly on α_i , when the α_i are the proportions for each class in the Monte Carlo. This approach is only correct when the test sample has the same proportions than the classified one, and this cannot be supposed in advanced (remember that the goal is to measure these proportions).

2.3.2 Fit method

The above method has the limitation that it inefficiently uses the information present in the output neurons. A more efficient way of handling this information is to build normalized reference output distributions for each class and fit the test distributions to a linear combination of the former. The matrix convolution method can be viewed as a particular case of a fit using only two bins for each dimension in the output neuron space. A question is: Why don't perform this fit directly in the input space? Obviously, the NN outputs cannot carry more information about the patterns that the information present in the input space. The answer is related to the fact that the number of examples needed to build the reference distributions grows exponentially with the dimension of the space. Normally, the number of classes is lower than the number of input variables. The only question that remains is related with the fact that making the projection into a lower dimension space we can have a non desired loss in the information on the proportion of each class. This point will be clarified in the next section.

2.3.3 Optimal choice of the fitting space

This section is devoted to show that given a n -dimensional sample composed by a mixture of m subsamples with different p.d.f., it is possible to build a $(m - 1)$ -dimensional distribution that carries all the information about the subsample proportions in the mixture sample. In this way, if $m - 1 < n$ it is possible to estimate the proportions in the mixture sample in the low dimensional $m - 1$ space without losing sensitivity.

In order to prove this consider $P_t(\vec{x}, \vec{\alpha}) = \sum_{i=1}^m \alpha_i p_i(\vec{x})$ the p.d.f for the mixture sample, where the $p_i(\vec{x})$ are the p.d.f. for the subsample and α_i their proportion. An estimator for the α_i can be obtained by maximizing the LogLikelihood function for the $\{\vec{x}_i\}$ sample

$$l_{\vec{x}} = - \sum_i \ln P_t(\vec{x}_i, \vec{\alpha}) + \lambda (\sum_i \alpha_i - 1) \quad (2.19)$$

Let $T(\vec{x}) = \vec{y} \oplus \vec{r}$ be an invertible transformation where $\dim(\vec{r}) = n - \dim(\vec{y})$. The p.d.f for the class i as a function of the y_i variables can be written as

$$q_i(\vec{y}) = \int d\vec{r} p_i(\vec{y}, \vec{r}) J_T(\vec{y}, \vec{r}) \quad (2.20)$$

where $J_T(\vec{y}, \vec{r})$ is the jacobian for the T transform. The goal is to find a set of variables $\vec{y}$ ($\dim(\vec{y}) < \dim(\vec{x})$) not depending on $\vec{\alpha}$, for which the LogLikelihood function

$$l_{\vec{y}} = - \sum_i \ln \sum_{k=1}^m \alpha_k q_k(\vec{y}_i) + \lambda (\sum_i \alpha_i - 1) \quad (2.21)$$

is the same as $l_{\vec{x}}$ up to a constant not depending on $\vec{\alpha}$. This assures not only that the expected value for the $\vec{\alpha}$ estimator will be the same, but thus its variance will be as well; in other words, the fit sensitivity will not be reduced.

Deriving $l_{\vec{x}}$ and $l_{\vec{y}}$ with respect to α_k and equalling the two expressions gives

$$\sum_i \left(\frac{1}{\sum_j \alpha_j \frac{p_j(\vec{x}_i)}{p_k(\vec{x}_i)}} - \frac{1}{\sum_j \alpha_j \frac{q_j(\vec{y}_i)}{q_k(\vec{y}_i)}} \right) = 0 \quad (2.22)$$

The only way to assure the above result for the whole $\vec{\alpha}$ range, is to impose

$$\frac{p_k(\vec{x})}{p_j(\vec{x})} = \frac{q_k(\vec{y})}{q_j(\vec{y})} \quad \forall j, k \quad (2.23)$$

Not all the above equalities are independent. Fixing j to an arbitrary class (for instance m) results in a set of $m - 1$ equations

$$p_i^*(\vec{x}) = \frac{p_i(\vec{x})}{p_m(\vec{x})} = \frac{q_i(\vec{y})}{q_m(\vec{y})} = q_i^*(\vec{y}) \quad (2.24)$$

for $i = 1, m - 1$.

The equation $z_i = p_i^*(\vec{x})$ maps the n -dimensional vector $\vec{x}$ to the $(m - 1)$ -dimensional space of $\vec{z}$. If $n < m - 1$, the $\vec{z}$ vector will span a n -dimensional manifold embedded in the $(m - 1)$ -dimensional space. This special case will be treated later. Let us assume that $n \geq m - 1$. Due to the last equality it must be possible to perform the transformation in two steps: first mapping the $\vec{x}$ vector to the $\vec{y}$ space (with unknown dimension), and then to the $\vec{z}$ space. The dimension of $\vec{y}$ cannot be lower than the dimension of $\vec{z}$. The most favorable case (such that minimizes the dimension of the projection space) is when the $\vec{y}$ vector is also $(m - 1)$ -dimensional. In this case it is possible to build explicitly an expression for $\vec{y}$ assuming that q^* is an invertible transformation

$$\vec{y} = q^{*-1}(p_i^*(\vec{x})) \quad (2.25)$$

so that, $\vec{y}$ must be any invertible transformation of $\frac{p_i(\vec{x})}{p_m(\vec{x})}, i = 1 \dots m - 1$.

A convenient choice is to use

$$\kappa_i = \frac{\alpha_i^0 p_i}{\sum_k \alpha_k^0 p_k} \quad (2.26)$$

with α_k^0 arbitrary numbers fulfilling $\sum_k \alpha_k^0 = 1$.

A neural net trained with a sample with proportions $\vec{\alpha}^0$ provides an approximation to this κ_i for a given event. This justifies that a fit in the output neuron space has the same sensitivity (up to the goodness of the $\vec{\kappa}$ approximation given by the network) as performing the fit in the input variables space. The Matrix method is a special case of the fit method with only two bins per output neuron. The above proof explains why the bayesian function provides an optimal efficiency matrix .

When the number of classes becomes larger than the number of inputs this approach is not practical because there is no reduction on the dimensionality of the fit space. Nevertheless the unidimensional projections of the output space carries enough information about the proportions in the sample as will be seen later. Thus, a simultaneous fit of the one-dimensional output neuron projections taking into account the corresponding correlations is proposed.

From the above result is derived that a single κ_i has full sensitivity to its corresponding α_i when the training sample has the same proportions that the sample we are trying to classify for all $j \neq i$ except a global factor. This can be shown by applying the proof for only two classes: p_i and the rest

$$p_r = \frac{\sum_{j \neq i} \alpha_j^0 p_j}{\sum_{j \neq i} \alpha_j^0} \quad (2.27)$$

supposing that the α_j^0 have values equal to the real proportions α_j for all $j \neq i$ except a global factor. The single variable

$$\kappa_i = \frac{\alpha_i^0 p_i}{\alpha_i^0 p_i + (1 - \alpha_i^0) p_r} \quad (2.28)$$

has the same information about the i class proportion as all the input space. *No other unidimensional projection can be more sensitive than κ_i* (up to an invertible transformation of the κ_i). For practical applications the α_i^0 are chosen very close to the expected proportions in the sample to classify. In that case, from what has been described above, it is expected that each single κ_i is very sensitive to its corresponding α_i . This explains why we don't perform unidimensional fits directly in the input space but it can be performed in the neuron output space (κ_i).

When the simultaneous fit of the one-dimensional output neuron projection is performed, a single event enters one time in each unidimensional projection appearing bin to bin correlations. A general way to calculate these correlations is presented here.

Let $N_{a_i b_j \dots z_n}$ denote the number of events laying simultaneously in the bin i of histogram a , bin j of histogram $b \dots$ and bin n of histogram z . Let us assume that the $N_{a_i b_j \dots z_n}$ are normal uncorrelated variables. N_{a_i} , the number of entries in bin i for the histogram a can be expressed as

$$N_{a_i} = \sum_{b_j \dots z_n} N_{a_i b_j \dots z_n} \quad (2.29)$$

The covariance matrix can then be expressed as

$$\text{cov}(N_{a_i}, N_{b_j}) = \sum_{b_k c_m \dots z_n} \sum_{a_l c_{m'} \dots z_{n'}} (< N_{a_i b_k c_m \dots z_n} N_{a_l b_j c_{m'} \dots z_{n'}} > - < N_{a_i b_k c_m \dots z_n} > < N_{a_l b_j c_{m'} \dots z_{n'}} >) \quad (2.30)$$

All the terms in the sum are zero except for the case that the indices for the two sums are equal. This results in

$$\text{cov}(N_{a_i}, N_{b_j}) = \sum_{c_m \dots z_n} \text{cov}(N_{a_i b_j c_m \dots z_n}, N_{a_i b_j c_m \dots z_n}) = \sum_{c_m \dots z_n} \sigma_{a_i b_j c_m \dots z_n}^2 = N_{a_i b_j} \quad (2.31)$$

So that, the correlation between bins i of histogram a and j of histogram b can be computed as

$$\text{corr}(N_{a_i}, N_{b_j}) = \frac{N_{a_i b_j}}{\sqrt{N_{a_i}} \sqrt{N_{b_j}}} \quad (2.32)$$

It is convenient to compute this correlation matrix from the data because the correlations depends on the α 's. This matrix will be used when performing the fit rescaled by the variance on the number of events of each bin for the real data plus the same quantity for the Monte Carlo. Notice that the procedure consisting in fitting the one-dimensional output projections using the correlation matrix is not strictly equivalent to a fit in the whole multidimensional output space because we are neglecting higher order correlations. Nevertheless this high order correlations are normally very small (This must be checked for each particular problem) and the fit result must be statistically compatible in both cases.

In order to illustrate the methodology proposed, a toy example will be discussed. Let be a sample of events belonging to three different classes described by the two-dimensional p.d.f shown in Fig 2.9 top.

A 2-X-2 NN (where X stands for any hidden layer) trained with equal proportions of each class will learn an approximation to the conditional probabilities shown in Fig 2.9 center as expected from eq 2.8. (Notice that for three classes only two output neurons are needed because the three conditional probabilities must sum 1.)

The two-dimensional (O_2 versus O_1) output distributions for each class is shown in Fig 2.9 bottom. These will be called *reference distributions*, when properly normalized. The proportions for each class in a test sample can be obtained by adjusting the total output neuron distribution for this sample to a linear combination of the three two-dimensional reference distributions.

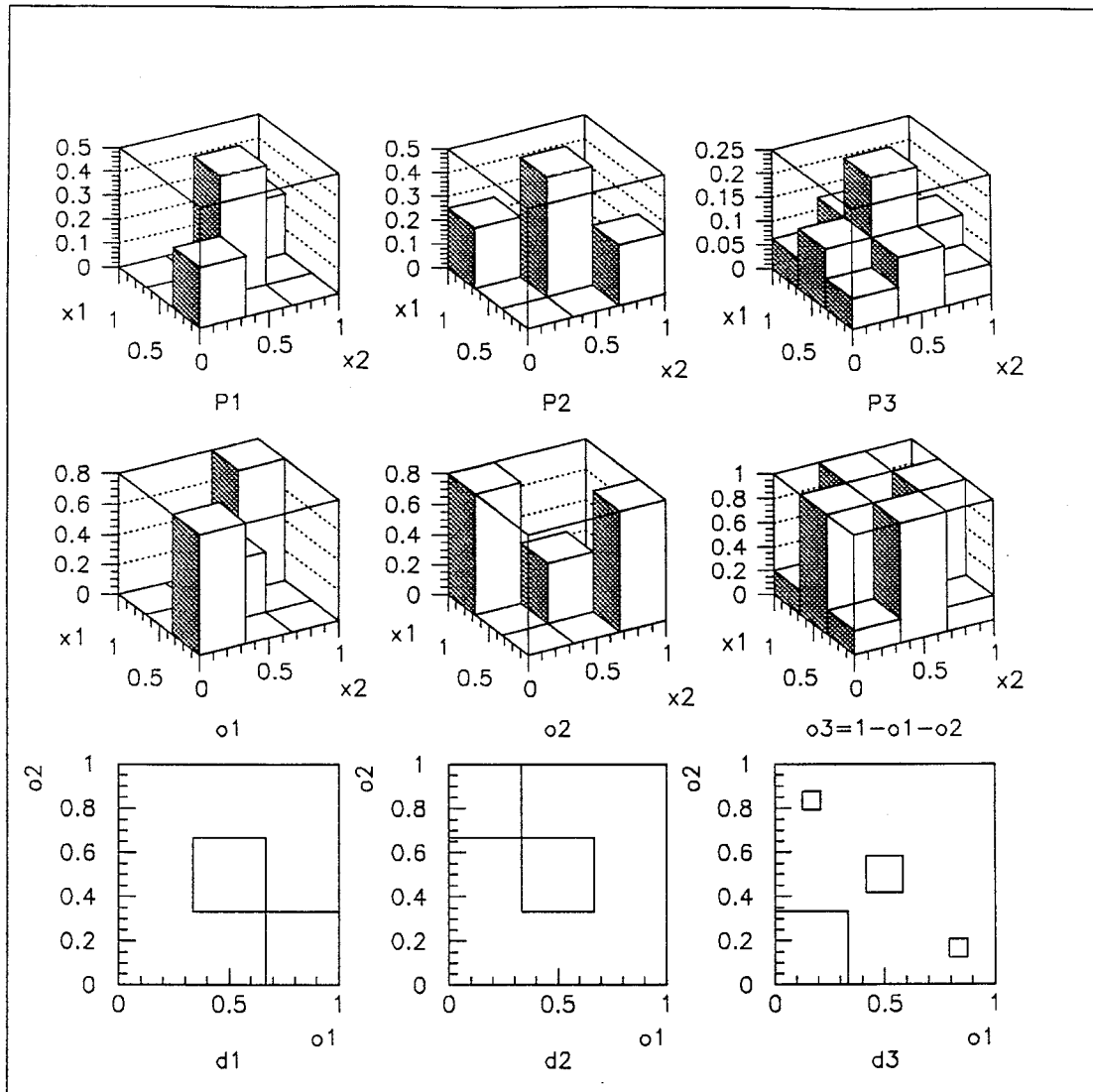


Figure 2.9: Top: Bidimensional probability distribution functions for three classes. Center: Function learned by a 2-X-2 neural net trained to distinguish the three classes. The third output neuron (O_3) is redundant. Bottom: bidimensional neuron output distributions for each class.

Another approach is to use one-dimensional projections for doing this task. The

one-dimensional projections in the input space are exactly the same for the three classes, then there is no sensitivity in this projections to the class. Nevertheless, the corresponding one-dimensional projections (one-dimensional reference distributions) in the output space (Fig 2.10 top and center) are fully sensitive to the proportions. (In fact, in this case a single one dimensional projection will suffice to calculate the proportions, but this is not generally true).

Figure 2.10 bottom left shows the two dimensional output distribution for a mixture sample with proportions 0.2,0.3,0.5. The other two figures at the bottom show the projections of this distribution over the two output neurons (solid line), and its composition in terms of the three single classes (Dashed line: class 1, Dotted line: class 2, Dotted-Dashed line: class 3) obtained by making a fit of these two projections to a linear combination of the reference unidimensional distributions. The proportions obtained are in good agreement with the original ones.

From this example is clear that the use of one-dimensional projections in the output neuron space is a reliable tool for calculating the proportions in a certain sample by making a one-dimensional fit instead of performing a multidimensional fit, that normally is more difficult. This is the method we will apply in following chapters.

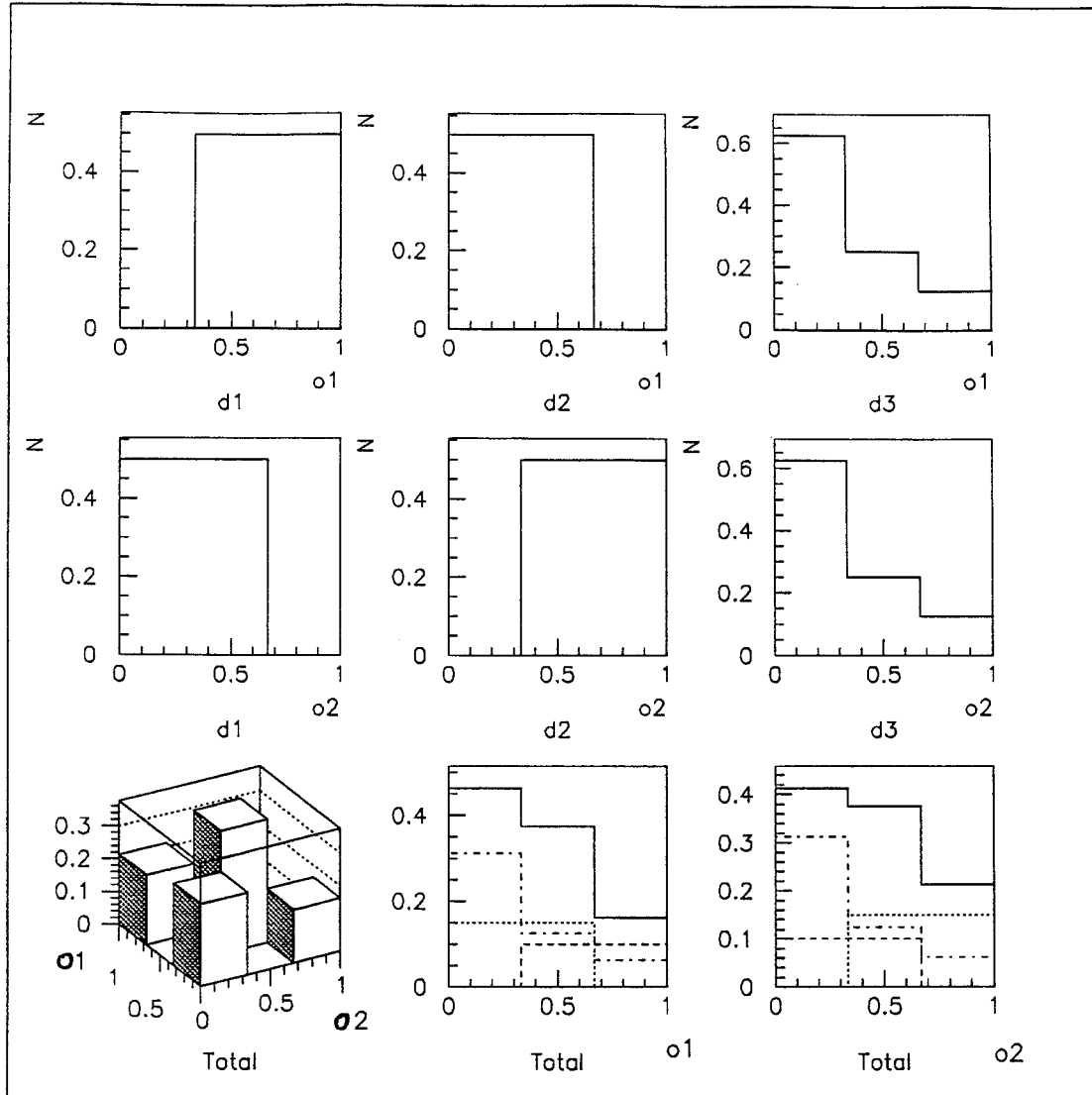


Figure 2.10: Top: One-dimensional reference distribution for each class corresponding to output neuron one. Center: One-dimensional reference distribution for each class corresponding to output neuron two. Bottom left: two dimensional output distribution for a mixture sample with proportions 0.2,0.3,0.5. The other two figures at the bottom show the projections of this distribution over the two output neurons (solid line), and its composition in terms of the three single classes (Dashed line: class 1, Dotted line: class 2, Dotted-Dashed line: class 3) obtained by making a fit of these two projections to a linear combination of the reference unidimensional distributions. The proportions obtained are in good agreement with the original ones.

2.4 Testing the agreement between multidimensional distributions.

2.4.1 Introduction

An everyday task in all areas of science is the comparison of theoretical models with experimental data. In some cases the theoretical models cannot be checked directly because their explicit analytical form does not exist (e.g.: the models are the result of a large number of calculations with complex algorithms) and/or there are additional effects not included in the model (e.g.: detector effects during the acquisition of the experimental data). A commonly used strategy to overcome this problem is to generate Monte Carlo events according to the theoretical model and to fold them with the additional effects. The result is a simulated data sample (MC) that can be checked against the experimental data.

The check between the experimental data and the MC is normally done by the comparison of single distributions for the most relevant variables. To evaluate the agreement between them, statistical tests like the χ^2 , the Likelihood or the Kolmogorov test are done [19] . In cases where the number of variables is low, bidimensional distributions of pairs of variables are also checked in order to see higher-order effects.

The ideal method would be to compare the two m-dimensional distributions globally, in which case the standard methods need to bin the variable space. Even for a low number of bins per variable, large amounts of data may be necessary, because the number of data points needed to fill the bins with statistical significance grows exponentially with the number of variables.

Here we want to present a new test which is free of binning, that has the particularity of looking at the two m-dimensional distribution of the variables as global distributions, being able to find deviations that traditional methods normally cannot find. The method transforms a difficult test in the m-dimensional input space to a simple test in a one-dimensional space with the same sensitivity. This dimensional reduction is done without losing the relevant information by using a suitable projection function.

2.4.2 The method

Let us assume that we have two samples with n_1 and n_2 independent observations, and we want to test the hypothesis that both samples can be described by the same probability density function (pdf): $P_1(\vec{x}) = P_2(\vec{x})$, where $\vec{x} \in \mathbf{R}^m$.

If the data is one-dimensional, several tests can be performed like Pearson's χ^2 test or the Kolmogorov test. The major difference between them is that while in

the former it is necessary to bin the observations, the later is free of binning.

When multidimensional data is treated there is no equivalent to the unbinned Kolmogorov test in one dimension and a normal procedure is to bin the space and perform a χ^2 test. Only some extensions of the Kolmogorov test to two-dimensional data exists[24]. When discretizing the space the volume of the bins can vary and a common accepted criterion to determine it, is that the number of expected events per bin is greater than 5. Assuming for simplicity that the number of observations for the two samples is the same ($n_1 = n_2 = n$) the χ^2 can be written as

$$\chi^2 = \sum_{i=1}^{N_{bin}} \frac{(n_{1i} - n_{2i})^2}{n_{1i} + n_{2i}} \approx \sum_{i=1}^{N_{bin}} \frac{(n_{1i} - n_{2i})^2}{\sigma_{n_{1i}-n_{2i}}^2} \approx \sum_{i=1}^{N_{bin}} \frac{(n_{1i} - n_{2i})^2}{n(P_1(\vec{x}_i) + P_2(\vec{x}_i))\Delta V_i} \quad (2.33)$$

where the sum runs over all the bins (if the two distributions are normalized to the same number of events, N_{bin} is taken as the total number of bins minus 1 to avoid undesired correlations), n_{ji} is the number of events of sample j at cell i , and we have used the relation $\sigma_{n_{1i}-n_{2i}}^2 = \sigma_{n_{1i}}^2 + \sigma_{n_{2i}}^2 = \langle n_{1i} \rangle + \langle n_{2i} \rangle \approx n(P_1(\vec{x}_i) + P_2(\vec{x}_i))\Delta V_i$, where $P_j(\vec{x}_i)$ is the pdf of class j at cell i , and ΔV_i its volume. Assuming that the two samples are described by the same $P(\vec{x})$ it is well known that the expectation value $\langle \chi^2 \rangle$ is N_{bin} .

Let us try to estimate $\langle \chi^2 \rangle$ in case that the two samples are not described by the same probability distributions, $P_1(\vec{x}) \neq P_2(\vec{x})$. In such case

$$\begin{aligned} \langle \chi^2 \rangle &= \sum_{i=1}^{N_{bin}} \frac{\langle (n_{1i} - n_{2i})^2 \rangle}{n(P_1(\vec{x}_i) + P_2(\vec{x}_i))\Delta V_i} = \\ &= \sum_{i=1}^{N_{bin}} \frac{\sigma_{n_{1i}-n_{2i}}^2 + \langle (n_{1i} - n_{2i}) \rangle^2}{n(P_1(\vec{x}_i) + P_2(\vec{x}_i))\Delta V_i} = \\ &= \sum_{i=1}^{N_{bin}} \frac{P_1(\vec{x}_i) + P_2(\vec{x}_i) + nP_1(\vec{x}_i)^2\Delta V_i + nP_2(\vec{x}_i)^2\Delta V_i - 2nP_1(\vec{x}_i)P_2(\vec{x}_i)\Delta V_i}{P_1(\vec{x}_i) + P_2(\vec{x}_i)} \approx \\ &= \sum_{i=1}^{N_{bin}} 1 + n \int \frac{(P_1(\vec{x}) - P_2(\vec{x}))^2}{P_1(\vec{x}) + P_2(\vec{x})} d\vec{x} \end{aligned} \quad (2.34)$$

and defining $I \equiv \int (P_1(\vec{x}) - P_2(\vec{x}))^2 / (P_1(\vec{x}) + P_2(\vec{x})) d\vec{x}$ we finally obtain

$$\langle \chi^2 \rangle = N_{bin} + n \int \frac{(P_1(\vec{x}) - P_2(\vec{x}))^2}{P_1(\vec{x}) + P_2(\vec{x})} d\vec{x} = N_{bin} + nI \quad (2.35)$$

In the case that $P_1(\vec{x}) = P_2(\vec{x})$ we recover $\langle \chi^2 \rangle = N_{bin}$, but in the case that both sample sets come from different distributions $\langle \chi^2 \rangle$ is shifted by the term nI , which is proportional to n showing that the sensitivity increases with the number of events. It is easy to show that the integral I is functionally invariant under

change of variables. From the last equation it is clear that any method sensitive to I can be used to test the hypothesis.

Defining $\kappa \equiv P_1(\vec{x})/(P_1(\vec{x}) + P_2(\vec{x}))$ we obtain

$$\langle \chi^2 \rangle = N_{bin} + n(\langle \kappa \rangle_1 + \langle 1 - 3\kappa \rangle_2) \quad (2.36)$$

where $\langle \dots \rangle_i$ means the expectation value of sample i . This expression assures that knowing the projected one-dimensional distribution densities $P'_i(\kappa)$ instead of the m -dimensional original densities $P_i(\vec{x})$, we still have all the necessary information to obtain I . In fact performing a χ^2 test in the projected κ space we will obtain

$$\langle \chi^2 \rangle \approx N'_{bin} + n \int \frac{(P'_1(\kappa) - P'_2(\kappa))^2}{P'_1(\kappa) + P'_2(\kappa)} d\kappa \quad (2.37)$$

and as we will show the last term in this equation has the same value that nI of equation 2.35, as expected from what was mentioned before. In order to show that, consider a transformation from the $\vec{x}$ space to $\kappa \oplus \vec{r}$, where $\vec{r}$ lies in the space orthogonal to κ and is chosen to obtain an invertible transformation. This change of coordinates gives an expression for $P_i(\vec{x})$

$$P_i(\vec{x}) = P'_i(\kappa, \vec{r}) \left| \frac{\partial(\kappa, \vec{r})}{\partial \vec{x}} \right| \quad (2.38)$$

Therefore

$$\frac{P'_1(\kappa)}{P'_1(\kappa) + P'_2(\kappa)} = \frac{\int d\vec{r} P'_1(\kappa, \vec{r})}{\int d\vec{r} (P'_1(\kappa, \vec{r}) + P'_2(\kappa, \vec{r}))} = \frac{\int d\vec{r} P'_1(\kappa, \vec{r})}{\frac{1}{\kappa} \int d\vec{r} P'_1(\kappa, \vec{r})} = \kappa \quad (2.39)$$

which implies

$$I' \equiv \int d\kappa \frac{(P'_1(\kappa) - P'_2(\kappa))^2}{P'_1(\kappa) + P'_2(\kappa)} = \langle \kappa \rangle_1 + \langle 1 - 3\kappa \rangle_2 = I \quad (2.40)$$

It means that changing the description of our two data samples from the original variables $\vec{x}_i$ to κ_i and performing then a χ^2 test or a Kolmogorov test, we will have the same sensitivity as in the original m -dimensional space.

The problem arises from the fact that we do not know $P_1(\vec{x})$ and $P_2(\vec{x})$, which seems to make the projection in the κ space unrealistic. Nevertheless we propose here a method to estimate this projection.

Let us define $E[o(\vec{x})]$ as

$$E[o(\vec{x})] = \int d\vec{x} (P_1(\vec{x})(o(\vec{x}) - 1)^2 + P_2(\vec{x})o(\vec{x})^2) \quad (2.41)$$

where $o(\vec{x})$ is an arbitrary function of $\vec{x}$. Assuming no constraint on the form of $o(\vec{x})$, the functional minimum of $E[o(\vec{x})]$ is reached when

$$\frac{\delta E[o(\vec{x})]}{\delta o(\vec{x}')} = 2 \int d\vec{x} (P_1(\vec{x})(o(\vec{x}) - 1) + P_2(\vec{x})o(\vec{x})) \delta(\vec{x} - \vec{x}') = 0 \quad (2.42)$$

i.e.

$$o(\vec{x}) = \frac{P_1(\vec{x})}{P_1(\vec{x}) + P_2(\vec{x})} \quad (2.43)$$

Notice that eq. 2.43 is just the definition of κ . The procedure now becomes clear: estimate the arbitrary function $o(\vec{x})$ from the two data samples minimizing

$$E = \sum_{i \in \text{sample}_1} (o(\vec{x}_i) - 1)^2 + \sum_{j \in \text{sample}_2} o(\vec{x}_j)^2 \quad (2.44)$$

and use this estimation of κ to make the projections. As we will explain in the next section, good candidates for parametric functions $o(\vec{x})$ can be layered neural nets due to their capacity of interpolating sparse data [47, 25].

2.4.3 Neural Net implementation

The basic idea is to train a layered feed-forward neural net to distinguish the two samples. This will be called NNAC (Neural Net Agreement Confidence). The input of such a net contains n neurons that are activated by the quantities of the n relevant variables to the problem and the output is only one neuron, for which the desired activation is 1 for the first sample and 0 for the second sample. To teach the net to give the correct answer we minimize

$$E = \sum_{i \in \text{sample}_1} (o(\vec{x}_i, \vec{w}) - 1)^2 + \sum_{j \in \text{sample}_2} o(\vec{x}_j, \vec{w})^2 \quad (2.45)$$

as a function of the connection weights between the neurons ($\vec{w}$ parameters) using the well-known error Back-Propagation method[14, 4].

After the training step, during which a sample containing a mixture of experimental data and MC is presented to the net, we obtain a function $f(\vec{x}) : \mathbf{R}^m \rightarrow \mathbf{R}$ that maps the m -dimensional space of the input variables to $[0, 1] \in \mathbf{R}$, which, as we know from eq. 2.43, has to be an approximation to κ . The goodness of the approximation depends upon the architecture of the net, the ability to escape from local minima during the training step and the existence of adequate data.

It is important to mention that one of the two samples used during the learning step should not be the same as the one used to give the level of agreement in order to avoid the fact that the neural net can learn statistical fluctuations as a real difference. A typical example is the comparison between experimental data and generated MC data. In this case the MC sample used during the learning step of the neural net has to be different, but statistically compatible, with the one used to compare experimental data with the MC data.

Another important feature of the NNAC here is the possibility, once we have the evidence that both samples are not compatible, to trace back where in the input space is located the difference. This can be done in different ways, some of which will be shown in the following examples.

2.4.4 Examples and results

In order to show how this neural method performs in non-trivial conditions, we build a couple of artificial examples with inherent drawbacks for standard methods. In the first example we will try to distinguish between two 10-dimensional distributions with the same projection in one dimension. In the second example we will try to distinguish between two 3-dimensional distributions which have identical projections in one and two dimensions.

In the first case the probability density functions for samples 1 and 2 are :

$$P_i(\vec{x}) = \frac{1}{(2\pi)^{n/2} \sqrt{\det(M_i)}} e^{-\frac{1}{2}(\vec{x}-\vec{\bar{x}})^T M_i^{-1}(\vec{x}-\vec{\bar{x}})}, i = 1, 2 \quad (2.46)$$

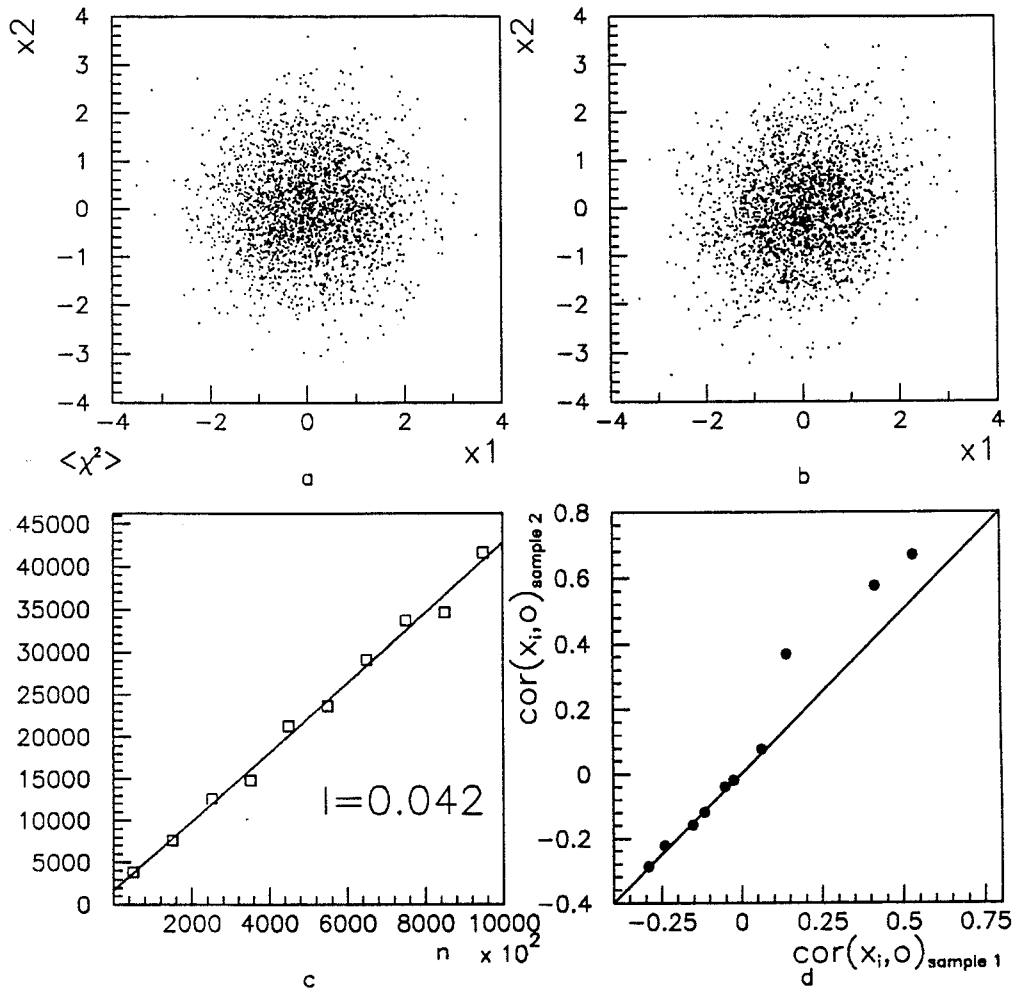
where M is the 10×10 covariance matrix of the x variables. Both sets of variables have the same means and standard deviations, but in the first sample there are no correlations between them and in the second sample the first three variables have a correlation of 20% between them. Fig. 2.11.a shows a bidimensional projection between any two variables of the first sample and Fig. 2.11.b is the same plot for two correlated variables of the second sample. The differences between Fig 2.11.a and Fig. 2.11.b shows the level of disagreement that we want our method to be able to find.

To test all formulae given in section 2.4.2 we have checked expressions 2.35 and 2.37. The value of the integral I in eq. 2.35 obtained using numerical integration is in this case $\approx .05$. Generating several event samples and making the κ projection, figure 2.12.a gives the variation of the $\langle \chi^2 \rangle$ test as a function of n at the 10 dimensional input space, and fig 2.12.b shows the same quantity in the projected κ space. The slope in both cases is very close to the expected value 0.05, showing that the sensitivity is the same in both spaces.

To test the performance of the neural nets estimating the value of I we have used a Feed-Forward Layered neural net with 10 input units, 9 units in a single hidden layer, and one unit as output (by convention this is a 10-9-1 net), trained with Back-Propagation, and demanding that the desired output is 0 for the uncorrelated sample and 1 for the other. After the training step the neural net is able to discover the slight differences between the two samples, as can be seen in Fig. 2.13.b, where we plot the distributions of the output of the net. This can be compared with the exact κ distributions given in figure 2.13.a.

All distributions are located around .5 (the number of examples used for each distribution is the same.), but if we look at the histograms for each sample separately, a shift evidences the different correlations. The Kolmogorov test between the two distributions of the net outputs of Fig 2.13.b gives a probability that the two are truly compatible of less than 10^{-4} , showing the power of the method presented. Studying the dependence of $\langle \chi^2 \rangle$ as a function of n (Fig 2.11.c) we have found a slope of .04 very close to the expected .05 .

Figure 2.11: a) Bidimensional projection (x_1 vs x_2) for sample 1. b) Bidimensional projection (x_1 vs x_2) for sample 2. c) $\langle \chi^2 \rangle$ as function of the number of events "n" between the output of the net for sample 1 and sample 2. d) Correlation between the output neuron and each input variables for sample 1 versus the same quantity for sample 2. The three points out of the diagonal corresponds to the variables with correlation at the original space.



Another important fact, as mentioned above, is that the network output can be used to understand the differences between the samples. Here we present a method based on the study of the correlations between the input variables and the net

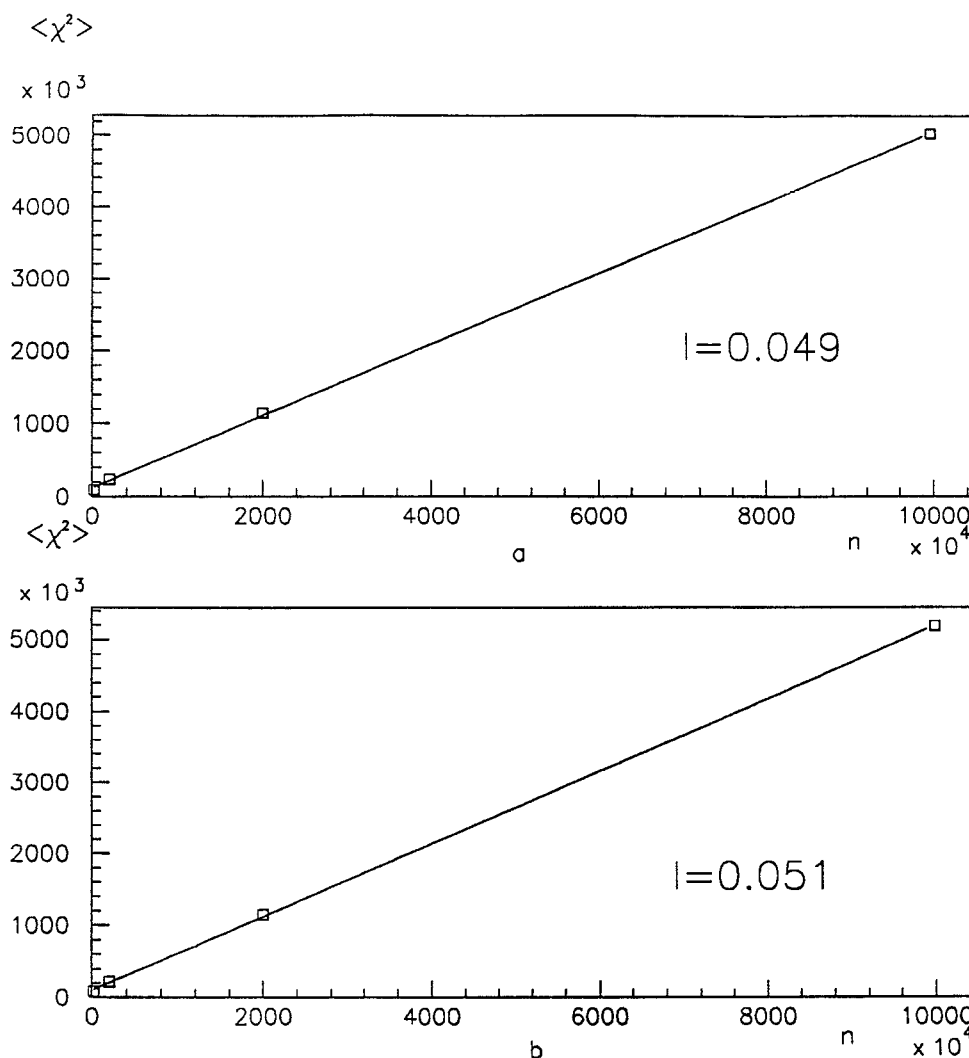


Figure 2.12: Plot of $\langle \chi^2 \rangle$ vs the number of events n of each sample, using the known probabilities distributions a) at the input space. The slope of the fitted line gives $I = 0.049$. b) at the κ space. The slope of the fitted line gives $I = 0.051$.

output for both samples. We would expect no correlation when the two samples have the same underlying distribution because in such case the network output should not depend on the inputs; nevertheless, due to the approximation introduced by the net a correlation different from 0 can appear, but has to be statistically compatible between the samples. Only in cases where the two samples have different underlying distributions, the variables causing this effect may have correlations that

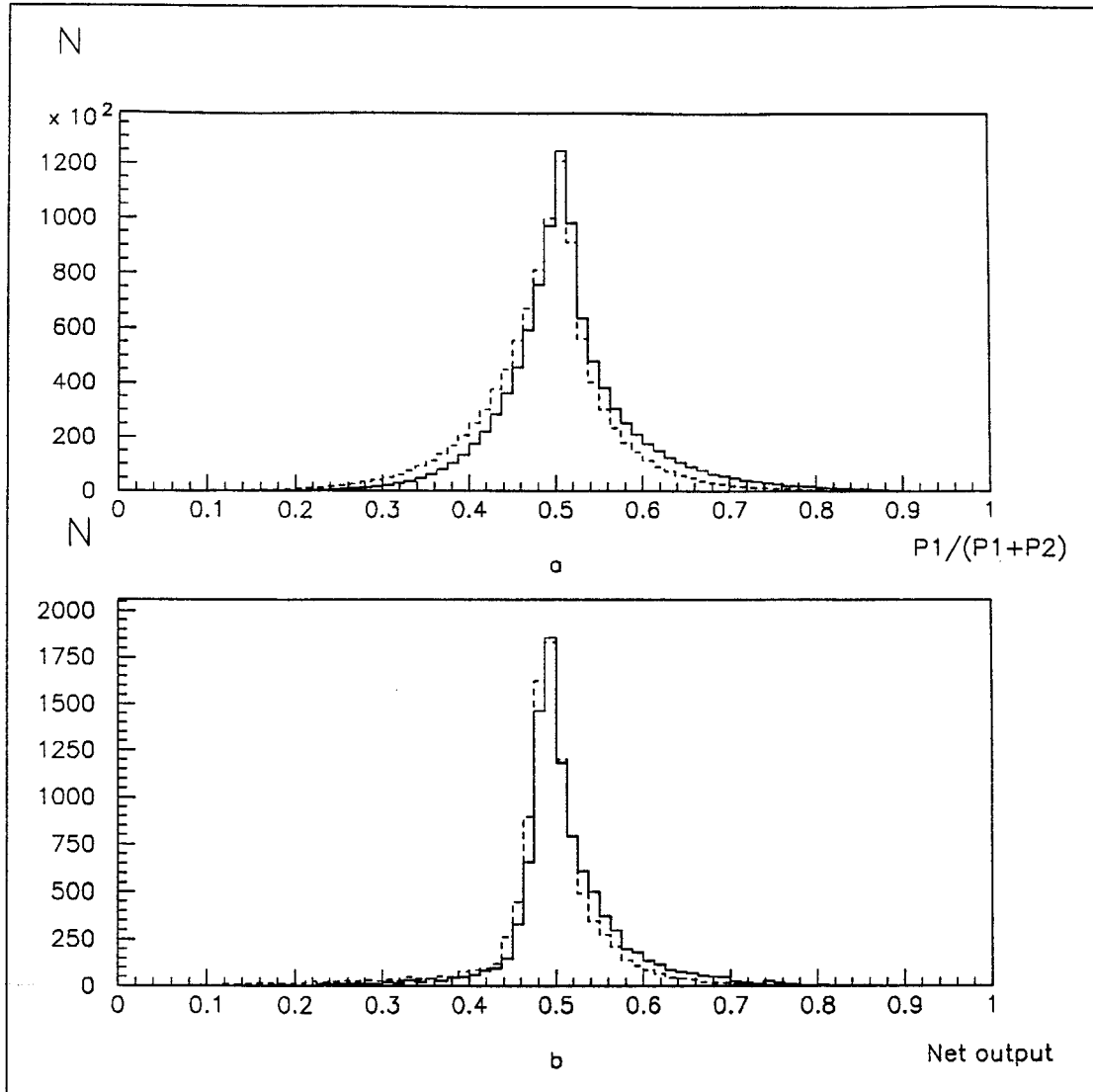


Figure 2.13: Distribution of events in the projected κ space (full line corresponds to sample 1 and dashed line to sample 2) a) Calculated from the analytical expressions ($\kappa = P_1/(P_1 + P_2)$). b) Estimated from the output of the neural network.

are not statistically compatible. Fig. 2.11.d shows the correlations between the input variables and the net output for sample 1 versus the same quantities for sample 2. We can see that points far from the diagonal (corresponding to the first three variables) are the ones causing the difference, as we expect from how they have been generated.

In the second example we want to distinguish the following two distributions

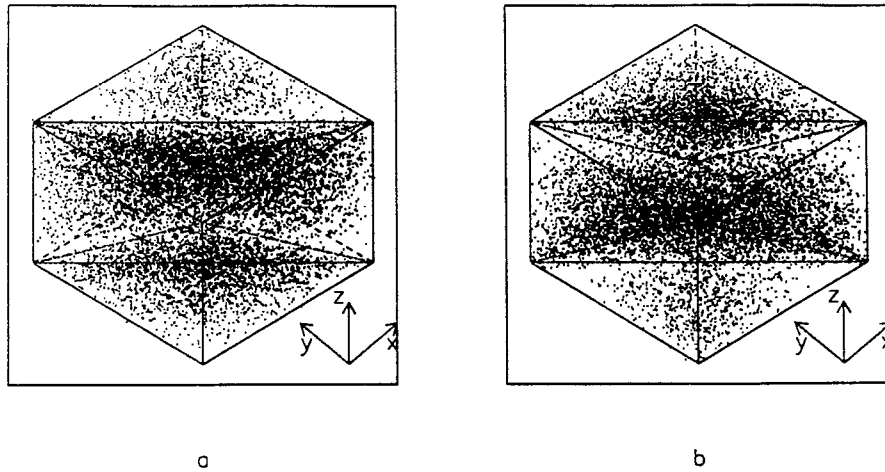


Figure 2.14: View of the 3-dimensional generated samples (The two triangles plotted corresponds to the intersection between the cube and the planes $x + y + z = 1$ and $x + y + z = 2$ respectively). a) view of sample 1. b) view of sample 2.

$$p_1(x, y, z) = \frac{1}{c_1} \left(\frac{1}{2} - \left| \text{frac}(x + y + z) - \frac{1}{2} \right| \right) \quad (2.47)$$

and

$$p_2(x, y, z) = \frac{1}{c_2} \left(\left| \text{frac}(x + y + z) - \frac{1}{2} \right| \right) \quad (2.48)$$

where $\text{frac}(x)$ is the fractional part of x , c_1 and c_2 are normalization constants, and both functions are only defined in the cube $[0, 1]^3 \in \mathbb{R}^3$. As mentioned above, the projections of these distributions are exactly the same in one and two dimensions and only special projections not parallel to the X, Y, Z axis can show differences (Fig. 2.14.a,b).

A 3-5-1 network, however, finds a disagreement between the two samples, as seen in Fig 2.15.a. A Kolmogorov test between them is not necessary due to the evident incompatibility.

The correlation study, described above, shows for this example that all 3 variables are responsible for the difference (Fig. 2.15.b). The two regions with more

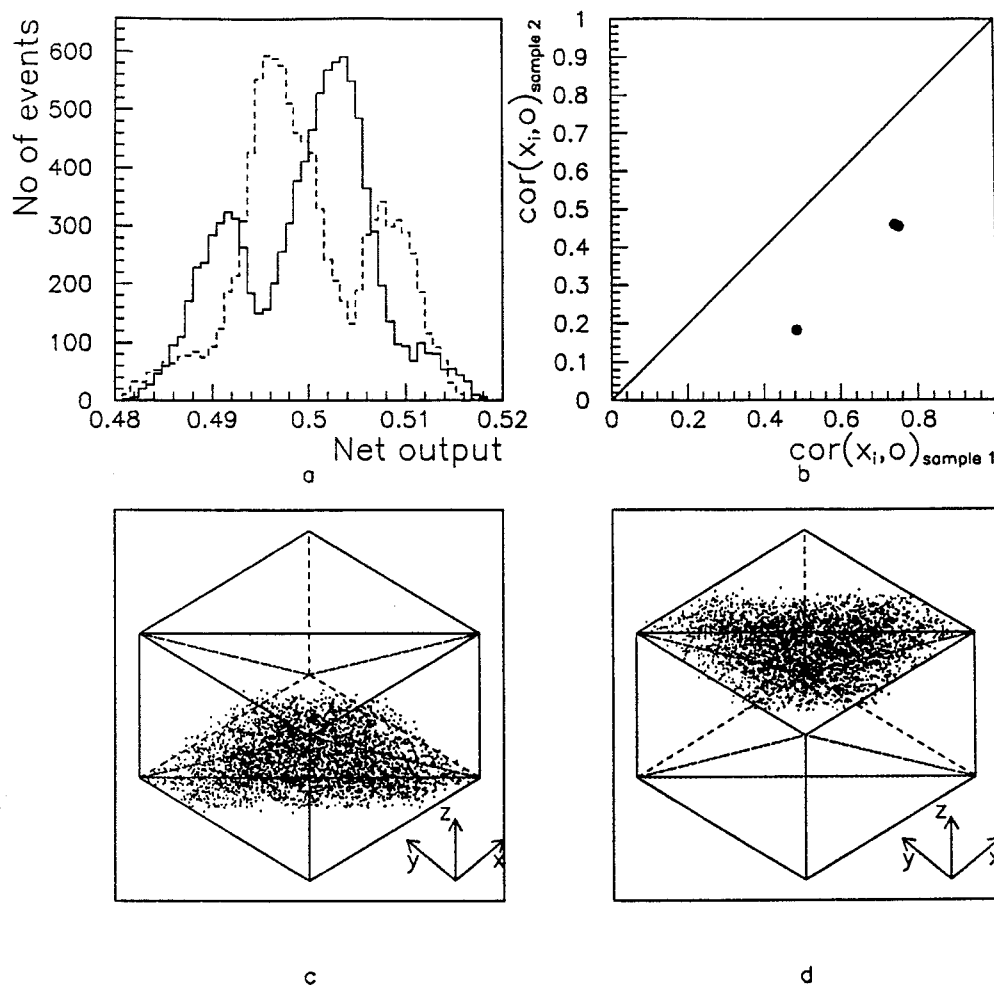


Figure 2.15: a) Output neuron distribution (full line corresponds to sample 1 and dashed line to sample 2). b) Correlation between the output neuron and each input variables for sample 1 versus the same quantity for sample 2. c) Points in the cube which neuron output is in the interval: $[0.49, 0.495]$ d) Points in the cube which neuron output is in the interval: $[0.505, 0.51]$.

disagreement are located at $\kappa \approx 0.495$ and $\kappa \approx 0.505$. Fig. 2.15.c and 2.15.d show the two regions in the input space where the output of the net is located around these two values respectively. This reflects the regions where the two distributions are more incompatible as expected from Fig 2.14.a and 2.14.b.

2.4.5 High Energy Physics example

With this example we want to show the ability of our method to find *new physics* in High Energy Physics experimental data, through the identification of the characteristics of the non simulated *new events* in the MC.

We start from a set of generated events that contains mostly τ leptons selected from a sample containing all possible final states in e^+e^- collisions at the Z^0 resonance. The generation includes the simulation of a detector with a tracking device, an electromagnetic calorimeter with two layers (ELECC) and a hadronic calorimeter (HADRC). The resolution for these devices is typical of detectors taking data at the LEP collider[26],[27],[28], [29]. The τ selection is made accepting events with low multiplicity (up to six tracks), with two back-to-back collimated jets and some missing energy (due to the unobserved neutrinos). The resulting sample also contains some background ($\approx 3\%$ of the total) coming from low multiplicity hadron and Bhabha (electrons pairs) events with photon conversions principally.

As *real* data we take a sample built in the above fashion using the world-average branching fractions [30] for the τ decays. For the *Monte Carlo* we modify two branching fractions: τ 's decaying into $\pi\pi^0\pi^0\pi^0$ are eliminated from the sample, and the number of τ 's decaying into $\pi\pi^0\pi^0$ is raised in order to maintain the same proportions for the other decay channels as in the *real data*. In this way the real data contains a *new* decay channel ($\pi\pi^0\pi^0\pi^0$) that is not simulated in the MC.

The variables used here to discriminate between τ decay channels are presented in Table 2.1. If we look at the one-dimensional projections for each variable, *real data* and *Monte Carlo* are compatible, as shown by the Kolmogorov test given in Table 2.1. Only a method capable to look in the multidimensional space of the input variables could detect the *new physics*. In this case our neural network approach can be useful. Using a 13-5-1 network, after the training step, the output distributions for the two samples are incompatible giving a Kolmogorov test probability of 0.004. This study has been repeated several times with different samples and we have obtained a quasi uniform distribution of the Kolmogorov test for the input variables (showing their compatibility) and a value never bigger than 1% for the net output.

The correlation study shows that the incompatibility of the net output comes mainly from the variable subspace spanned by the number of neutral calorimetric objects and their energy (as expected from the difference in the number of neutral pions in the *new* decay), and the number of charged tracks (this can be explained from the different number of conversions $\pi^0 \rightarrow \gamma\gamma$, $\gamma \rightarrow e^+e^-$).

This kind of analysis can also be used to detect unsimulated background, to tune detector effects or event generator parameters, and anything else that involves comparison of data and MC.

Table 2.1: Input variables. The charged track with the highest momentum corresponds to the particle p_1 .

Variable Description	Kolmogorov C.L.
Number of charged tracks	0.237
Momentum of p_1	0.339
Transverse momentum	0.723
Ionization for p_1	0.834
Energy in the first layer of ELECC	0.104
Energy in the second layer of ELECC	0.758
Number of clusters in the ELECC associated to p_1	0.754
Energy for the most energetic of the above clusters	0.688
Number of clusters in the HADRC associated to p_1	0.999
Energy for the most energetic of the above clusters	0.987
Number of clusters not associated to a charged track	0.435
Energy for the most energetic of the above clusters in ELECC	0.664
Energy for the most energetic of the above clusters in HADRC	0.823
Output neuron	0.004

Chapter 3

Unsupervised learning

3.1 Introduction

A very common problem in many areas of science is the classification of objects in clusters. It has been said many times that the human eye is the best pattern recognizer when the objects are represented by only one, two or three variables. If this is not the case, one of the strategies is to reduce the number of original variables by some transformation to 1, 2 or 3, giving a complementary view of the input data set.

To avoid confusion we would like to write a few words in order to explain the difference between clustering, encoding and dimension reduction of multidimensional data. In the first two we try to “compress” the original data in such a way that it will be possible to recover the maximum information from the compressed data afterwards, losing some of the original information on the final results. On the other hand the dimension reduction works with redundant data and the problem is to define a computational mapping between locations on an n -dimensional input space, and locations on an m -dimensional constraint surface embedded in the n -dimensional input space without any loss of information; in this way, abstraction and simplification in the description of data is reached [31].

The difference between encoding and clustering can be found in the requirements over the functional form of the transformation from the input data into the compressed one. For clustering we wish to maintain the general structure of the input data, that is, the aim is to preserve the original topology of the data set, i.e., points that are close/far away in the input have to be close/far away in the compressed data. These requirements are necessary if one wants to extract visual conclusions from the compressed data, but are not necessary in the encoder case. In fact the best encoder is such that the compressed data has a uniform distribution (max. information).

One method used most frequently in clustering and dimension reduction is the “Principal component analysis (PCA)” [32] where the reduced dimensions are linear combinations of the original variables. Here we want to present an alternative, based on Neural Nets, which is not restricted to linear combinations, and therefore, permit more complicated transformations between the input distribution and the projected one (NNA).

3.2 The method

In this section a brief outline of the NNA method is presented. The method is based on the technique of Self-Supervised Backpropagation (SSBP) also known as the “encoder” problem [33]. This algorithm is implemented on a neural network that has n units in the input layer activated with the quantities of the n -dimensional distribution under study, n units in the output layer which are forced to match the activation, one by one, of the input layer units; and one hidden layer, that contain only 1,2 or 3 units whose activation is going to be interpreted as the final projection (from here on we will call this units the “neck units”). An example of this architecture can be found in figure 3.1.

As discussed in Chapter 1, a neural network provides a mapping f from a set of inputs $\vec{in}^{(p)}$ to a set of desired outputs $\vec{out}^{(p)}$, where f is learnt (approximated) from a set of $(\vec{in}^{(p)}, \vec{out}^{(p)})$ pairs contained in the training sample. This function f is required to minimize the error function defined in Eq. 2.2. In the SSBP $\vec{out}^{(p)} = \vec{in}^{(p)}$, so the error function is the euclidean distance between an input point $\vec{in}^{(p)}$ and its output $f(\vec{in}^{(p)})$, $E = \sum_p (\vec{in}^{(p)} - f(\vec{in}^{(p)}))^2$. The function f can be separated into two functions $f1$ and $f2$, of several weights parameters (w_{ij}), such

$$\text{INPUT} \in R^n \xrightarrow{f1} \text{COMPRESS} \in R^m \xrightarrow{f2} \text{OUTPUT} \in R^n$$

where $m < n$.

The first function $f1$, transforms the input data with a certain dimension into the compressed data, the neck units with a lower dimension. The second function, $f2$, transforms the compressed data into the output data, which has the same dimension as the input. The network maps the input to a m -dimensional manifold embedded in the n -dimensional input or output space, in such a way that all the output points will be in it. The result of the error minimization is that the manifold tries to be as close as possible, depending on the flexibility given to $f2$, to the input points. The other function, $f1$, will project an input point to another one on the $f2$ surface as close as possible to it, depending again on the flexibility given to $f1$.

If the input data has redundancy then the problem is to define a computational mapping between locations on the n -dimensional input space and locations on the

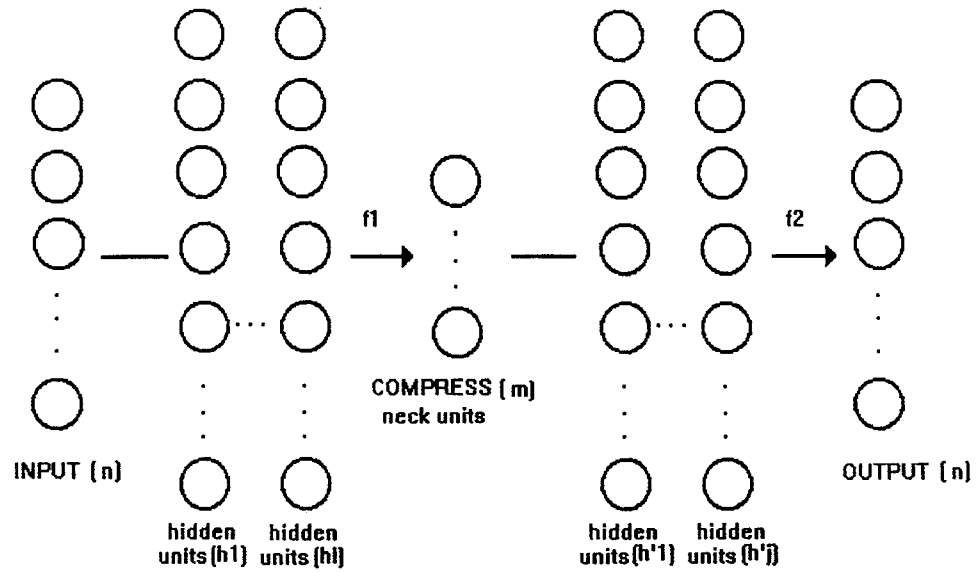


Figure 3.1: Network topology for the encoder task

m -dimensional constraint surface embedded in the n -dimensional space. We can interpret the function f_1 as a feature extraction function and then, a dimension reduction task is made by the NNA. On the other hand, unless the surface function f_2 makes loops, this projecting method from the input data to the neck units can also be useful to show cluster structures, as we show in an artificial example.

Sanger [33] showed that SSBP with only one hidden layer and a linear response function for the neurons, is equivalent to PCA (in this case f_1 and f_2 are linear). Saund [31] described an approach for performing the dimension reduction task. The

method also use a neural network with 3 layers but the sigmoidal function, for the neuron response, is adopted. Sanger shows that the assumption of linearity need not be made about the underlying constraint surface (now f_1 and f_2 are not restricted to be linear), though the method fails when the constraint surface doubles back on itself sharply.

The NNA method, proposed in Oja [34], consists of the most general case in which we add two new hidden layers, one of them between the input layer and the neck units, and the other between neck units and the output layer (Fig. 3.1). This architecture gives maximum flexibility to the functions f_1 and f_2 , and as we show in the next section, NNA is able to map complicated nonlinear surfaces.

3.3 Examples and results

First a strongly nonlinear example is considered. A two-dimensional input space is taken, and the input points are concentrated on the circle

$$(x = 0.5 + (0.5 - n) \cos(\theta), y = 0.5 + (0.5 - n) \sin(\theta)) \quad (3.1)$$

where n is an added random factor in order to simulate real noise (Fig. 3.2, top). The input space can then be parameterized with only one parameter, the angle θ . It is fairly evident that a linear reduction process as PCA is not able to solve the circle problem presented here. A three layer neural network with one neck unit has the problem of forming the mapping $f_1(x, y) = \theta$ from the input layer to the neck unit, and the mapping $f_2(\theta) = (0.5 + (0.5 - n) \cos(\theta), 0.5 + (0.5 - n) \sin(\theta))$, and it can be shown [35] that this is not possible, in spite of considering any general neural activation function. However, as both functions, f_1 and f_2 , are continuous a 5 layer neural network similar to Fig. 3.2 is theoretically able to do the job. Fig. 3.3 shows the results yielded by a 2:5:1:5:2 neural network using a sigmoid function for the neuron response, after the MSBP was applied. On the top we can see the learned mapping f_2 (a one-dimensional surface, the continuous line) superposed to the input two-dimensional points, whereas on the bottom the activation of the neck unit versus the angle θ is plotted, showing that the neural network has indeed found the function f_1 . The fact that the NNA method can solve nonlinear problems is claimed.

To show the improvement of the NNA method, respect to PCA, detecting cluster data structures, we have chosen, as a test example, the problem of distinguishing four gaussian distributions contained in the two-dimensional input space (Fig. 3.4, top), when a projection into 1 dimension is made. Figure 3.4 (bottom) shows the PCA result obtained by the perpendicular projection of the input point distributions over the continuous straight line of Fig. 3.4 top. This line corresponds of the new axis where the projected points have maximum variance. As we can see in Fig. 3.4 bottom, the PCA method only found 3 clusters, because gaussian 1 and 3 are superposed. We perform the same projection with the NNA method using a 2:5:1:5:2 neural network with a sigmoid function for the neuron response. Fig. 3.4 bottom shows the activation of the neck unit, after the MSBP was done. The 4 peaks of this figure show that the 4 gaussian distribution have been separated. Translating the activation of the neck unit into the input space (continuous curve line in Fig. 3.4 up), once again the nonlinear character of the projection is showed.

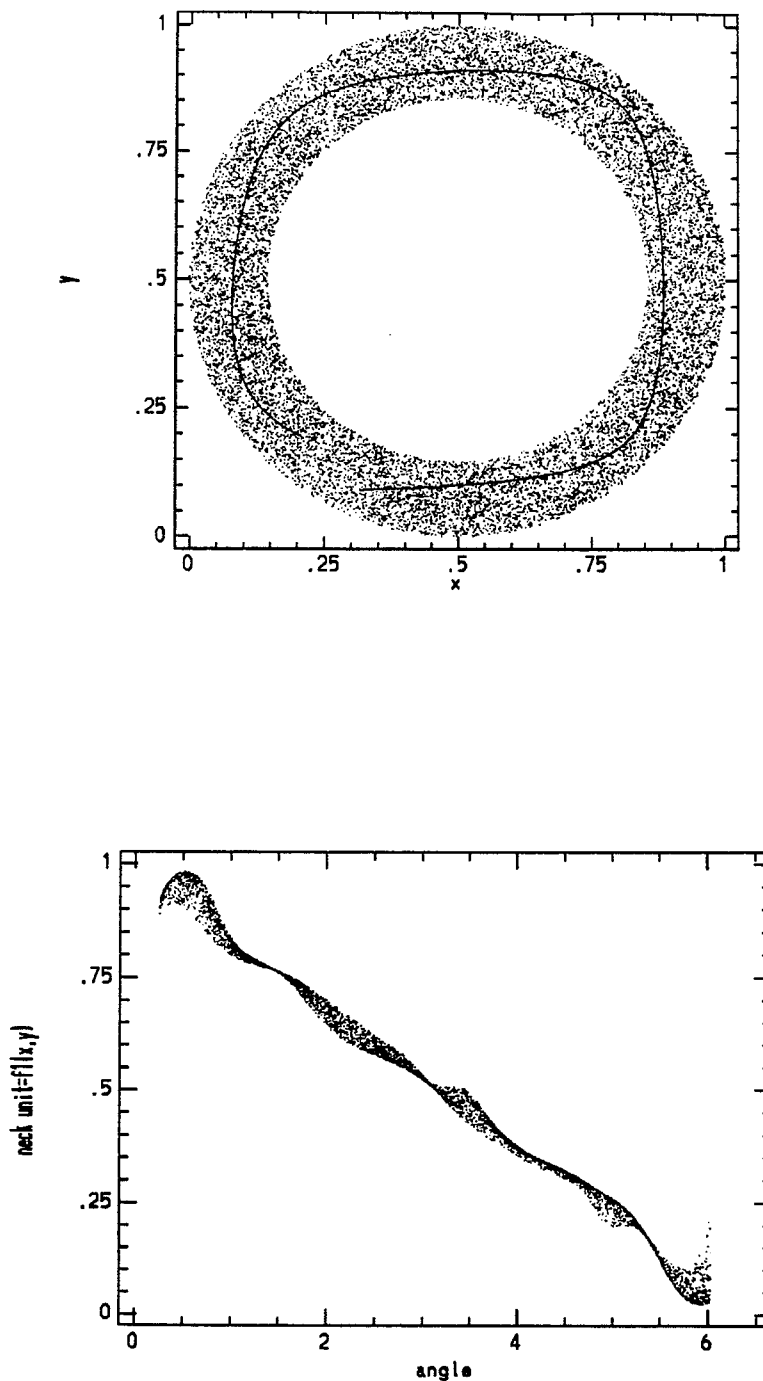


Figure 3.2: Top: we show the learned mapping f_2 (a one-dimensional surface, the continuous line), superposed to the input two-dimensional points, yielded by a 2:5:1:5:2 neural network using a sigmoid function for the neuron response, after the MSBP was applied. Bottom: the activation of the neck unit versus the angle θ is plotted.

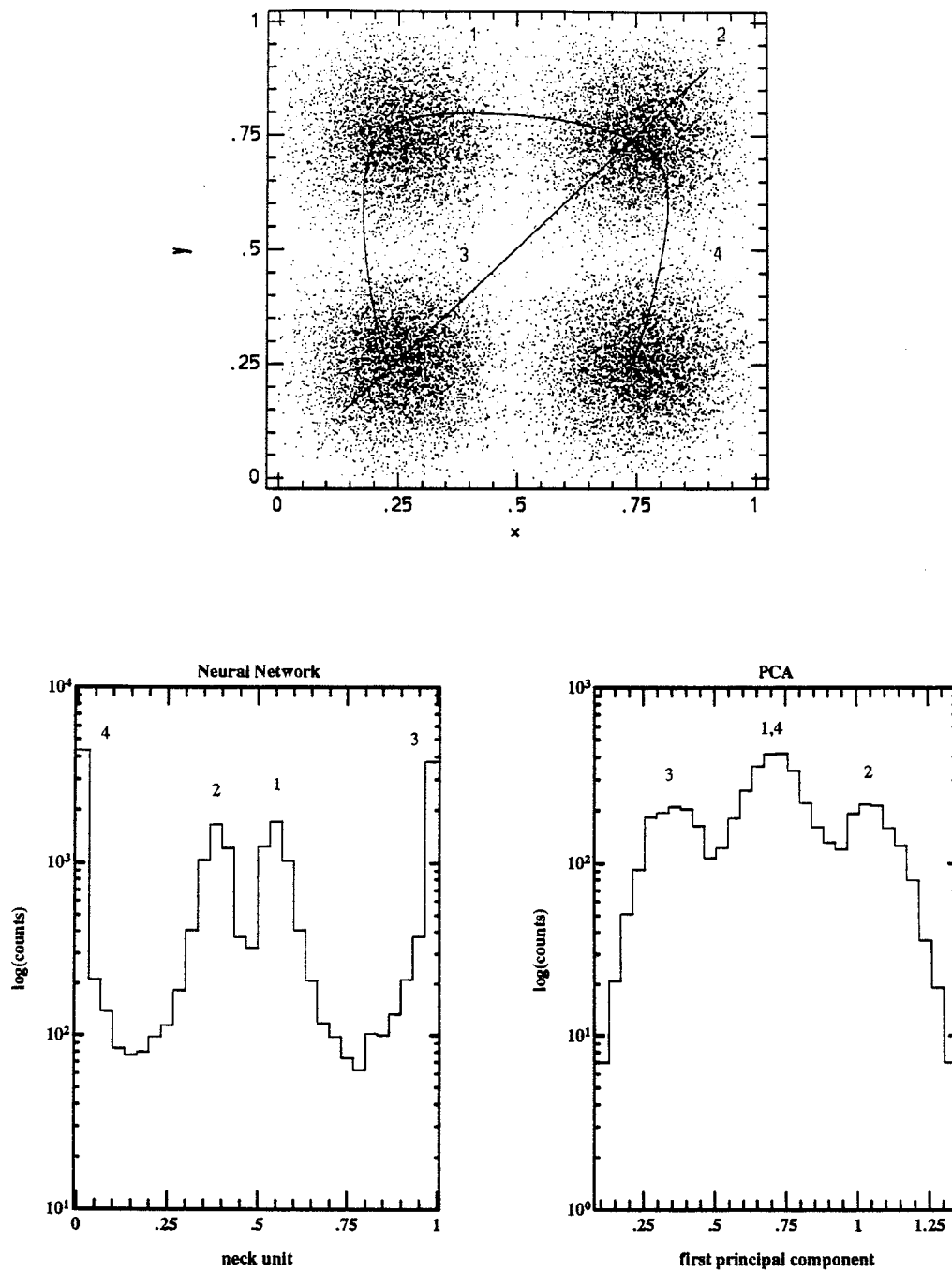


Figure 3.3: Results yielded by the PCA and the NNA (bottom) when we attempt to separate 4 Gaussian distribution contained in the two-dimensional input space (top), when a projection onto one-dimension is made.

Chapter 4

Hopfield Networks

4.1 Neural nets with constraints

4.1.1 General considerations

Fully connected single-layer networks (fig 4.1) are similar to spin-glasses [35] where each unit can be in one of two possible spin states $S_i = \pm 1$ (spin up or down) and where each spin interacts with the other spins. We can characterize the state of neurons in the same way, where $S = 1$ means activation and $S = -1$ deactivation. The connection strength (weight) between two neurons, denoted by T_{ij} for the pair of neurons i and j , can take both positive and negative values, and is in most of the models symmetric ($T_{ij} = T_{ji}$) with vanishing diagonal elements ($T_{ii} = 0$).

The dynamics of the system is given by

$$S_i = \text{sign}(\sum_j T_{ij} S_j). \quad (4.1)$$

It is easy to show [36] that this local update rule evolves the system into a local minimum of the energy function

$$E(S) = -\frac{1}{2} \sum_{ij} T_{ij} S_i S_j \quad (4.2)$$

if T_{ij} is symmetric.

4.2 MFT

In most of the applications we are not interested in local minima but in the global minimum. One way to reach the global minimum is the introduction of thermal

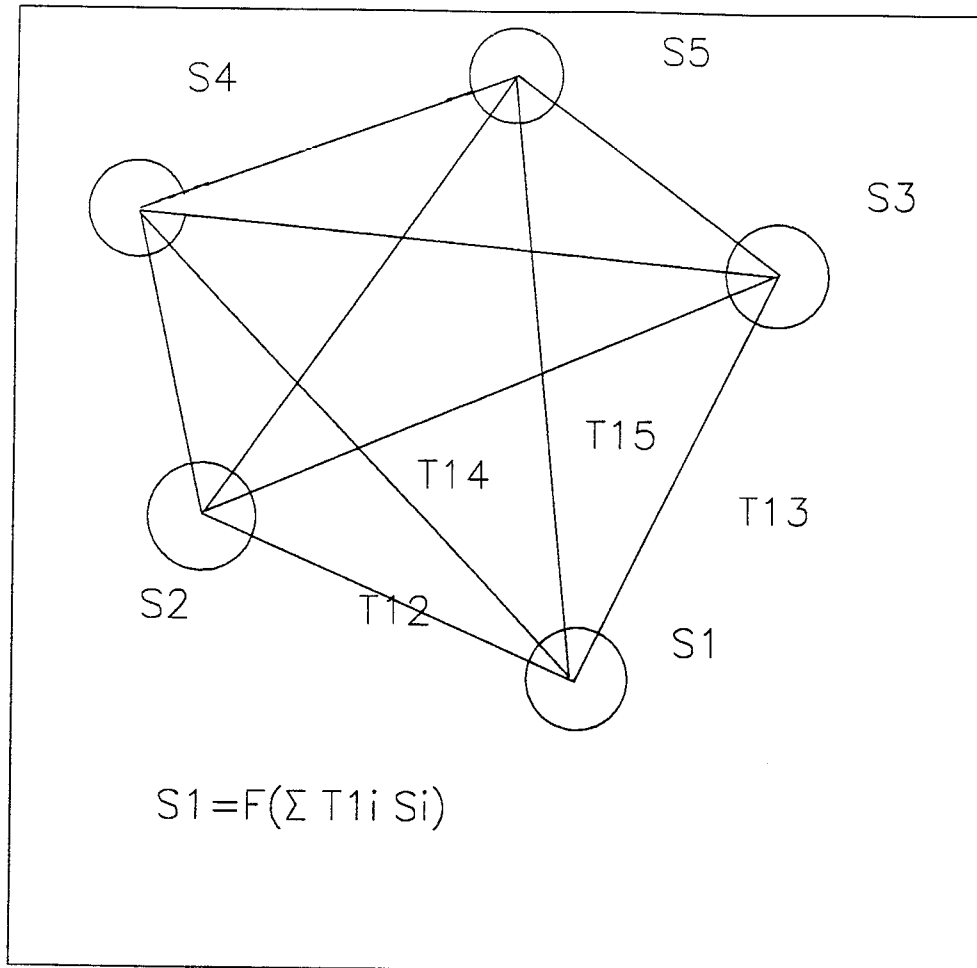


Figure 4.1: Network topology for the Hopfield model.

noise to allow the system to leave a local minimum. In this case the state vector S is Boltzmann distributed with the probability

$$P(S) = \frac{1}{Z} e^{-E(S)/T}. \quad (4.3)$$

The partition function

$$Z = \sum_S e^{-E(S)/T} \quad (4.4)$$

is the sum over all possible states and T is the temperature.

We can calculate Z with the mean field theory (MFT) approximation (See Appendix D) and get for the thermal average of S_i

$$V_i \equiv \langle S_i \rangle_T = \tanh\left(-\frac{1}{T} \frac{\partial E}{\partial V_i}\right) \quad (4.5)$$

where the energy is defined in analogy to equation (4.2) as

$$E = -\frac{1}{2} \sum_{ij} T_{ij} V_i V_j \quad (4.6)$$

The global minimum can be found by iterative calculation of the activation V_i . We will use this approximation in the following.

V_i can take any value between -1 and 1 . For many applications it is more convenient to limit the range to the interval $[0, 1]$ in order to get no contribution from deactivated neurons. In this case we make the linear transformation

$$V_i = \frac{1}{2} \left[1 + \tanh\left(-\frac{1}{T} \frac{\partial E}{\partial V_i}\right) \right] \quad (4.7)$$

This expression is equivalent to the 'logistic function'

$$V_i = \frac{1}{1 + e^{-X}} \quad (4.8)$$

with

$$X = -\frac{2}{T} \frac{\partial E}{\partial V_i} \quad (4.9)$$

We can generalize the energy function in order to satisfy some requirements on the network activation by imposing constraints in the form

$$E = -\frac{1}{2} \left(\sum_{ij} (T_{ij} - \alpha C_{ij}) V_i V_j - \beta D \right) \quad (4.10)$$

C is a constraint matrix, D a global constraint for all activation, and α and β are Lagrangian multiplier. Positive weights enforce the activation whereas the two constraint terms weaken it (therefore they are also called penalty terms).

4.3 Breaking NP-hard problems with Hopfield networks.

An application of the model presented deals with discrete systems where there is a large but finite set of solutions to certain optimization problem. Typically if the problem is of "size" N , then there are of the order of e^N or $n!$ possible solutions, of

which we want the one that minimizes the cost function. These kind of problems are often referred as combinatorial optimization problems.

Optimization problems can be divided into classes according to the time it takes to solve them. If there exist an algorithm that solves the problem in a time that grows only polynomially (or slower) with the size N of the problem, then it is said to be polynomial and belongs to the class P . P is a subclass of another class called NP (non-deterministic polynomial). NP problems are those for which one can test in polynomial time whether any "guess" of the solution is right. An important subclass of NP is that of the NP -complete problems. They are the hardest NP problems and are loosely characterized as follows: if one could find a deterministic algorithm that solves one NP -complete problem in polynomial time, then all other NP problems could be solved in polynomial time. In that case P, NP and NP -complete would all be the same class, but is most likely (though not proven) that $P \neq NP$. The probable situation is shown in fig 4.2.

Empirically the time it takes to solve an NP -complete problem tends to scale exponentially with the size N . Hopfield-MFT networks are good candidates for solving NP -complete problems. The feeling is that they are able to find sub-optimal solutions to combinatorial optimization problems in polynomial time. For several problems a sub-optimal solution would be sufficient. This approach has been applied successfully to k -partitioning a graph [38], the Traveling Salesmen Problem [37] and image segmentation. As we will see later this network model performs satisfactorily in the track finding task.

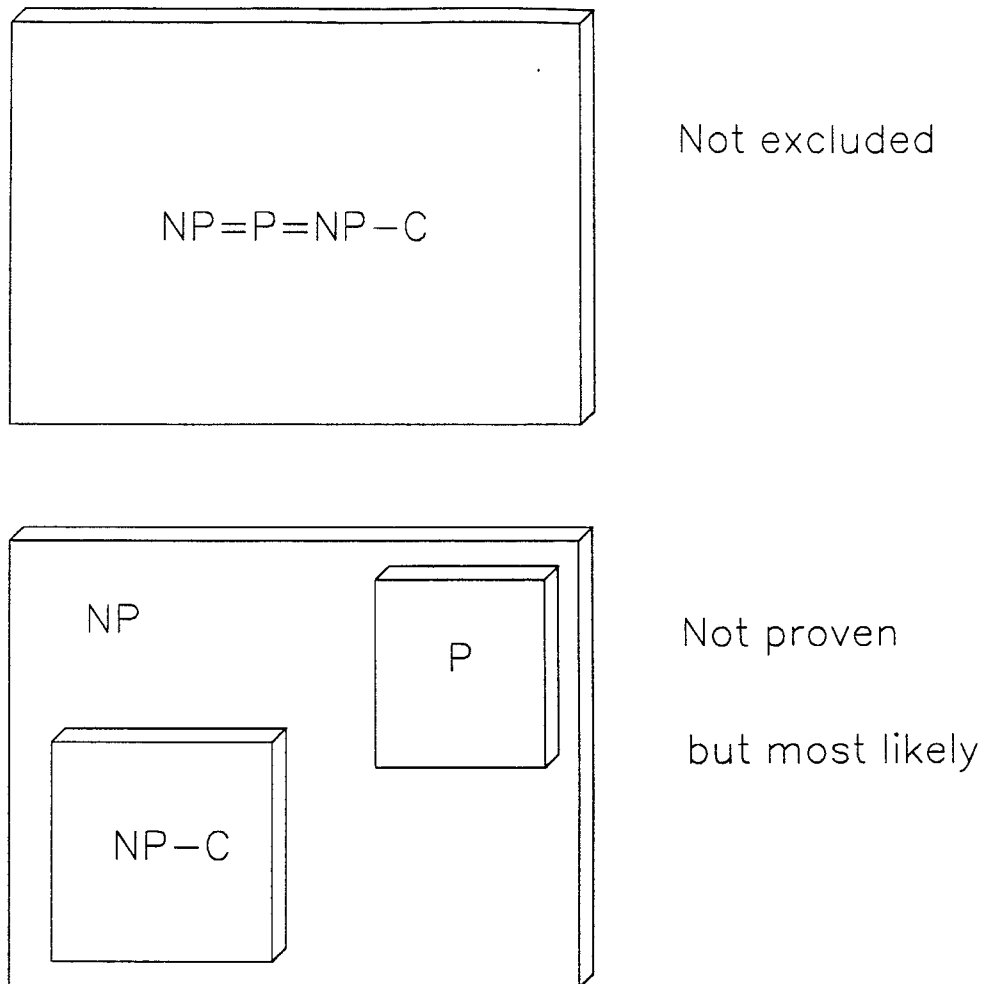


Figure 4.2: Two possibilities on the problem complexity classification: In the first case if a NP problem is solved in polynomial time, then there exists a polynomial algorithm for any problem in the NP class. Although this cannot be excluded, the most likely situation is shown in bottom.

Chapter 5

Data analysis

5.1 Introduction

The LEP machine is an e^+e^- collider working on the peak of the Z^0 resonance. The Z^0 decays provide a lot of information on the structure of the electroweak and strong interactions. Several measurements have been done that fulfill all the parameter spectra of the Standard Model available at 90 GeV [39] with good agreement up to the experimental errors. At this moment the main goal is to use the high statistics accumulated in order to reduce still more the errors (including systematic) on the measured parameters.

One interesting observable is the polarization of the τ pairs produced from the Z^0 decay. This is an indirect measurement, its interpretation depends strongly on the tau decay details. For this reason it can be useful to use a suitable multidimensional approach in order to extract the maximum information from the experimental data.

Neural Networks seem to be a reasonable approach to this problem. In the next sections the τ branching ratios and polarization will be measured from the data taken with the ALEPH detector in 1992.

5.2 ALEPH

The ALEPH detector is described in detail elsewhere [26]. The main subcomponents relevant for the analysis are:

- A microvertex device (VDET) for very accurate vertex tagging of tracks coming from the interaction point. Located just around the beam pipe, it gives three dimensional points for tracks that cross it ($\sigma_{r\phi} = 13\mu m$, $\sigma_z = 20\mu m$).

- The Luminosity Calorimeter (LCAL). It is built with the same technique as the Electromagnetic Calorimeter, and it is used to detect low angle Bhabha events in order to measure the luminosity accumulated by the detector.
- The inner tracking chamber (ITC), a cylindrical 8-layer axial-wire drift chamber, extending from 13 to 29 cm radius.
- The time projection chamber (TPC) , extending from 0.4 m to 1.8 m in radius, which combined with the preceding detector provides a resolution of $\delta p/p^2 = 1.2 \times 10^{-3} \text{GeV}^{-1}$. The end-plates are divided into 21 pad-rows, the radius of the first pad-row is about 40 cm and the radius of the last pad-row about 180 cm. The radial distance between two pad-rows is roughly 6 cm. Each half of the TPC has a lateral distance (Z) of 220 cm. The coordinates are measured with a resolution of about 200 μm in XY and of about 2 mm in Z.
- The electromagnetic calorimeter (ECAL), a lead sheet/wire chamber sandwich of 45 layers, and a fine grained (16 $\times$ 16 mrad) projective tower cathode pad readout segmented in three stacks of 4, 9 and 9 radiation lengths in depth, having an energy resolution of $18.7\%/\sqrt{E} \oplus 2.1\%$, and with an independent low-noise readout of each of the 45 sense wire planes in a module.
- The hadronic calorimeter (HCAL), consisting of 23 layers of streamer tubes interleaved in the magnetic iron return yoke, with the individual tube hits recorded digitally and a projective tower cathode pad readout of hadronic energy.

ITC, TPC and ECAL lie all inside the superconducting solenoid, which provides a 1.5 Tesla magnetic field.

The study of τ decays benefits from the main features of the detector: 3-dimensional track separation in the TPC, high granularity of the electromagnetic calorimeter, and the HCAL digital pattern. This provides optimal particle separation and identification, crucial for the polarization analysis.

5.3 Tau Physics

5.3.1 Introduction

The polarization of the taus observed at LEP results from parity violation in Z^0 production and decay. The inequality of the Z couplings to the left-handed ($g_V^l + g_A^l$) and right handed ($g_V^l - g_A^l$) leptons induces a polarization of the Z produced in collisions between unpolarised e^+ and e^- beams. Similarly this inequality induces a non-zero polarization of the taus produced in the decay of the Z which is dependent

on the angle between the Z polarization vector, along the beam, and the tau line of flight. The polarization measurement gives information about the couplings of the Z to taus and electrons and gives a good measurement of the effective weak mixing angle [22].

The normal procedure to extract the polarization from τ decays is to work separately for each τ decay channel and finally average the results in order to give a global number. This procedure has several limitations:

- The event must be divided into two hemispheres to perform the channel separation. This implies a loss of the information provided by the helicity correlation for the two event sides.
- The channel to channel background is a source of systematic error. In fact, to estimate the background for a given channel it is necessary to assume a polarization and a set of branching ratios for the Monte Carlo. Efficiencies and background for each channel selection depends on the Monte Carlo polarization and branching ratios.
- In order to get a reliable polarization result in each channel it is necessary to make cuts in order to raise the sample purity. This results in a loss of statistics.

Because of these facts, a more coherent approach is to measure simultaneously the decay channel and the helicity for each $\tau\tau$ event. This is not an easy task mainly due to the different correlation between the kinematical observable for each channel and its polarization. Standard analysis methods have a very hard time solving this kind of problems, while NN are well suited for this multidimensional and multi-class data analysis.

In the next section we will start by presenting a method to extract the polarization from a sample of τ decaying into ρ . The method can be applied in the same way to the other single channel. Later we will present a simultaneous measurement of the branching ratios and polarization using the complete sample of τ , without the need of selecting each channel separately. For this study we will use a set of event shape variables and estimators for particle type. From the results it will be seen that one of the most important systematic effects for the NN method is the accuracy of the Monte Carlo simulation. Finally, using a more reliable set of variables, results for τ branching ratios and polarization are presented and compared with other measurements.

5.3.2 A new method to determine the τ polarization for a single channel

In the process $e^-e^+ \rightarrow \tau^-\tau^+$ the polarization of the τ is defined as :

$$Pol = \frac{N^+ - N^-}{N^+ + N^-} \quad (5.1)$$

where $N^+(N^-)$ is the number of τ^- produced with positive (negative) helicity. Here, we restrict ourselves to events where one of the τ decays into $\rho + \nu$. The NN technique developed in the section 2.2.1 (see that section for details about the network topology and input variables) is able to find the helicity of the $\tau \rightarrow \rho + \nu$, with high efficiency; therefore it should be easy to determine the polarization.

The matrix deconvolution method will be applied. Knowing the probability for the net to find the correct helicity of the τ , the number of events observed with an output neuron lower (N_1) or higher (N_2) than the chosen cut can be written as:

$$N_1 = N^+ P_1^+ + N^- P_1^- \quad (5.2)$$

$$N_2 = N^+ P_2^+ + N^- P_2^- \quad (5.3)$$

where $P_1^-(P_2^-)$ is the probability that a τ^- with negative helicity gives an output lower (higher) than the chosen cut. Equivalent definitions are valid for P_i^+ . Obviously $P_1^- + P_2^- = 1$ and $P_1^+ + P_2^+ = 1$. These probabilities depend on the proportions (α_i) used in the training but this does not introduce any bias on this analysis because the test sample is treated with the same $n(\vec{x})$ as $P_+(\vec{x})$ and $P_-(\vec{x})$. Using equation 5.1, 5.2 and 5.3 we obtain:

$$Pol = \frac{N_1 - N_2}{N_1 + N_2} \frac{1}{P_1^+ - P_1^-} + \frac{1 - P_1^+ - P_1^-}{P_1^+ - P_1^-}. \quad (5.4)$$

This equation is similar to the well-known forward-backward asymmetry but rescaled by a factor $1/(P_1^+ - P_1^-)$ and shifted by $(1 - P_1^+ - P_1^-)/(P_1^+ - P_1^-)$. The last equation shows that the polarization is a function of the statistics of the data (N_1, N_2) and of the precision of the Montecarlo predictions ($P_i^\pm$). The error on the polarization can be obtained by error propagation applied to equation 5.4.

We tested our method for finding the tau polarization in various samples of Monte Carlo τ 's using the same trained net. Looking at the measured polarization obtained from the simulated data sample as a function of the generated polarization we observe an exact correlation without any bias. The error in the polarization coming from the data (10,000 events in total) and from the errors on $P_i^\pm$ (we have use 50,000 Monte Carlo events for its determination) is 0.05 independently of the polarization generated for the test sample.

This method was applied to a sample of real τ decaying into ρ from the 1990 ALEPH data using the selection criteria in [23]. The inputs given to the network

are the charged and neutral energy (From this variables it will be possible to reconstruct the decay angles defined in chapter 2 up to the detector resolution). Fig 5.1 shows the distribution for the two input variables for data and Monte Carlo and the resulting neuron output. The number obtained (statistical error only) is

$$P = -0.22 \pm 0.10$$

which was in good agreement with measurement performed at that moment (statistical plus systematic are quoted) [40, 41]

$$P = -0.22 \pm 0.12 \pm 0.15$$

The method shown here can be generalized in the same way for all the one prong events of the τ decay.

5.3.3 Simultaneous measurement of the τ polarization and branching ratios with neural networks: Raw input approach

A neural network (NN) technique has been developed as a method to simultaneously determine the helicity and decay channel of the τ particles in the process: $e^+e^- \rightarrow (Z^0, \gamma^*) \rightarrow \tau^+\tau^- \rightarrow X$. The method takes advantage of the fact that both taus (with opposite helicities) contain polarization information; it gives efficiencies comparable to the Bayesian method (This is verified explicitly for the ρ channel in chapter 2).

The standard ALEPH tau selection is the starting point. The τ selection is made accepting events with low multiplicity (up to six tracks), with two back-to-back collimated jets and some missing energy (due to the unobserved neutrinos). This leads to an efficiency above 72% slightly dependent on the decay channel and the polarization. The resulting sample also contains some background ($\approx 3\%$ of the total) principally coming from low multiplicity hadron and Bhabha (electrons pairs) events with photon conversions.

The NN chosen consist of 13+13+3 input neurons, one hidden layer with 18 units and an output layer with 8+8+2 units (Fig 5.2) (for 8 possible decay channels in each hemisphere plus two neuron for the τ^+ polarization). Notice that this output representation corresponds to three decoupled unary representations not implying a loss in the information in the channel separation (the probability to decay in a given channel for one hemisphere is independent of the decay in the opposite hemisphere), and assumes that the helicity is independent of the channel. If we do not assume these facts the number of neurons becomes as larger as the number of classes ($8 \times 8 \times 2 = 128$), too large for practical purposes.

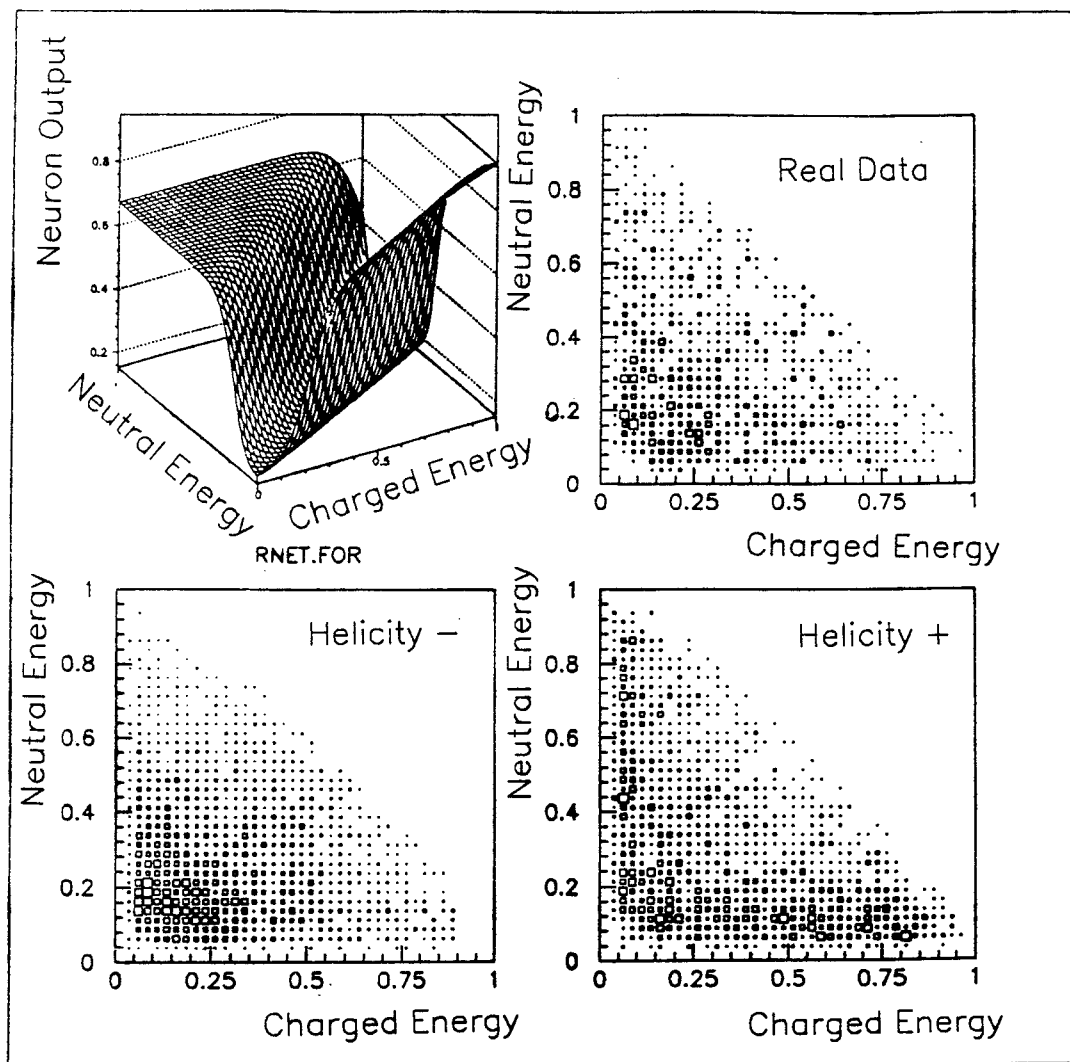


Figure 5.1: Distribution for the two input variables for data and Monte Carlo and the resulting neuron output for the τ decaying into ρ polarization analysis from the 90 ALEPH data.

The input neurons are activated by the quantities we found relevant for the classification task: the number of good charged and neutral tracks (clusters of energy in the calorimeters not associated to charged tracks) for each hemisphere, their momenta and energies, information about the identified particles (e, μ, γ) and some global event variables (invariant mass and planarity) (figure 5.3).

The network is trained to distinguish the decay channel in each hemisphere and

Network architecture:

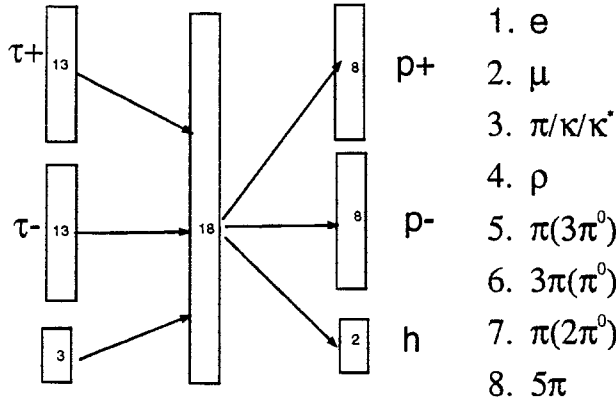


Figure 5.2: NN topologies for the polarization analysis.

the event helicity using the Monte Carlo information.

After the training (using the full detector simulation Monte Carlo), the fit method is used to extract the measurements. In this case, the number of classes is too high to perform a fit in the whole output neuron space. Instead the projection method is used (Section 2.3).

Unidimensional projections using Monte Carlo events are made to each output neuron for each couple of decay channels and τ^+ helicity. This results (by skipping one of the helicity neurons for redundancy) in a total of 17 (output neurons) $\times$ 8 (τ^+ decays) $\times$ 8 (τ^- decays) $\times$ 2 (helicities) histogram with 50 bins each for the Monte Carlo and 17 histograms for the real data (this results in a total of 850 bins to fit). These distributions are corrected for the τ selection global efficiency for each class ($\epsilon_{ij}^\pm$). The same distributions are built for the non τ background and weighted with the estimates derived from the Monte Carlo. Then the output distributions for the real data are fitted to a linear combination of the reference Monte Carlo distributions (τ 's + background), using as free parameters the τ branching ratios and the polarization:

$$\chi^2 = \sum_{k,k'=1}^{50} \sum_{l,l'=1}^{17} (R_k^l - M_k^l) C_{klk'l'}^{-1} (R_k^{l'} - M_k^{l'})$$

Tau selection and input variables:

***SELTAU**

*** Global variables:**

- Invariant mass
- Acolinearity
- $\cos(\theta^*)$

***For each hemisphere:**

- # good tracks
- P max
- Pt
- dE/dx
- Ecal energy % stack 1,2
- # clusters associated to max p track in ECAL, HCAL
- # neutral clusters ECAL, HCAL
- Energy of the most energetic clusters in each case.

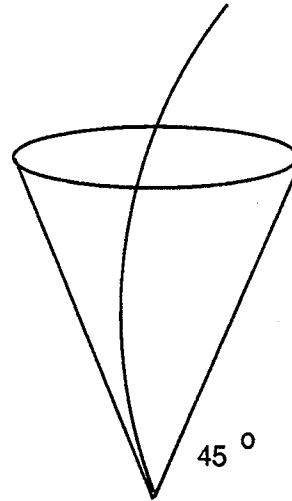


Figure 5.3: Variables used in the polarization analysis.

where R_k^l is the k -th bin of the histogram of the l output neuron for real data, $C_{klk'l'}$ is the bin-to-bin covariance matrix and

$$M_k^l = N \sum_{i=1}^8 \sum_{j=1}^8 (Br(i)Br(j) \frac{1}{2} (1+Pol) D_{k;ij}^{l+} + Br(i)Br(j) \frac{1}{2} (1-Pol) D_{k;ij}^{l-}) + \sum_{i=1}^4 Bck(i) B_{k,i}$$

with $Br(i)$ the branching ratio for the i -th channel and Pol the polarization as a free parameters and N the total number of real events. The normalized Monte Carlo reference distributions $D_{k;ij}^{l\pm}$ are computed as

Table 5.1: Input variables for the polarization analysis. The charged track with greater momentum corresponds to the particle p_1 .

Variable Description	Kolmogorov C.L.
Number of charged tracks	0.913
Momentum of p_1	0.349
Transverse momentum	0.135
Ionization for p_1	0.572
Energy in the first layer of ECAL	0.133
Energy in the second layer of ECAL	0.174
Number of clusters in the ECAL associated to p_1	0.627
Energy for the most energetic of the above clusters	0.335
Number of clusters in the HCAL associated to p_1	0.993
Energy for the most energetic of the above clusters	0.989
Number of clusters no associated to a charged track	0.437
Energy for the most energetic of the above clusters in ECAL	0.925
Energy for the most energetic of the above clusters in HCAL	0.853
Output neuron	0.025

$$D_{k;ij}^{l\pm} = \epsilon_{ij}^{\pm} \frac{N_{k;ij}^{\pm}}{\sum_{k=1}^{50} N_{k;ij}^{l\pm}}$$

where $N_{k;ij}^{l\pm}$ is the number of Monte Carlo events with $\pm$ polarization for which the τ^+ decays into channel i and the τ^- decays into channel j lying in the k -th bin of the neuron l . The $B_{k;i}$ corresponds to the background distributions (for $e, \mu, \text{hadrons}$ and two γ), where the $B_{ck}(i)$ are fixed to the expected background for each type. Finally, $\epsilon_{ij}^{\pm}$, the τ selection efficiency for events decaying into channel i and channel j with $\pm$ helicity is evaluated from the Monte Carlo.

The 850×850 bin to bin correlation matrix is computed from the real data using the formula 2.32 and inverted using the Singular Value Decomposition method (SVD) [43] due to the fact that direct inversion (LU decomposition or solving a linear equation system) has numerical accuracy problems.

When this method is applied to simulated data, the results are in perfect agreement with the parameters put into the Monte Carlo. The errors obtained outperform previous methods used for this selection. Nevertheless, the fit for real data has a very high χ^2 . This indicates a disagreement between the real data and the Monte Carlo simulation. This was the main motivation to develop the NNAC method (Section 2.2.4).

In order to study this behaviour, the NNAC method is applied to the input variables for real data and Monte Carlo data. Table 5.1 shows the KCL for each variable and the KCL for the output neuron. The KCL for the unidimensional projections does not indicate a big disagreement between data and Monte Carlo, but surprisingly, the output neuron provides a very low KCL. In fact, using these output neuron for classification is possible to separate real data and Monte Carlo with a 60% efficiency!(fig 5.4).

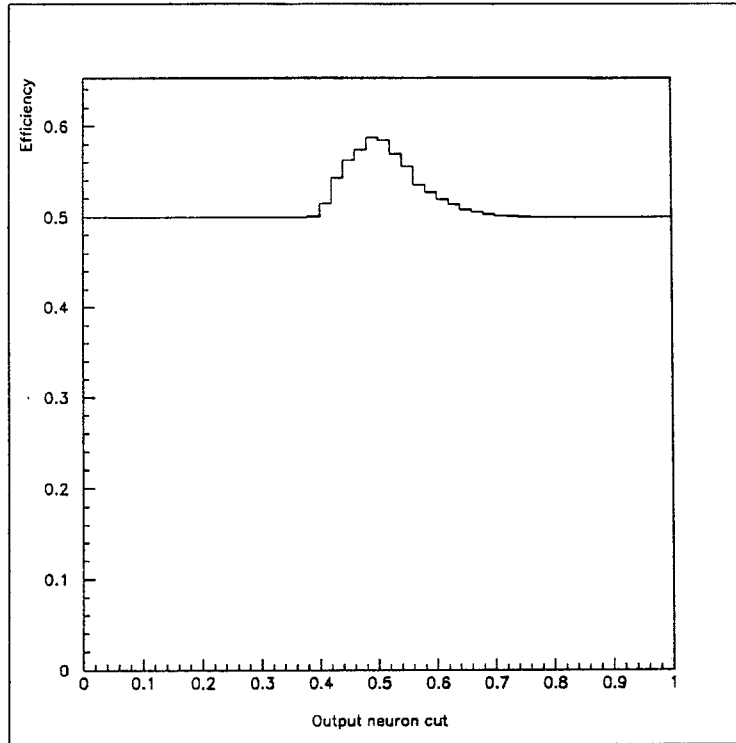


Figure 5.4: Efficiency for distinguish real and Monte Carlo data using the NNAC analysis output neuron to classify them.

The correlation plot shows that the worst-simulated variables are the number of neutral objects and their energy. A closer look reveals that the correlation between this variables is responsible for the poor results given by the NN in the real data case.

Standard analysis overcome this difficulty by making cuts in the minimal energy requirements for considering a neutral cluster as a "good" calorimetric object. On the other hand, NNs amplify the effect due to the fact that this phase space zone carries a lot of information about the event helicity and decay channel.

These results put an alert on the proper way to use NN for event selection:

good agreement for the unidimensional projections does not assure agreement in the whole space. It is extremely useful to make a NNAC analysis for the input variables first, because NN scan all the available input space to discriminate between classes. Badly simulated regions must be skipped or the Monte Carlo must be corrected to simulate better this conflictive zones.

In the next section, a second approach is followed by making use of the corrected neutral energies provided by the Energy Flow algorithm of ALEPH [42].

5.3.4 Simultaneous measurement of the τ polarization and branching ratios with neural networks: Energy Flow approach

The Energy Flow algorithm from ALEPH [42] uses the charged and neutral energy from the detector in order to have a better estimate of the particles momenta and energies.

In this case the method makes use of a new set of input variables (table 5.2).

The agreement between real data and Monte Carlo is assured by the fact that the energy flow algorithm has some adjustable parameters related to the detector simulation and event reconstruction that are fitted to the real data. The NN used for this classification has a 18-36-17 topology. The 18 input variables are shown in table V. The meaning of the output neurons is the same as in the previous section. Two methods are used to extract the measurements, the matrix an fit methods.

In the first case the 128×128 efficiency matrix E is inverted. (Fig 5.5). Normal inversion methods lead to unstable behaviour, mostly due to poor statistics for some channels. Singular Value Decomposition is found as the most robust method for the inversion. Due to the inefficient use of the information given by the matrix method (Section 2.3) the result obtained has a large statistical error (table 5.3) and no systematics errors are quoted.

More suitable results are obtained from the fit method. The same procedure used in the previous section is used to compute the projected distributions. Fig 5.6 shows the event distribution for the first three output neurons and the polarization neuron for data and the best Monte Carlo fit, and Fig 5.7 the bon-to-bin correalation matrix used for the fit.

The systematics that will be taken into account in this method come only from real and Monte Carlo data disagreement. In this case the errors are computed as follows. By comparing each unidimensional projected distributions for the NN input for Monte Carlo and real data a maximum shifting or smearing is computed. Each of the input variables of the real data is modified using these deviations and a new fit is performed from the network output. The deviation in the fitted parameters from

Table 5.2: Input variables for the polarization analysis using energy flow.

Input neuron	Variable Description	Kolmogorov C.L.
1	Number of charged tracks in hemisphere +	0.947
2	Number of charged tracks in hemisphere -	0.339
3	Number of neutral tracks in hemisphere +	0.010
4	Number of neutral tracks in hemisphere -	0.047
5	Total charged energy in hemisphere +	0.131
6	Total charged energy in hemisphere -	0.078
7	Total neutral energy in hemisphere +	0.874
8	Total neutral energy in hemisphere -	0.995
9	Number of identified μ in hemisphere +	1.000
10	Number of identified μ in hemisphere -	1.000
11	Number of identified electrons in hemisphere +	0.367
12	Number of identified electrons in hemisphere -	0.921
13	Number of identified γ in hemisphere +	0.258
14	Number of identified γ in hemisphere -	0.746
15	Planarity	0.489
16	Total momentum in hemisphere +	0.523
17	Total momentum in hemisphere -	0.534
18	Invariant mass	0.90621
-	Output neuron	0.457

the fit without deviation is computed as a systematic error. Finally all systematic errors for each parameter are added linearly.

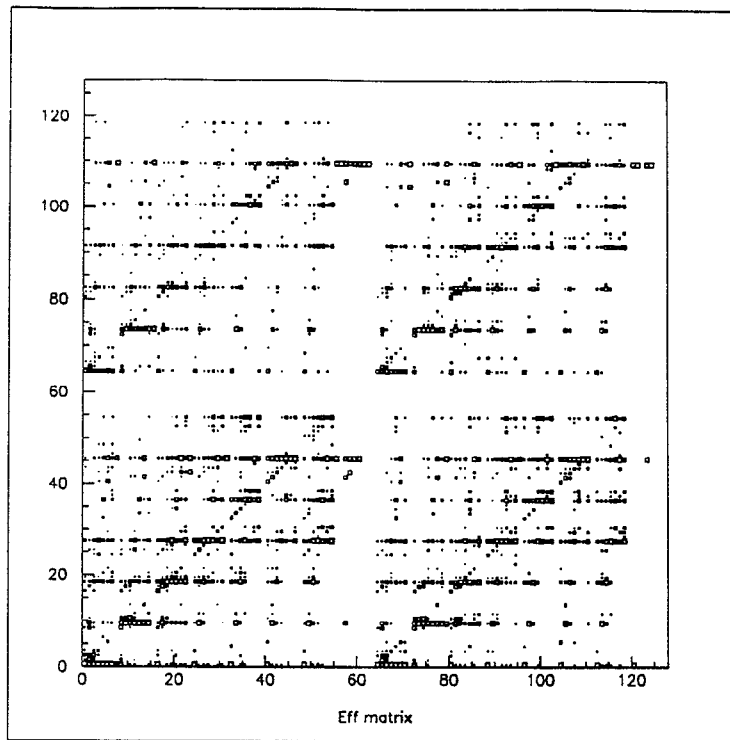


Figure 5.5: Efficiency matrix.

Table 5.3: Branching ratios and Polarization results for the 1992 data.

Channel	Matrix method (%) (stat. only)	Fit method ($\chi^2/\text{n.of.bins} = 0.912$) (stat. + sys.)
e	18.11 ± 0.74	$17.95 \pm 0.23 \pm 0.17$
μ	19.40 ± 0.81	$17.54 \pm 0.22 \pm 0.12$
π, κ, κ^*	13.66 ± 1.01	$13.46 \pm 0.27 \pm 1.06$
ρ	22.15 ± 1.06	$23.34 \pm 0.36 \pm 0.26$
$\pi + 3\pi^0$	0.57 ± 1.30	$0.36 \pm 0.24 \pm 0.31$
$3\pi(\pi^0)$	12.74 ± 3.20	$15.52 \pm 0.30 \pm 0.28$
$\pi + 2\pi^0$	11.76 ± 1.27	$9.93 \pm 0.38 \pm 0.88$
$\pi > 3$	1.16 ± 0.97	$0.93 \pm 0.19 \pm 0.15$
Polarization	-0.10 ± 0.072	$-0.129 \pm 0.017 \pm 0.029$

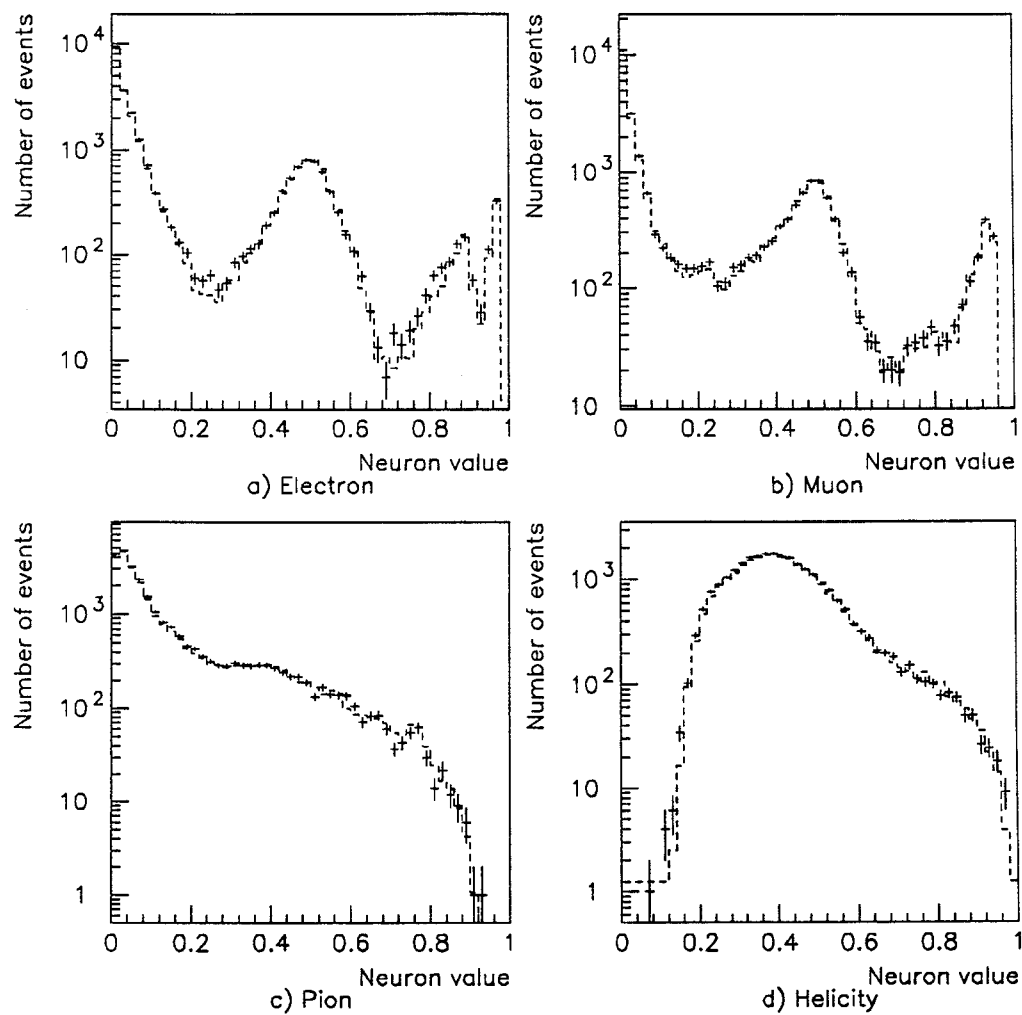


Figure 5.6: Event distribution for the first three output neurons (a,b,c) and the polarization neuron (d) for data and the best Monte Carlo fit. The dashed line corresponds to Monte Carlo, errors bars to real data

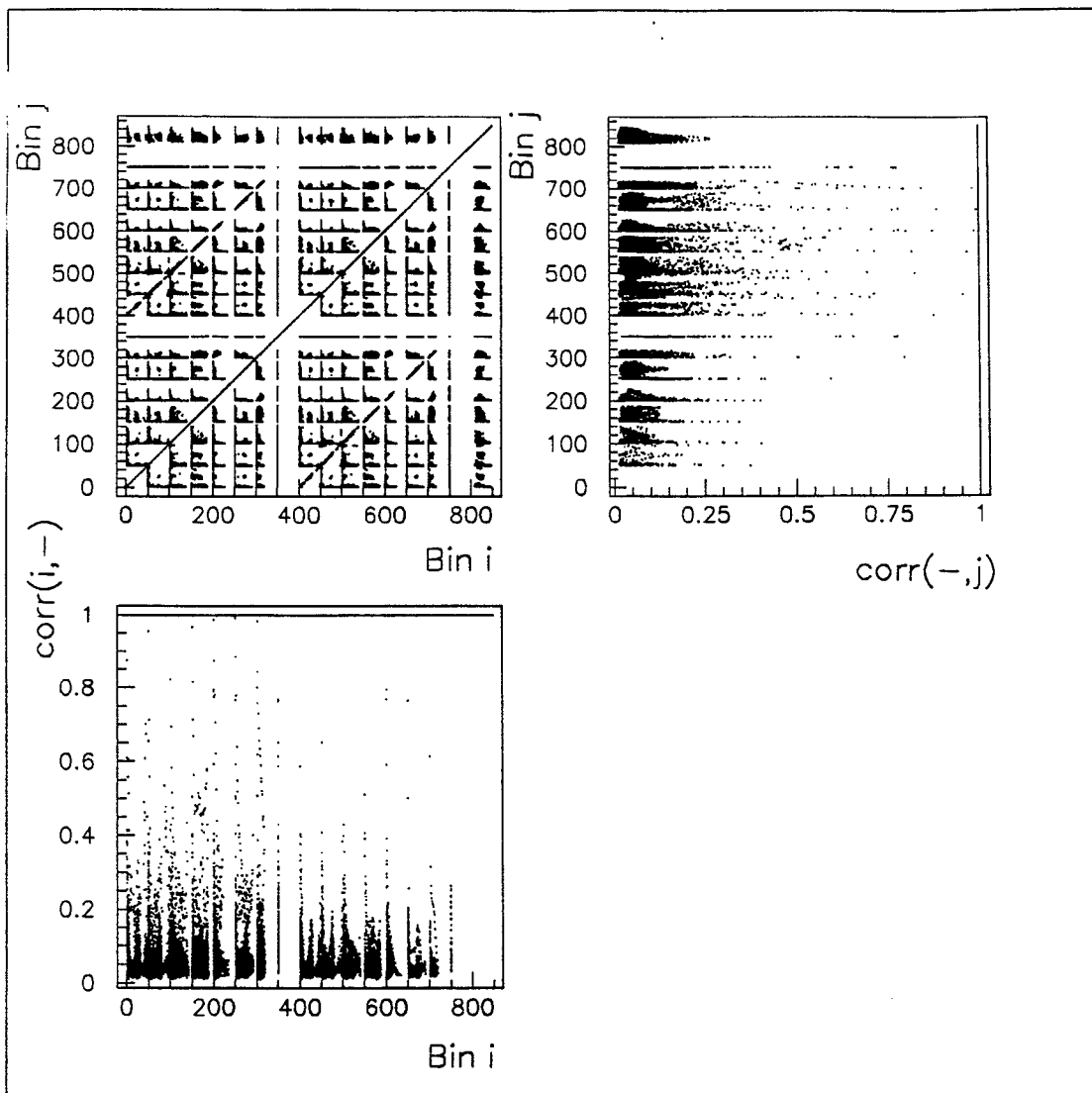


Figure 5.7: Top: Correlation matrix. Black pixels shows bin to bin correlations different from 0. Next to the X-Y axes the correlation values are shown. Top right and bottom left :correlation values of a bin with all the others .

5.4 Unsupervised τ classification

An interesting application for the encoder is to try to classify a data sample without any a priori knowledge. The idea is to perform a projection of multidimensional data in a classification space of lower dimension. In this space, the class can be separated by visual inspection. The criterion for the projection will be provided by the capability to recover the input space information from the classification space in a Mean Square Error sense. For convenience of visualization the classification space must be bidimensional. From this point of view, dimensionality reduction is used to help the data clustering.

To illustrate the power of this method, a sample of real τ decays is used. The input variables used are the same as in the previous section.

A 13-27-2-27-13 network was trained to recover the input information in the output layer. The projection of the input over the two units in the central layer reveals several clouds of points (See figures 5.8,5.9 for two different network solutions), that can be assigned to sets of events with similar characteristics. In order to identify the subsample formed, Monte Carlo data were passed through the network. Fig 5.8,5.9 are labeled with the decay class of the event found in the Monte Carlo. The result is that the network is capable of identifying the τ decays without any model. This provides a qualitative insight into the structure of the classes of a data sample, but this approach will not be useful for quantitative measurements, because a model to compute the overlap between classes is needed.

This method could be very useful in the first steps of an analysis. The important fact is that it reveals which is the natural clustering of the data, and when it is necessary to provide a model to classify them.

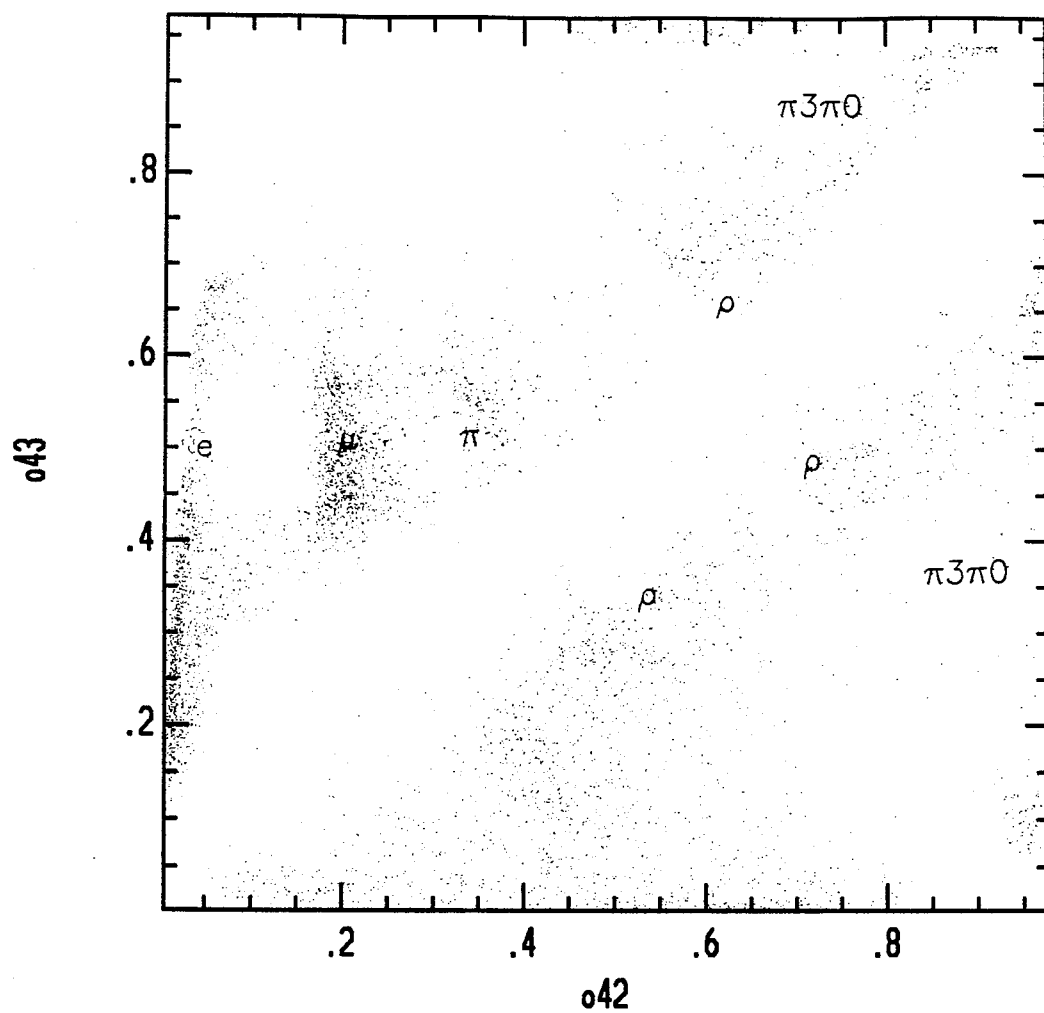


Figure 5.8: Unsupervised tau decay clusterization.

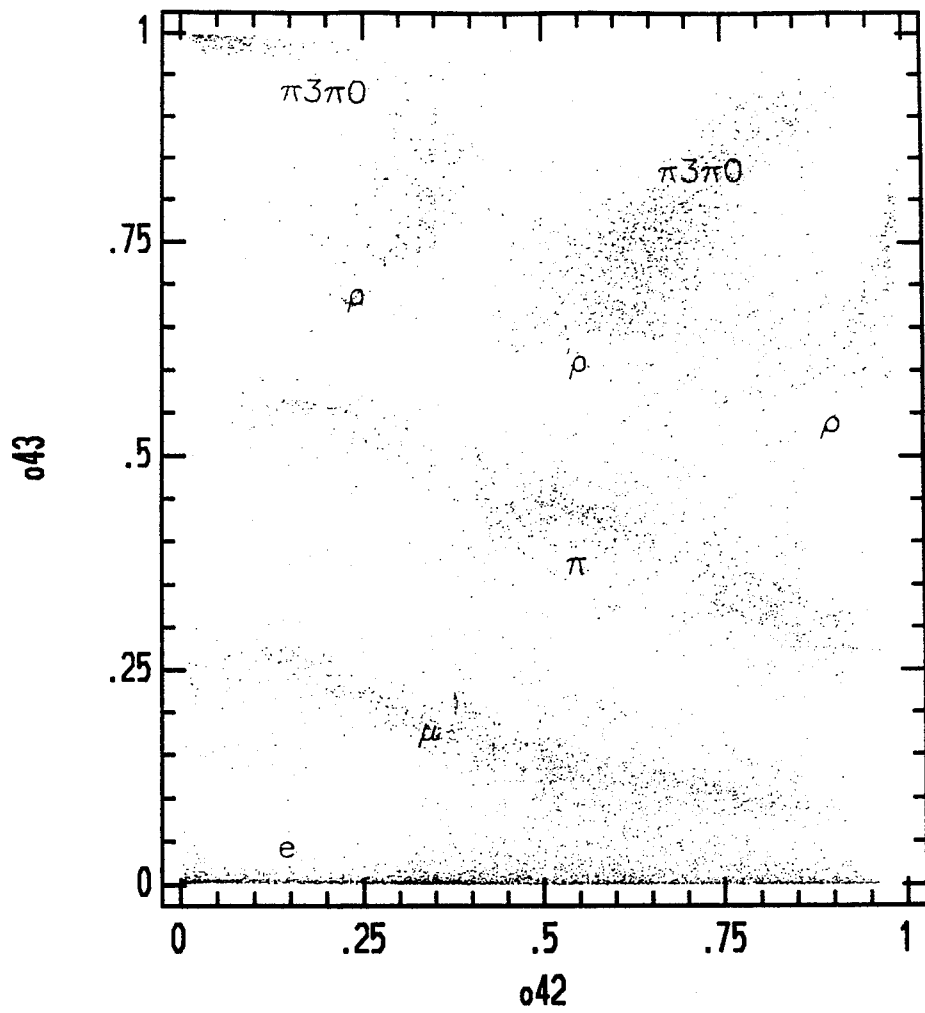


Figure 5.9: Unsupervised tau decay clusterization.

Chapter 6

Pattern recognition

6.1 Track Finding

6.1.1 Introduction

Track finding can be regarded as the prototype of pattern recognition tasks in High Energy Physics. A possible definition of the problem can be [44]

DEFINITION: Given a set of position measurements in a detector the task of track finding is to split this set into subsets such that

- Each class contains measurements which could be caused by the same particle.
- One class (which may be possibly empty) contains all the measurements that cannot be associated with particles with sufficient certainty.

This definition reduces track finding to a cluster analysis problem. A cluster is, in this sense, a collection of measurements which span a small interval in a given metric (track model).

The procedure comes normally in two steps. In the first, track clusters are found as efficiently as the eye does. This is the pattern recognition step and the most difficult part. In the second step track clusters are tested by a fit to the track model.

The methods for track finding can be classified into two groups: global and local ones. In the global methods all the points enter into the algorithm in the same way. It works well with low density of hits, but can be slow. Examples of global methods are the complete combinatorial method [44] or the histogramming method [45]. The local methods start with a few points and extrapolate to the other points. These methods are mostly used because they usually are faster than the global ones.

Here we will follow the work done by G. Stimpff and L. Garrido [3] and extend it to a discrete approach that can run on special hardware.

6.2 Track finding with Neural Networks

In track reconstruction we want to associate coordinates to tracks. We can regard a track with n coordinates as a set of $n - 1$ consecutive lines ('track segments') with a smooth shape and without bifurcation. These track segments are chosen as the neurons. The weights depend on the geometry (length of the lines, crossing angle between the lines, etc.). For the calculation of the energy terms we have to use the coordinates. Therefore we characterize in the following the neuron i (j) by the coordinate k (m) at which it starts and by the coordinate l (n) where it ends. The most general energy function for a Hopfield network is

$$E = -\frac{1}{2} \left(\sum_{klmn} (T_{klmn} - \alpha C_{klmn}) V_{kl} V_{mn} - \beta D \right). \quad (6.1)$$

Since we are only interested to connect neurons with one coordinate in common we have

$$T_{klmn} = \delta_{lm} T_{klln} \quad (6.2)$$

and we describe these matrix elements from now on by the indices of its 3 coordinates, T_{kln} .

The constraints are formulated in such a way that we do not associate a coordinate to more than one track (α term) and that we activate the expected number of neurons (β term). For convenience we limit the range of V_{kl} to the interval $[0,1]$. The inhibition terms can now be written in the form [49]

$$C_{klmn} = \delta_{km}(1 - \delta_{ln}) + \delta_{ln}(1 - \delta_{km}) \quad (6.3)$$

$$D = \left(\sum_{mn} V_{mn} - N_a \right)^2. \quad (6.4)$$

N_a is the expected number of activated neurons. Combining the last 4 equations yields

$$E = -\frac{1}{2} \left[\sum_{kln} T_{kln} V_{kl} V_{ln} - \alpha \left(\sum_{kln(n \neq l)} V_{kl} V_{kn} + \sum_{klm(m \neq k)} V_{kl} V_{ml} \right) - \beta \left(\sum_{mn} V_{mn} - N_a \right)^2 \right]. \quad (6.5)$$

This means that we have in the α term for each neuron kl to sum over all the other neurons which start at the coordinate k or end at the coordinate l . Hence there is a sort of competition between the neurons starting or ending at the same coordinate and each neuron tries to switch off the competing neurons via this constraint term.

Combining equation 4.7 with 6.5 we finally get the following update rule for the neurons

$$V_{kl} = \frac{1}{2} \left[1 + \tanh \left(\frac{1}{T} \sum_n T_{kln} V_{ln} - \frac{\alpha}{T} \left(\sum_{n \neq l} V_{kn} + \sum_{m \neq k} V_{ml} \right) - \frac{\beta}{T} \left(\sum_{mn} V_{mn} - N_a \right) \right) \right]. \quad (6.6)$$

A complete discussion in the choice of the parameters and the update strategy can be found in [3].

We have developed a new method for track reconstruction based on the ideas outlined in the previous chapter. Our goal was to find a neural network algorithm which can compete with conventional methods concerning efficiency, memory usage and execution time on normal computers under realistic conditions. As such a test case we have choose the coordinates measured by the TPC of the Aleph detector at LEP, described shortly in chapter 5.

6.2.1 Track finding in the ALEPH TPC: Conventional method

ALEPH uses a local method for track finding [51](see Figure 6.1 for the algorithm flow chart). In the first step, the algorithm searches for a set of neighboring coordinates that lie on a helix. Then it propagates them to obtain the longest possible chain and merge broken chains. Finally it assigns unassociated coordinates to found tracks using larger tolerances. The method works very well with reasonable speed. Hence it is a very good candidate for comparison with neural network algorithms.

6.2.2 Definition of the neurons

In the Aleph TPC we have to cope with events with more than 500 coordinates. If we connect all these coordinates we get more than 2.5×10^5 neurons and more than 6×10^{10} connections between these neurons. To keep the size of the problem reasonable we apply several cuts.

First we make a quality cut on the coordinates and keep only coordinates which are reasonably well measured :

- $\sigma_{xy} < 0.3cm$
- $\sigma_z < 1.0cm$.

In this way we eliminate about 5% of the coordinates. The remaining coordinates are only connected by lines if

- the difference in phi is < 0.15 rad

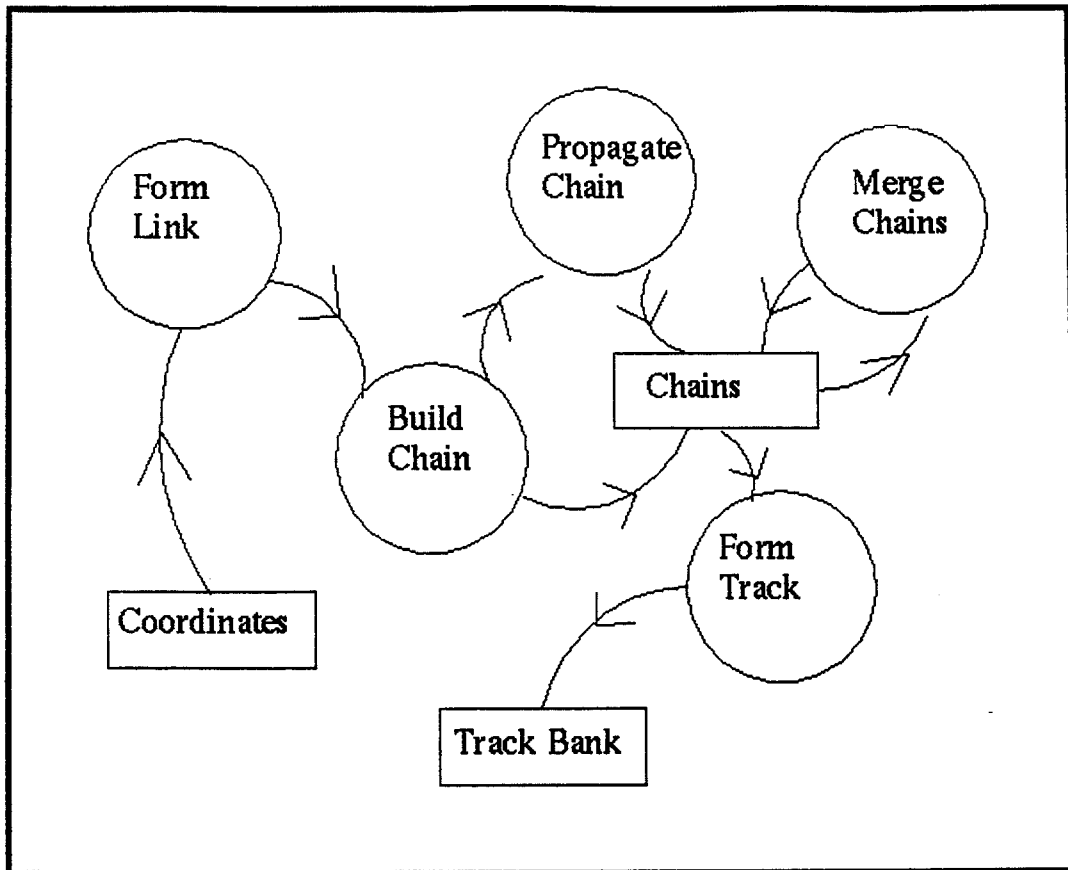


Figure 6.1: Flow chart for the standard ALEPH track finding algorithm.

- the difference in theta is < 0.10 rad
- the difference in the pad-row number is < 4 .

By applying these cuts we reduce the number of neurons by more than a factor 100. We connect the coordinate pairs only in one direction and we chose the coordinate with the bigger radius in the X-Y plane ($\rho = \sqrt{x^2 + y^2}$) as the starting point. The lines point therefore inwards.

6.2.3 Definition of the weight matrix

Since the coordinates in the Aleph TPC are real 3-dimensional points we propose a weight expression in the form

$$T_{kln} = \frac{\cos^\lambda \psi_{kln}}{d_{kl}^\mu + d_{ln}^\mu} . \quad (6.7)$$

ψ is the angle in space between the two lines and d_{kl} the distance between the coordinates k and l in XY. Since we do not want to give smaller weights to forward tracks we do not take the Z coordinate into account in the definition of d . The distances are normalized to the distance between two pad-rows to have maximal weights of the order of 1. We give the above weights to connections which fulfill the condition

$$\cos \psi_{kln} > 0.90 .$$

All other weights are set to zero. We prefer not to have negative weights because it happens rather frequently that a neuron belonging to a track has several bad connections and it might get switched off if their contributions are negative. We rather want to eliminate such connections by the inhibition term α . Although we connect two points only in one direction (from bigger ρ to smaller ρ) we have to sum in the weight term for a given neuron kl over the contribution of all 'incoming' lines (ending at k) and all 'outgoing' lines (starting at l). This approach of a simple local pattern recognition works astonishingly well. We found it far superior over other methods which we tried out, for example a penalty term for line crossings. The time and memory demand of such a term is rather high and one has an additional parameter to tune.

6.2.4 Update of the neurons

We use the same α term as described in section 6.2 but instead of the rather weak and ambiguous global constraint term β we prefer to have an input bias term B which has the same value for all neurons. We give B a small positive value to stimulate the activation of the neurons at the beginning, and we can lower its value towards the end of the iterations in order to switch off neurons which are no longer enforced by their neighbors. We also multiply the weight terms by the parameter c . This allows us to tune the interaction between the weights, the constraints and the bias with the parameters c , α and B and to treat the temperature as a mere normalization factor to control the speed of the convergence. The update rule for the neurons reads now

$$V_{kl} = \frac{1}{2} \left[1 + \tanh \left(\frac{c}{T} \sum_n T_{kln} - \frac{\alpha}{T} \left(\sum_{n \neq l} V_{kn} + \sum_{m \neq k} V_{ml} \right) + \frac{1}{T} B \right) \right] \quad (6.8)$$

with the weights T_{kln} given in equation (18).

We update the neurons asynchronously starting at the outer boundary and we iterate this process until the system converges. We can measure the convergence either by the change in energy between two iterations or by the sum over the changes of the activation of all neurons. We used the criterion

$$\frac{1}{N} \sum_{kl} |V_{kl}^I - V_{kl}^{I-1}| \leq .0005 \quad (6.9)$$

to stop the iterations. N denotes the number of neurons and I the iteration number. Since we use a limited number of weights in general the system converges rather fast (in less than 10 iterations).

6.2.5 Reconstruction of events and results

The algorithm described in the last section has been tested on real events measured by the Aleph TPC at LEP, and Monte Carlo generated events at the LHC energy, and compared its performance with the standard Aleph event reconstruction program JULIA. For this purpose we have selected decays of the Z^0 into hadrons with more than 20 charged particles.

We used for our study the parameter set

$$c = 10, \quad \alpha = 5, \quad B = 0.2, \quad T = 1, \quad \lambda = 5, \quad \mu = 2$$

and achieved good quality of the results and fast convergence. The behaviour of the network does not depend strongly on the parameters and it is therefore easy to find appropriate values for them.

First we want to demonstrate the convergence process on a real event. Figures 6.2 and 6.3 show all neurons in the XY and the RZ projection for a real Z^0 event measured in the Aleph TPC. We see that well separated tracks are already almost 'reconstructed', except from overlapping lines. But we have many lines in regions with nearby tracks.

We let the system evolve and it converges after 6 iterations. Figures 6.4 and 6.5 show all neurons with an activation $> .9$. All tracks have been found properly.

We have studied the track finding efficiency of this NN algorithm. The efficiency for a track which has more than 6 reconstructed coordinates and which leaves the TPC is bigger than 99%, comparable to the 99.7 % of the standard Aleph method. The biggest difference is the number of wrongly associated coordinates. While this is a rather rare case for the conventional method this happens more frequently in the NN approach. It is clear that a neural network program which uses only soft constraints and very simple geometrical constants is not as good as a sophisticated algorithm based on hard constraints (fits). But the big advantage of neural networks is their flexibility, the small number of parameters and their capability to find solutions near the optimum.

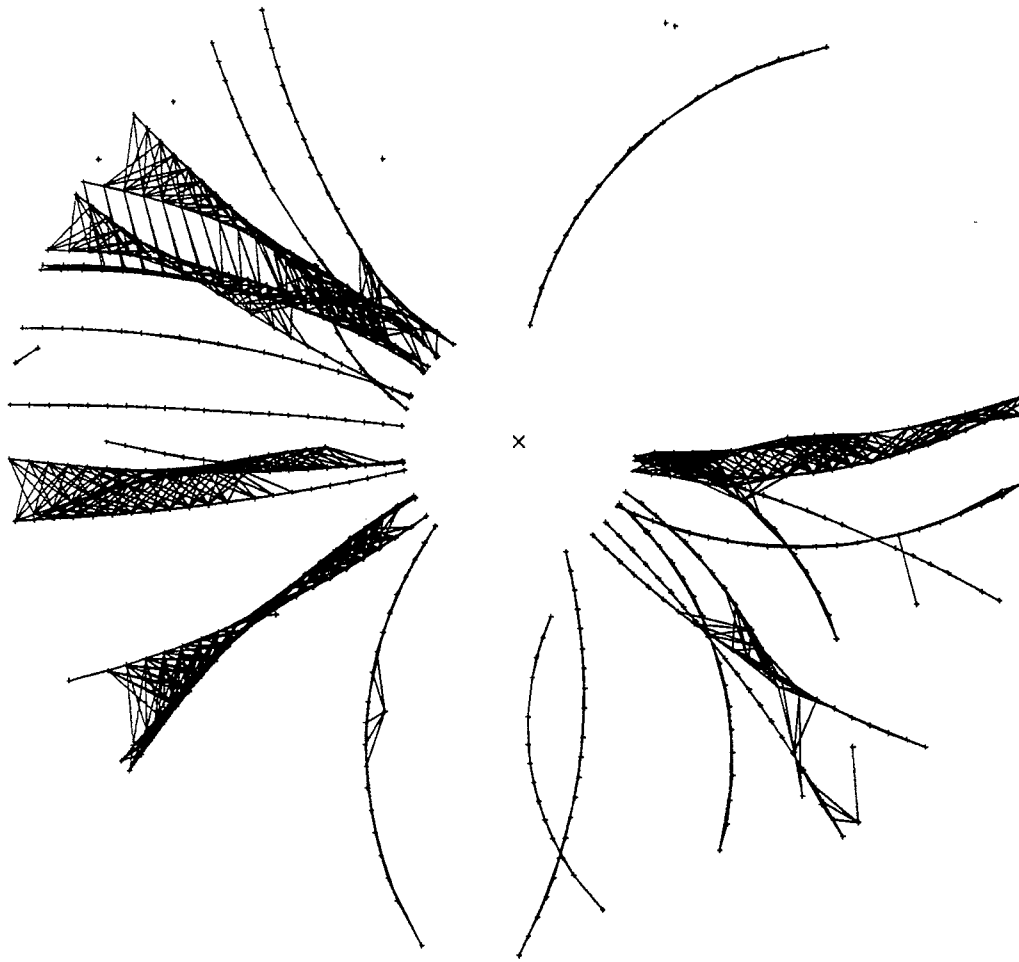


Figure 6.2: Display of all generated lines for a real $Z^0 \rightarrow \text{hadrons}$ (X-Y projection).

For LHC detectors we expect about 200 tracks per event. We studied the dependence of our NN approach as a function of the number of tracks per event with 21 points per track. Figure 6.6 shows the time used as a function of the number of tracks for the NN and the conventional algorithm. The NN method can again compete with the standard one. Furthermore we see that more than 60% of the NN time is used for the initialization.

As an example of a large reconstructed event we show in figure 6.7 the generated lines for a Monte Carlo event with 100 tracks, and in figure 6.8 the activated lines

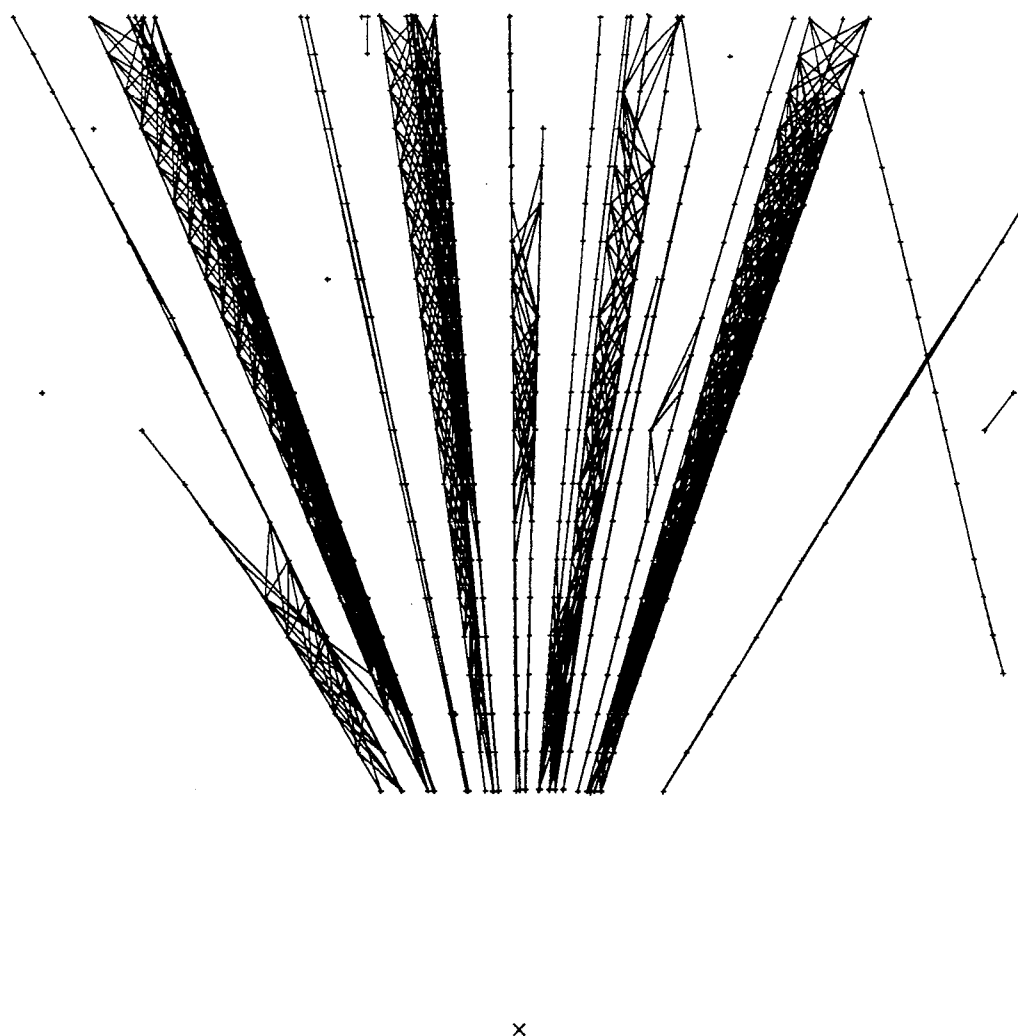


Figure 6.3: Display of all generated lines for a real $Z^0 \rightarrow \text{hadrons}$ (R-Z projection).

after convergence.

6.3 Discrete track finding

As we showed previously, the time spent in the setup of the iterations is quite long even in our very simplified program and makes up about 60% of the total execution time. Therefore we cannot profit much from the use of special hardware (e.g. a

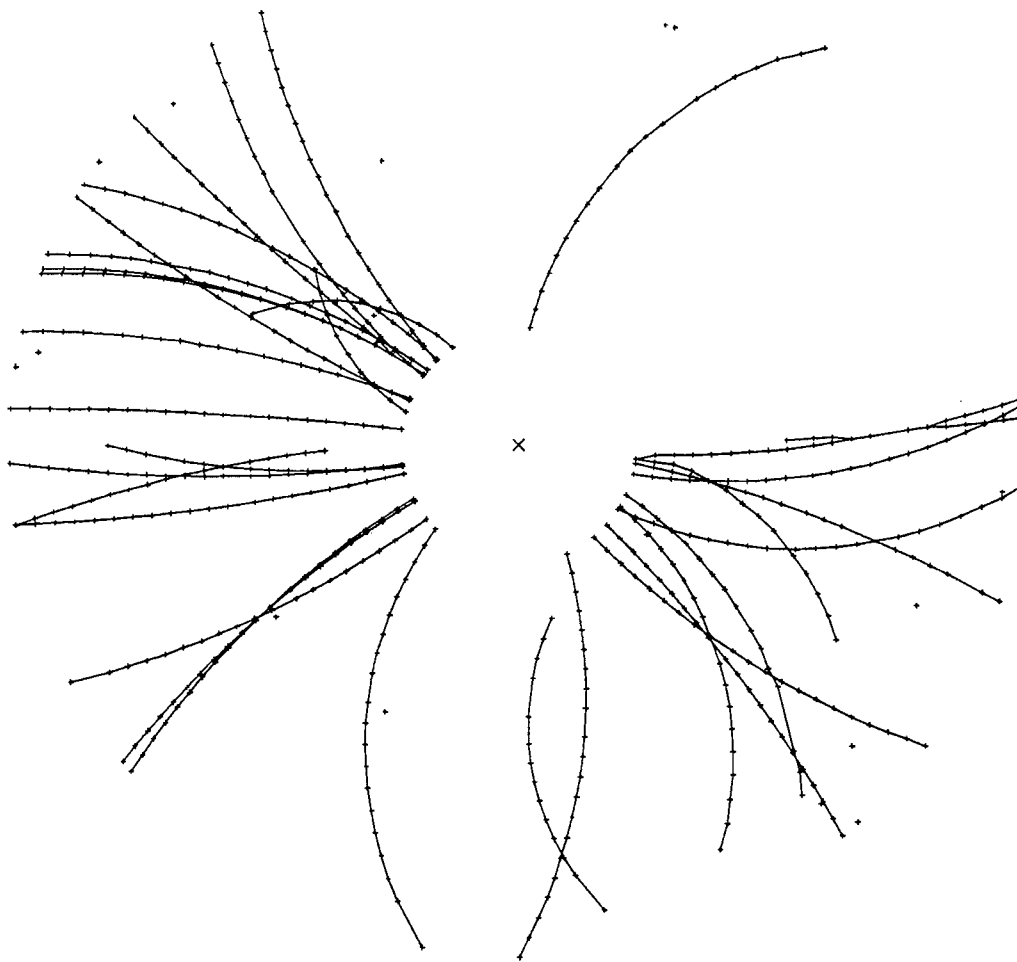


Figure 6.4: Display of the activated lines after convergence for a real $Z^0 \rightarrow$ hadrons (X-Y projection).

neural network chip) using this variable architecture . the setup of the network takes an important part of the execution time. The question is whether this part of the process can be avoided.

A promising approach is to discretize the detector volume and calculate in advance the weights for all the reasonable links (between neurons connecting only pads that can correspond to the same real track) (Figure 6.9).

There are two possible ways to give to the network the discretized coordinates

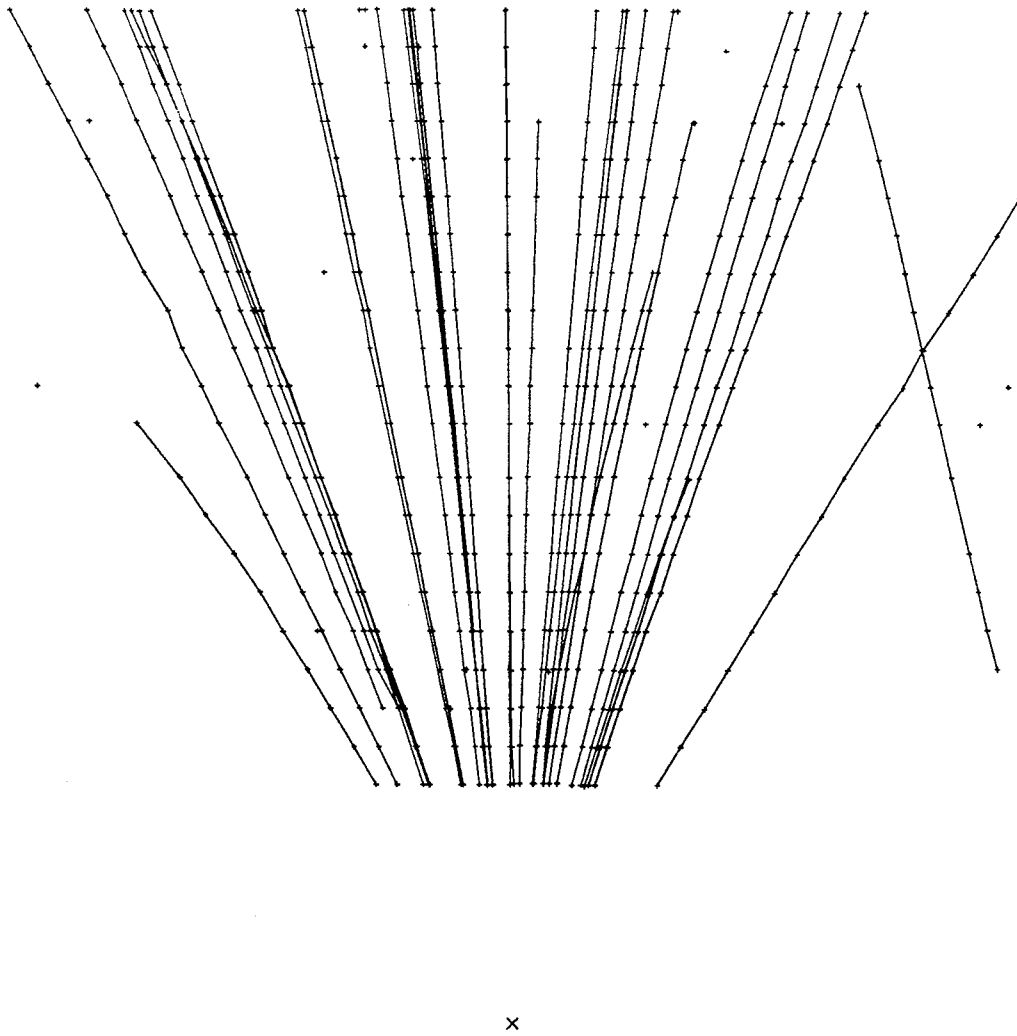


Figure 6.5: Display of the activated lines after convergence for a real $Z^0 \rightarrow$ hadrons (R-Z projection).

measured by the detector. One method is to put to zero all the weights that do not connect neurons belonging to pads with hits. The other method is to add a strong global inhibitory term and a constant excitatory term only to neurons connecting pads with hits.

The dynamics of the network is the same in our previous approach, but in this case it is not necessary to recalculate the weights for each event. Then, the execution time comes only from the relaxation of the network that can be performed

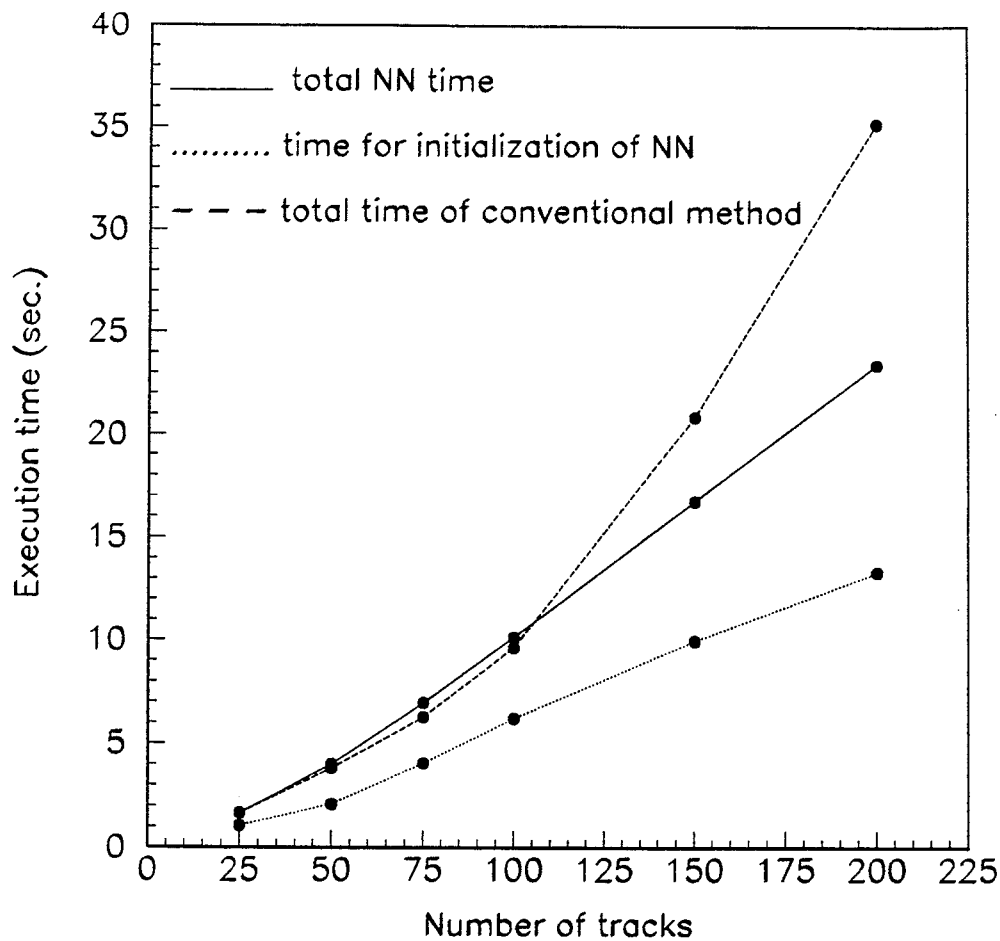


Figure 6.6: Execution time as a function of the number of tracks.

by a neural chip.

6.3.1 Simulation

We chose a tracking detector with cylindrical geometry segmented in polar angle and radius, but not in z (only information in the xy plane). In Figure 6.10 all the possible neurons for this detector are shown, and in Figure 6.11 a reconstructed event using this approach is shown.

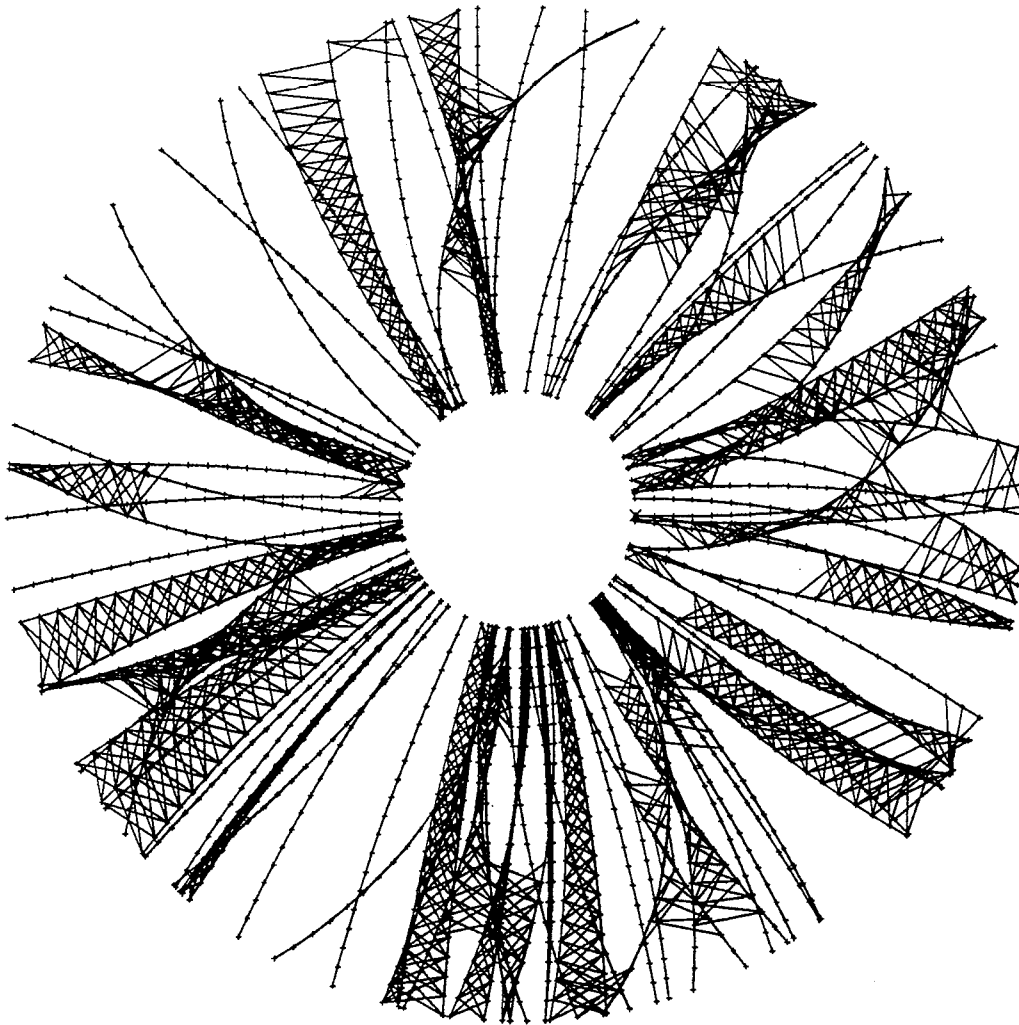


Figure 6.7: Display of the generated lines for a Monte Carlo event with 100 tracks (X-Y projection).

The track finding quality depends strongly in the granularity of the detector segmentation. Notice that the segmentation makes a cut in the momentum of the particles for given links.

If you want a high track efficiency, the number of neurons needed grows prohibitively, but, for trigger applications this approach can be useful.

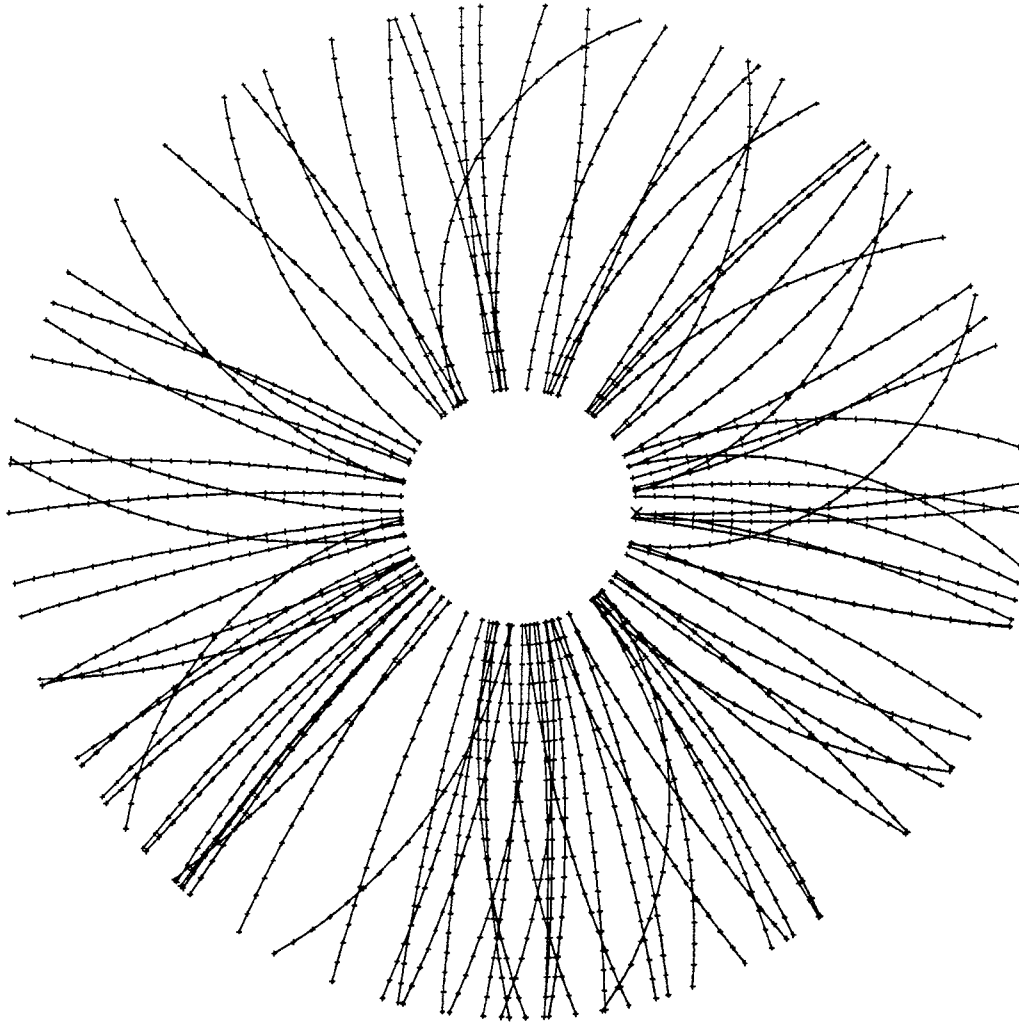


Figure 6.8: Display of the activated lines after convergence for a Monte Carlo event with 100 tracks (X-Y projection).

6.3 Hardware implementation

In this task it is important to have a good balance between the number of neurons and execution time. Analog implementations are faster, but the available chips has few neurons for this application. On the other hand, digital implementations can handle a big number of neurons, but its speed is not well suited for real-time data processing. One possibility for solving this problem is to use only discrete weights

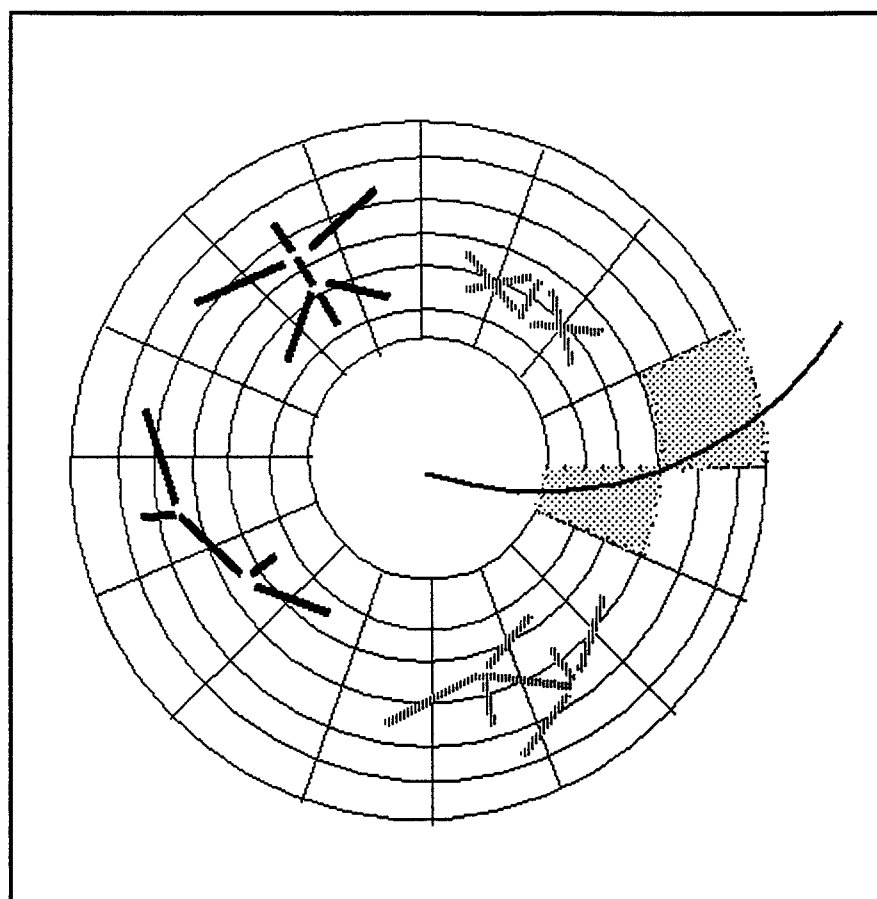


Figure 6.9: Schematic view of the detector segmentation. Solid lines: excitatory connections; dashed lines: inhibitory connections. The dotted area corresponds to the pads with hits caused by the track.

(1,0 and -1 for instance). This speeds up dramatically the update of the network and allow a fast digital implementation [52]. Discussion about the performance and learning algorithms for neural networks with discrete weights can be found in [53].

Using this idea, the Centre Nacional de Microelectrònica of U.A.B. designed one general purpose neural coprocessor called NDN3 [52]. This is a digital VLSI chip including up to 4096 fully connected binary neurons. The weights can be -1,1

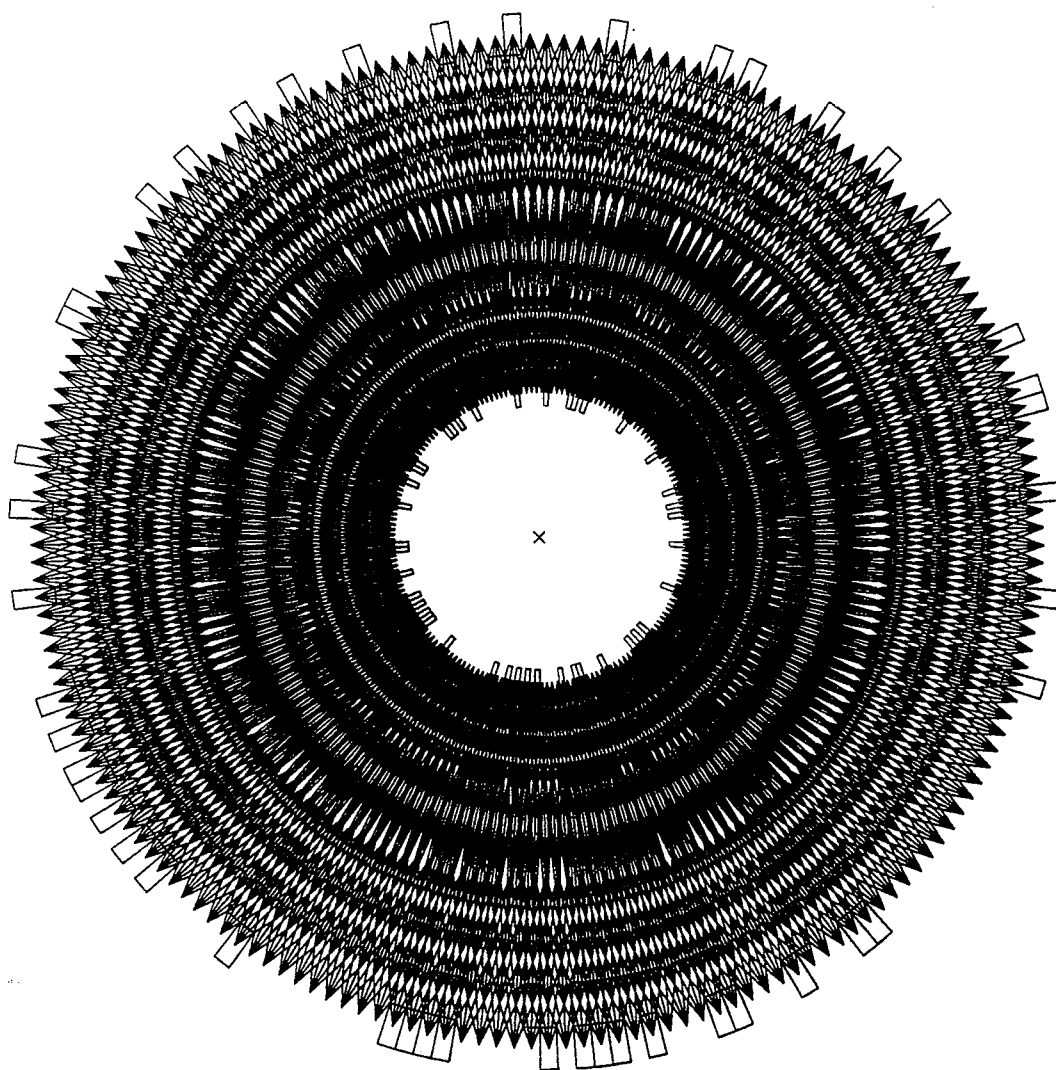


Figure 6.10: Display of all possible neurons using fixed architecture (X-Y projection).

or 0. This fact allows to chose the network architecture by putting to zero some connections.

Running in a PC at 12 MHz this chip can make $10^9 - 10^{10}$ updates per second. The time estimate for a typical event with 500 coordinates implies a relaxation time for the track finding task of the order of $10 \mu\text{s}$. This time can be improved using faster memories and processors, that make this approach useful for on-line track finding. The remaining question is how the method performs with discrete weights.

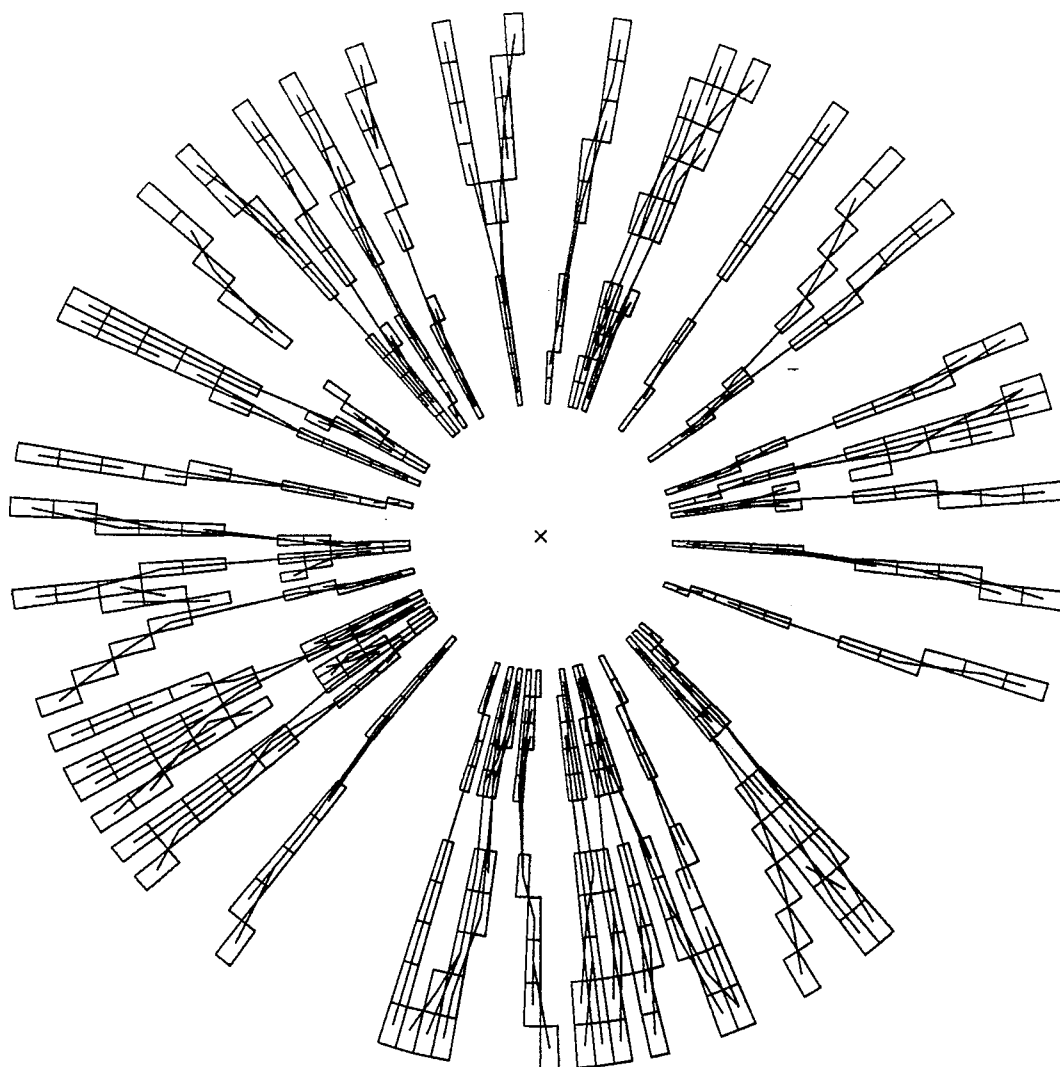


Figure 6.11: Display of a reconstructed event with 50 tracks using fixed architecture (X-Y projection).

Clipping the weights obtained in the continuous case gives the pattern in Figure 6.12. The simulations performed using this set of weights converge to the same solutions as in the previous case (discrete volume but continuous weights). Then, the loss of track finding quality comes only from the volume segmentation step. We can conclude that discrete weights and two-valued neurons are good candidates for this neural network algorithm.

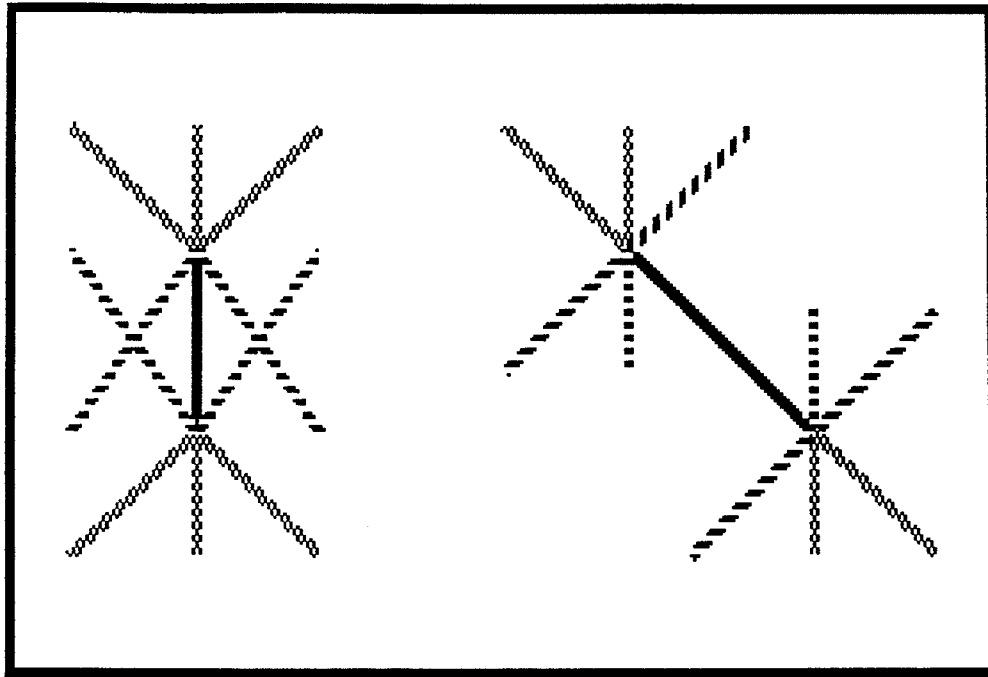


Figure 6.12: Weights obtained after clipping. Dashed (dotted) line corresponds to a +1 (-1) weight in the connection with the solid neuron.

6.3.1 New Hardware developements

The time response of the above technique can be dramatically sped-up using special purpose implementations . The network connectivity shown in fig 6.9 implies that the influence on a neuron can be reduced to five neurons in the horizontal direction, and three clusters in the vertical one, a cluster being the set of three neurons corresponding to the potential track emerging from the same point. It is also shown that there are only three different patterns for the neurons connectivity.

As a consequence, a new chip [54] was conceived with fixed weights corresponding to this pattern of connection. The two strategies for the implementation of analog neural networks come from the classification of computational styles: analog and digital. These styles could be mapped in terms of VLSI implementation with a higher degree of integration without the requirement of a very specialized manufacturing process. Both have a wide range of application based on the balance

between parallelism versus precision.

In the implementation proposed, a direct study shows that analog implementations, based on current mode computations together with digital logic for data I/O, have higher performance than digital ones (composed of 19 full-adders, 4 basic combinational gates and 1 memory element per neuron) in terms of both area (considering full-custom design for both implementations), and speed; the recall speed will be slightly faster in analog implementations. Analog implementations have also better performance in terms of redesign of the network, due to the fact that synaptic weights are directly mapped onto transistors, that are placed in specific layout blocks, which are easy to modify according to new requirements, whereas digital implementations require resynthesis and redesign of the logic structure.

Figure 6.13 shows a basic schematic of the cluster, i.e. of the set of neurons corresponding to the three segments originating from of a track pixel. In this figure the excitatory and inhibitory weights are seen to be mapped as transistors connected to a two rail bus that is sensed by a transconductance amplifier. The management of data is carried out by a dynamic shift register.

Although fixed weights mean a loss of programmability, it is the only way to obtain an IC able to deal with higher amounts of data.

The symmetry of the whole IC allows the use of most of the tools common in low level design, including design for portability among different technologies. Portability means a design with simple structures for easy recompaction, and a structure compiler to easily build ICs with different number of rows and columns according to the structure of the detector. Figure 6.14 shows the physical implementation of synaptic connections. The high density achieved by a semi-automatic compaction gives the small dimensions of 90x126 microns , for 30 transistors in 1.5 CMOS technology.

System architecture

The non-regular structure of the IC is due to the special characteristics of the detector, in which the radial precision is limited to twelve values, and the angular precision should be as high as possible.

In order to increase angular precision the IC has been designed taking into account modularity, in such a way that chips would be connected only through their left and right pixel array sides, leaving top and bottom sides for data input and output (twelve cycles). The structure of the cylinder will then be reconstructed by a ring architecture of this neuroIC.

In addition to the neural filter, some special circuitry was designed in order to detect filtered tracks. This circuitry is also designed taking modularity into account.

This task requires very high speed ($1.8 \cdot 10^{12}$ cps.), that could be reached through

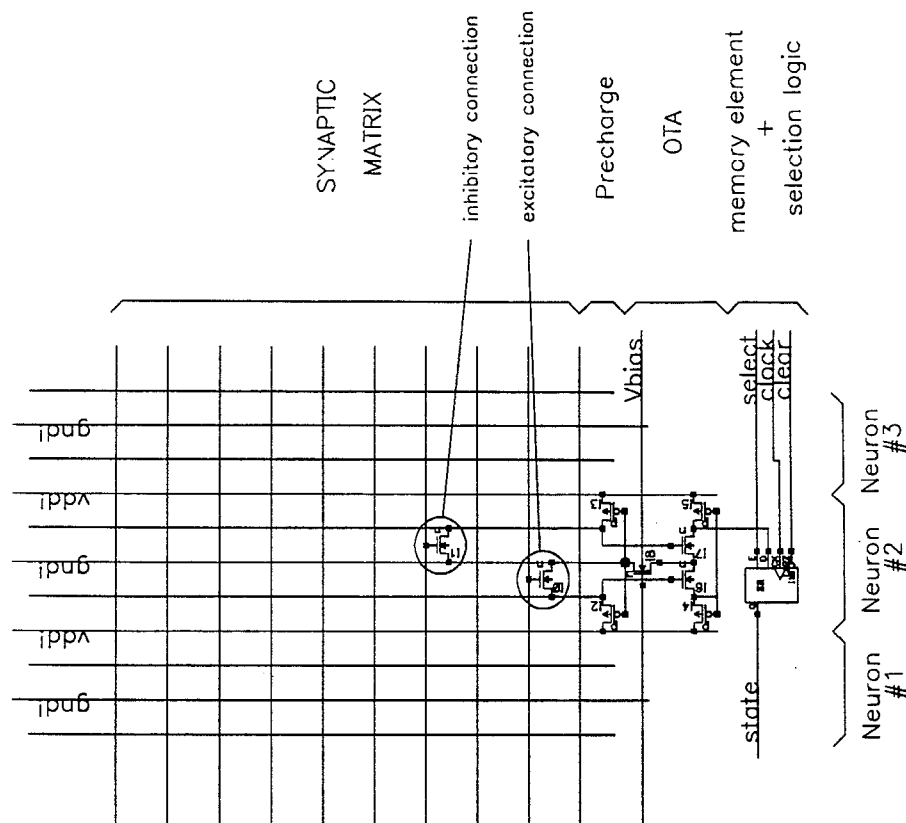


Figure 6.13: Basic schematic of the neuron cluster.

a non programmable analog neural network, with low connectivity of discrete weights.

The main computational parameters of the circuit are:

clusters (or pixels) : 600 (50x12)

neurons : 1800

weights per cluster: 30

area: 40 mm² (1.5 ES2)

package: 120 PGA

clock speed : 100 MHz.

neural speed : $1.8 \cdot 10^{12}$ cps.

This chip will be available very soon. At this moment is still in test phase.

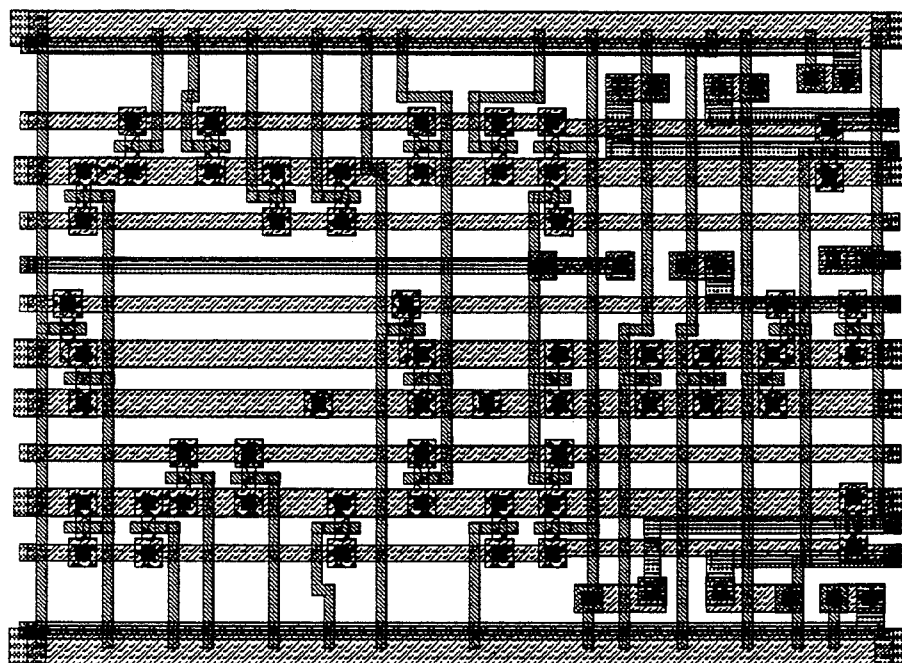


Figure 6.14: Physical implementation of synaptic connections.

Chapter 7

Some ANN applications in HEP

7.1 Current developements on NN in HEP

In this chapter a brief outline on the state-of-the-art is presented by refering several abstracts of works recently submitted for publication or presented at workshop. Two main areas of developement are quoted: Triggering and event reconstruction, and off-line data analysis. (Author and its adress are given in case you want more information from them)

7.1.1 Triggering and event reconstrucction

— CHARGED PARTICLE TRACK RECONSTRUCTION USING ARTIFICIAL NEURAL NETWORKS

Charles Glover, Tony Gabriel, Peter Fu Oak Ridge National Laboratory, Oak Ridge, Tennessee, USA Thomas Handler Physics Dept., University of Tennessee, Knoxville, Tennessee USA

We present an ANN algorithm for reconstructing charged particle tracks directly from tracking chamber data rather than from the set of all possible track segments which is the technique used by most ANN track reconstruction algorithms. This algorithm uses a model of the oriented receptive fields found in the retina. Each tracking chamber's data point (or hit-point) has a set of neurons associated with it. Each neuron represents a direction in space that a track could take while passing through the given hit-point. Each neuron's activation level (output) is proportional to the probability that the track is oriented along the direction represented by the neuron. A neuron's output is a result of cooperation with neurons within its oriented receptive field and its competition with neurons of different orientations. This ANN algorithm generates a vector field within the tracking chamber's volume delineating the

charged particle tracks. In addition, VLSI hardware exists to implement this algorithm and the only inputs needed are the coordinates for each hit-point, not segments as is true with other ANN tracking algorithms.

– FAST TRACK FINDING WITH NEURAL NETS

Georg Stimpfl-Abele, Laboratoire de Physique Corpusculaire Universite Blaise Pascal Clermont-Ferrand, Aubiere, France, Lluís Garrido, Departament de estructura i constituents de la Matèria, Universitat de Barcelona, Spain.

We have used a neural network (NN) technique for track reconstruction in a realistic environment. An algorithm based on an Hopfield-style recurrent NN was developed and tested on the track coordinates measured by the TPC of the ALEPH detector at LEP. The efficiency and time consumption are given and are compared with a conventional pattern recognition method. The performance of the algorithm for large numbers of tracks (up to 200), as expected for LHC and SSC detectors, is discussed.

– FINDING THE DECAY VERTEX OF A CHARGED TRACK WITH NEURAL NETWORKS

Georg Stimpfl-Abele Laboratoire de Physique Corpusculaire Universite Blaise Pascal Clermont-Ferrand, Aubiere, France

The task of finding decays of charged tracks inside a tracking device is divided into two parts. First a neural network is used to recognize kinks in well reconstructed tracks. The inputs to this classification network are the residuals and the curvature obtained by a one-track fit. If a kink has been found the same inputs are fed into a second neural network which gives the radial position of the decay vertex. Both NN algorithms use feed-forward nets with error back-propagation. Very good performance is found in comparison with conventional methods.

– APPLICATION OF CELLULAR AUTOMATA AND NEURAL NETWORKS FOR EVENT RECONSTRUCTION IN DISCRETE DETECTORS

I.V. Kisel, G.A. Ososkov Joint Institute for Nuclear Research, 141980 Dubna, Russia

Methods considered in the given work are based completely on the structure discreteness of detectors like multiwire proportional chambers (MWPC). This leads to the use of Chebyshev metrics instead of the conventional least square method. We successfully apply cellular automata by changing their logic for filtering track data of charged particles registered in MWPC. Neural networking renders possible to fulfill not only track search but also their parameters obtaining. Fast algorithm proposed for vertex search is based on the Chebyshev metrics and is effective even in the case of the big target. Results of the successful application of the above methods to experimental data observed on

the ARES cylindrical spectrometer during the search of forbidden and rare decays are given.

– FEASIBILITY OF USING NEURAL NETWORKS AS A LEVEL 2 CALORIMETER TRIGGER FOR JET TAGGING.

Charles Glover, Tony Gabriel, Surender Saini Oak Ridge National Laboratory, Oak Ridge, Tennessee, USA Thomas Handler Physics Dept., University of Tennessee, Knoxville, Tennessee, USA

The Higgs particle, through several of its decay modes, can end up in jets. We propose to incorporate a second level trigger in the SDC detector using neural network VLSI hardware to tag events wherein this has occurred. The input to the neural network is the energy depositions in both the barrel and endcap regions of the calorimeter. The neural network's output is a value representing the degree of correlation between the observed energy distribution and the type of physical scattering process that produced the distribution. Results will be presented from ongoing research where neural networks are trained from Monte Carlo simulations of the energy distributions deposited in the SDC calorimeter.

– FEED-FORWARD NEURAL NETWORKS FOR SECOND-LEVEL TRIGGERING ON CALORIMETER DATA

M. Kolstein, J.C.Vermeulen, L.W.Wiggers NIKHEF-H, Amsterdam, The Netherlands H.M.A.Andree, G.T.Barkema, A.J.Borgers, W.Lourens, A.Taal Faculty of Physics and Astronomy Utrecht University, Utrecht, The Netherlands

We present two types of feed-forward neural networks for the recognition of energy deposition patterns in a calorimeter. The first network is constructed on the basis of explicit linear threshold equations, implementable by simple linear threshold neurons. The second network results from a constructive learning algorithm with neurons performing concealed functions. We used Monte Carlo events for a hypothetical spaghetti calorimeter and for the Zeus uranium calorimeter for learning, and in the design and for testing of the neural networks. A hardware realization of a second-level trigger for an LHC experiment based on these algorithms and constructed from Digital Signal Processors (the Texas Instruments TMS320C40) seems feasible and will be discussed in the talk.

– NEURAL NETWORKS AT THE TEVATRON

W. Badgett, S. Bianchin, K. Burkett, M.K. Campbell, A. Caner, M. Dall'Agata, M. DeNardi, B. Denby, H. Haggerty K. Johns, C.S. Lindsey, M. Dickson, G. Pauletta, L. Santi L. Stanco, N. Wainer, D.Y. Wu, J.L. Wyss

Presented by B. Denby at COMPUTING IN HIGH ENERGY PHYSICS 21-25 September, 1992 Annecy, France

This paper reviews four neural network applications at the Fermilab Tevatron:

1) Online Muon Tracking with a VLSI Neural Network. This reviews the first application of a hardware neural network in a high energy physics detector: the Intel ETANN chip provided intercept resolution 50 times better than the conventional hodoscopic method in only 3 microseconds without increasing deadtime.

2) CDF and D0 Neural Network Triggers. The CDF experiment is currently debugging three triggers based on the ETANN chip: central photon isolation; end plug electron isolation; and central semileptonic b trigger. All are based on calorimetry. The central photon chip has recently been used successfully as a trigger in a test run. Longer runs and incorporation of the other two triggers will follow shortly.

A test using cosmic rays has been made of a trigger based on the D0 experiment muon chambers. A neural network made from discrete components was constructed which determines the beam line intercept of muon tracks in the chambers.

3) Offline Quark/Gluon Discrimination. A feedforward neural network trained on Monte Carlo quark and gluon jets was applied to real jets from the 1989 run of CDF. Although the quark and gluon network output distributions overlap significantly, the mean net output can be used as an indicator of the quark and gluon fractions. The mean output was found to increase with jet E_t , as expected from QCD.

4) A New Tool for Top to Multijets Recognition at CDF. A multilayer perceptron was trained to discriminate between Monte Carlo generated $t\bar{t}$ events decaying to six jets and real CDF events (assumed to be pure background) also containing six jets. Only the E_t , rapidity (η), and azimuthal angle (ϕ) of each jet were used as net inputs. The separation achieved is encouraging, and is significantly better than that obtained using the Fisher linear discriminant. This result confirms that a properly trained neural network must always perform as well or better than a linear discriminant. This study is also interesting since the input variables used, E_t , η , and ϕ , could readily be made available in a trigger system.

– FINDING THE TOP QUARK AT THE TEVATRON COLLIDER BY MEANS OF NEURAL NETWORKS

Debra Dzialo Karatas Center for Particle Physics Department of Physics, The University of Texas Austin, Texas 78712 H. Baer Florida State University Gian. F. Giudice The University of Texas

The search for the top quark at $p\bar{p}$ colliders in the one-lepton plus jets channel is plagued by an irremovable background from W-boson plus multi-jet production. We show how the top quark signal can be distinguished from

background in the distribution of neural network output. By making a cut on the network output, we maximize the ratio of signal to background in a final event sample, and compare our results with those obtained by making conventional kinematical cuts on the data sample. We also demonstrate the robustness of the neural network method by training the neural network on signal events of one top mass and testing upon another.

– OPTICAL NEURAL NETWORKS FOR THE SECOND LEVEL TRIGGERING

Pavel Bitzan Institute of Computer and Information Science 182 07 Prague 8, Czechoslovakia

Optical neural networks are systems that are able to substitute for some functions of standard neural nets which are important in information processing. They are able to learn by themselves, and to store and recall information. The response time of such a network is substantially sped-up in comparison to standard micro-electronical models. Consequently, such a model is appropriate for data processing in high energy physics. The goal of this contribution is to present a new model of optical neural networks and to demonstrate its performance for the second level triggering.

– PATTERN DISCRIMINATION WITH FEED-FORWARD NEURAL NETWORKS

Thorsteinn Rognvaldsson University of Lund

The Multilayer perceptron (MLP) network is shown to handle high-dimensional pattern classification problems in a more efficient way than the Learning Vector Quantization (LVQ) network. This is due to the sigmoidal form of the MLP transfer function, and to the fact that the MLP uses hyper-planes more efficiently. Both algorithms are also found to be equally robust to limited training sets.

– TRACK-FINDING WITH DEFORMABLE TEMPLATES

M. Ohlsson, C. Peterson, A. Yuille University of Lund

A novel algorithm for particle tracking is presented and evaluated. It is based on deformable templates that converge using a deterministic annealing algorithm. These deformable templates are initialized by Hough transforms. The algorithm, which effectively represents a merger between neuronic decision making and parameter fitting, naturally lends itself to parallel execution. Very good performance is obtained for both non-magnetic and magnetic tracks. For the latter simulated TPC tracks from the CERN DELPHI detector are used.

– NEURAL NET APPROACH FOR A CALORIMETER TRIGGER AT LHC

Philippe BUSSON LPNHE Ecole Polytechnique Route de Saclay F-91128 Palaiseau Cedex e-mail : BUSSON@CERNVM.CERN.CH Marc DURANTON Laboratoire d'Electronique Philips BP 15 22 Avenue Descartes F-94453 Limeil-Brevannes Cedex e-mail : DURANTON@LEP-PHILIPS.F

We propose to use a very simple NN in order to discriminate electron showers and jet showers. Preliminary studies made in the framework of RD11 project at CERN for a calorimeter a la SpaCal ($\Delta\eta * \Delta\phi = 0.03 * 0.03$) lead us to the following architecture.

For each cell of this calorimeter we associate only ONE neuron. This neuron uses as inputs the values of energy deposition of the cell together with those of the neighboring ones. Our study showed that only 25 inputs (for cells arranged in a square matrix) or 19 inputs (for cells arranged like in a brick wall) are necessary to obtain a rejection factor of about 50 against jets while giving an electron efficiency better than 95

One should notice that we only use transversal properties of the shower and, most important, that this approach doesn't require a pre-processing of the data fed to the neuron.

We have implemented this kind of algorithm on a special hardware developed by LEP (Laboratoire d'Electronique Philips). LEP lent us a PC-board using one transputer T800 (from INMOS) controlling (via its 32bit bus) four digital neural chips L_Neuro 1.0, which was designed in 1989 by LEP.

L_Neuro is able to perform 16 multiplications (in a SIMD way) of 8bit integers by 8bit integers (signed or not) and a final summation of the 16 results in less than 0.6 microsecond (12 cycles of a 20MHz clock). Using the LEP PC-board we measured about 6 microseconds. This difference of one order of magnitude in execution time is entirely due to the use of the transputer bus to drive the L_Neuro chips.

We have already built (in collaboration with Philips in the framework of the European project GALATEA) new PC-Boards (called ExBus board for Extended Bus) specially designed in order to reach the intrinsic performances of the chip. This new board is presently in an intensive debugging phase in our lab.

- A NEURAL NETWORK CALORIMETER TRIGGER USED IN CDF

W. Badgett, K. Burkett, M. Campbell, D.Y. Wu, University of Michigan Ann Arbor, MI 48109

B. Denby and C. Lindsey, Fermilab Batavia, IL 60510

R. Blair and S. Kuhlmann, Argonne Lab Argonne, IL 60439

J. Romano, University of Chicago Chicago, IL 60637

We describe a pattern recognition electronic trigger, built around an INTEL 80170NX analog neural network chip, supported by 10 full FASTBUS cards and 48 signal tap boards. One of our applications takes advantage of the chip's parallel processing ability, using it as an analog arithmetic unit, to select isolated particles, based on 50 input signals. We show preliminary performance results from data collected at the CDF high energy physics experiment at Fermilab's proton-anti-proton collider.

– A NEW METHOD FOR FAST TRACK RECOGNITION AT PRESENT AND FUTURE COLLIDERS.

Authors: T. Altherr, R. Vilela Mendes and J. Seixas

Abstract: Using a very simple cellular automata algorithm we show how to recognize a large number of spirals in a given pattern. The algorithm can (and in practical application should) be implemented as a massively parallel architecture, the connections between the units being non-adaptable. We present several examples of recognition of simulated particle tracks ranging from a few to one hundred.

How to get this paper: - At Cern th-preprints room (CERN-TH-6794/93) - on Cernvm (GIME TALTHERR, the LaTeX file is SPIRAL TEX and there are 2 postscript figures FIG2 PS and FIG3 PS (figure 1 is not available) - by E-mail to TALTHERR@CERNVM.CERN.CH or ALTHERR@FRCPN11.IN2P3.FR

7.1.2 Off-line analysis

– TAGGING B QUARK EVENTS IN e^+e^- COLLIDERS WITH NEURAL NETWORKS. METHODS TO IMPROVE THE PURITY OF A SAMPLE OF EVENTS.

J.Proriol Laboratoire de Physique Corpusculaire Universite Blaise Pascal, Clermont-Ferrand

We present a set different methods to tag b quark events with neural networks. We consider different sets of variables: global shape variables, jet related variables, lepton related variables, impact parameter related variables. With these different sets, we devise some methods to improve the purity of a b quark events sample. In some cases, we consider also the purity of a c quark events sample.

– BAYESIAN INTERPRETATION OF THE NEURAL NET OUTPUT AND ITS APPLICATION TO TEST THE AGREEMENT BETWEEN TWO EMPIRICAL DISTRIBUTIONS

L. Garrido, V. Gaitan Laboratori de Fisica d'Altes Energies Universitat Autònoma de Barcelona, Barcelona, Spain

We will show that a multilayer feed-forward neural net trained from a sample of examples minimizing the quadratic error function are approximations to a Bayesian Decision Rule. This fact is used to introduce a new method to test the agreement between two n -dimensional empirical distributions that is not restricted to work with cumulative distributions in fewer dimensions due to the lack of data, and with the relevant fact that it is free of binning.

– NEURAL NETWORKS FOR CLASSIFICATION OF QUARK FLAVOURS IN Z^0 DECAYS

G. Cosmo, A. De Angelis and A. Linussio Istituto di Fisica dell'Università di Udine and INFN Trieste Udine, Italy P. Eerola and J. Kalkkinen Research Institute for High Energy Physics, Helsinki

The measurement of the hadronic branching fractions of the Z^0 boson into all five known quarks, from the data collected by the DELPHI detector at LEP during 1990, is presented. The measurement was based on four single-output neural networks. Progress is reported on the design of more powerful network architectures, and on the calculation of systematic errors.

– IDENTIFICATION OF TAU DECAYS USING A NEURAL NETWORK

V. Innocente, Y.F. Wang and Z.P. Zhang CERN and INFN-Sezione di Napoli, Italy INFN-Sezione di Firenze and University of Florence, Italy World Laboratory, FBLJA Project, Geneva, Switzerland and University of Science and Technology of China, Hefei, China

A neural network is constructed to identify tau decays with the L3 detector at LEP. The same network is used to identify $\tau \rightarrow \pi \nu$ and $\tau \rightarrow e \bar{\nu} \nu$ as a cross check. High efficiency in the ρ channel is obtained. A major effort has been made to study the systematic errors introduced by the use of a neural network and no obvious bias has been found.

– THE USE OF NEURAL NETWORK CLASSIFIERS FOR HIGGS SEARCHES
ALESSANDRO PETROLINI

Dipartimento di Fisica dell'Università di Genova and INFN Via Dodecaneso 33, 16146 Genova, Italy

Abstract

Neural Network Classifiers are used to separate the signal from the background in a High-Energy Physics problem. The basic principles of Multidimensional Analysis by means of Back-Propagation Neural Network Classifiers and the application to the Higgs search are discussed. A search for the Minimal Standard Model Higgs boson, through the reaction $e^+e^- \rightarrow H^0 \nu \bar{\nu}$, using the data collected in 1990 by the DELPHI detector at LEP is made. The technique used allows to reach good detection efficiencies and no evidence of the Minimal Standard Model Higgs boson with mass less than 37 GeV is found. The

use of Neural Network Classifiers proves to be a very powerful classification method. A comparison of the method with standard analysis techniques is presented.

REFERENCE *-- Alessandro Petrolini, "The Use of Neural Network Classifiers for Higgs Searches", Int. J. Mod. Phys. C (IJMPC), Vol. 3, No. 4, Aug. 1992, *--

– A METHOD OF REDUCING SYSTEMATIC ERRORS IN CLASSIFICATION PROBLEMS

Louis Lyons Nuclear Physics Lab Oxford LYONS@VXCRNA

In classification problems, the usual way of estimating systematic errors consists of first deciding on an algorithm for determining the quantity of interest, and then seeing the effect on the answer of varying the relevant (Monte Carlo) input constants by suitable amounts. A better way of analysing the data, which can result in reduced systematic errors, is described. Its effect on a toy neural network example is presented; the improvement is striking.

– B-QUARK TAGGING USING NEURAL NETWORKS AND MULTIVARIATE STATISTICAL METHODS

Authors : K. H. Becks, F. Block, J. Drees, P. Langefeld, F. Seidel Physics Department, University of Wuppertal 5600 Wuppertal, Germany

We have studied the b-quark tagging problem from Z0 decays as measured in the DELPHI experiment at LEP using artificial neural networks and conventional multivariate statistical methods. Contrary to the classical approaches we used as input to the neural network very simple variables, closely related to the directly measured particle momenta. The results of the methods are compared and tested for possible complementation.

– COMPARISON OF 3 METHODS TO CLASSIFY HADRONIC EVENTS IN E(+)E(-) COLLISIONS: CUT ON THE MOST DISCRIMINATING VARIABLE, DISCRIMINANT ANALYSIS, NEURAL NETWORK J.Proriol Laboratoire de Physique Corpusculaire Universite Blaise Pascal, Clermont-Ferrand

We compare 3 methods of classification used in high energy physics. We apply these methods to 2 problems: the classification of an hadronic event according to the primary quark when considering the variables computed with the tracks of the whole event or of the most energetic jet.

– MLP-RBF: A COOPERATIVE MULTI-MODULAR NEURAL NETWORK; APPLICATION IN HIGH ENERGY PHYSICS. J.Proriol Laboratoire de Physique Corpusculaire Universite Blaise Pascal, Clermont-Ferrand

We present a cooperative multi-modular neural network architecture: a multi-layer perceptron (MLP) followed by a radial basis function We present a co-

operative multi-modular neural network architecture: a multi-layer perceptron(MLP) followed by a radial basis function neural network(RBF).With ALEPH Monte-Carlo,this architecture was able to outperform both a simple MLP,a modular MLP+LVQ and MLP+RBF trained sequentially.

Chapter 8

Conclusions

The main results derived from the present work are presented here. The first set is devoted to the use of NN in data analysis:

- It has been proved that the output of a multilayer perceptron trained for a classification task is an approximation to the conditional probability that an event belongs to a given class. This assures that NN provide optimal classification in the Bayesian sense.
- Two methods for extracting estimators for the proportions for each class present in a sample have been developed: The matrix and fit methods. Their optimality properties have also been proved. For the fit method it has been shown that one-dimensional projections in the output neuron space (taking into account the bin-to-bin correlation) is the most efficient procedure for extracting the information from the NN output.
- A method for checking the agreement between multidimensional distributions by making a one-dimensional projection fully sensitive to the differences and their NN implementation has been presented.
- A NN topology (encoder) has also been proposed, which is capable of performing non supervised classification in a qualitative fashion. This opens the possibility of extracting qualitative information from the data in a fully model-independent fashion (without Monte Carlo).
- The above results provide a set of consistent and complementary neural tools ready for use in exploratory data analysis. The methods presented match the requirements of HEP data analysis, and can be directly applied substituting more traditional statistical aproches.
- This set of tools has been used succesfully for simultaneously measuring the branching fractions and polarization of τ events from the ALEPH detector showing the power of the tools presented.

Table 8.1: Branching ratios and Polarization results for the 1992 data.

Channel	Br (%)
e	$17.95 \pm 0.23 \pm 0.17$
μ	$17.54 \pm 0.22 \pm 0.12$
π, κ, κ^*	$13.46 \pm 0.27 \pm 1.06$
ρ	$23.34 \pm 0.36 \pm 0.26$
$\pi + 3\pi^0$	$0.36 \pm 0.24 \pm 0.31$
$3\pi(\pi^0)$	$15.52 \pm 0.30 \pm 0.28$
$\pi + 2\pi^0$	$9.93 \pm 0.38 \pm 0.88$
$\pi > 3$	$0.93 \pm 0.19 \pm 0.15$
Polarization	$-0.129 \pm 0.017 \pm 0.029$

This study has also shown that agreement in one-dimensional projections between real and Monte Carlo data can hide unacceptable differences at the multidimensional level. (The Neural Network Agreement Confidence can help in making this check).

The second set deals with the use of NN for pattern recognition:

- The capabilities of Hopfield networks for solving combinatorial optimization problems using mean field annealing has been discussed.
- Previous work on using this method for track finding has been developed in order to make it capable of real-time tasks. A solution based on fixed architecture is presented and tested at software level.
- In collaboration with the Centro Nacional de Microelectronica the idea has been implemented in hardware: first, using an already developed general purpose neural coprocessor and then by designing specialized analog hardware with better time response. This second chip is still in the test phase.

Appendix A

Backpropagation algorithm

Many papers on backpropagation suggest that we need only to use the conventional chain rule for partial derivatives to calculate the derivatives of E with respect to all weights. Under certain conditions, this can be a rigorous approach, but its generality is limited, and requires great care with the boundary conditions; calculations of this sort can easily become confused and erroneous when networks and applications grow complex. Werbos derived in 1972 [14] the concept of ordered systems and ordered derivatives that leads to a more rigorous approach to backpropagation, which is valid also for networks with feedback as well.

Let $E(\{z_i\})$ be an ordered system if E is a function that depends on z_i , where z_i are recursively calculated as $z_i = z_i(\{z_k\})$ with $k < i$. For this system, the calculations must be performed in the rigorous order $z_1, z_2, \dots, z_n, E$. The total impact of z_i on E , E_{z_i} , must take into account the direct effect, and the indirect one through the system of equations that determines z_i . The chain rule for ordered derivatives can be written then [14]

$$E_{z_i} = \frac{\partial E}{\partial z_i} + \sum_{j>i} E_{z_j} \frac{\partial z_j}{\partial z_i}$$

Using this rule, the backpropagation algorithm can be easily derived for any network topology if the neuron update order is provided in advance.

Let the network update equation be

$$o_i = f(I_i)$$

$$I_i = \sum_j w_{ij} o_j$$

and

$$E^p = \frac{1}{2} \sum_i (o_i^p - d_i^p)^2$$

be the error function for pattern p , where i runs over the neurons considered as output. Forgetting the pattern index,

$$E_{I_k} = f'(I_k) E_{o_k}$$

$$E_{o_k} = \frac{\partial E}{\partial o_k} + \sum_{l>k} E_{o_l} \frac{\partial o_l}{\partial o_k}$$

So that

$$E_{o_k} = o_k - d_k$$

if o_k is an output neuron, and

$$E_{o_k} = \sum_{l<k} w_{lk} E_{I_k}$$

otherwise. Finally

$$E_{w_{ij}} = E_{I_i} o_j$$

This set of equations is able to compute all the impacts from the weights in the E function calculating recursively E_{I_i} in the backward direction and then E_{o_k} . For recurrent networks, backward means update in the reverse sequence order.

Appendix B

Heuristics for designing and training multilayer perceptrons

Backpropagation, like other local minimization methods, has very poor convergence properties. Pure gradient descent takes the error function to the closest minimum, which normally is a local one. Only for linearly separable problems it is possible to assure the error function convexity and, thus, the existence of a single global minimum. Another source of problems is the rate of convergence to the minimum (local or global), that can be very slow when some neurons are working in the saturation zone.

In order to eliminate this drawbacks, several hints are presented. Most of them are only empirical rules, loosely related to analytical results; however in general they provide good practical results.

B.1 Input preprocessing

Theoretically, a Neural Network with several layers has to be able to generate an optimal internal representation of the input data. Nevertheless some elemental preprocessing can help the learning process:

- Sign compensation: Shift the input values in order to have the distribution mean near 0. This assure that the sigmoids will be working in the linear zone (where the derivative has a larger value).
- Variance equalization: Scale the input values for equaling the input distribution variance in all input neurons. This prevents from overestimating the effect in the output of a variable with big variance.
- Outsiders elimination: Events with atypical values can distort the estimate of the mean or variance.

Two methods are proposed for implementing the above rules:

1. Scaling between the maximum and the minimum: Make a linear transformation of the input values that transform the maximum (minimum) into +1 (-1)

$$x' = 2 \frac{x - x_{min}}{x_{max} - x_{min}} - 1$$

This procedure works well for flat distributions.

2. Standardization: Make a linear transformation which gives a new input distribution with mean 0 and variance 1. A variant of this procedure is to scale the values transforming $x_{mean} + 3\sigma$ ($x_{mean} - 3\sigma$) into +1 (-1). For the encoder case, this kind of transformation is very useful.

B.2 Network topology

There are no general rules to determine the number of layers or neurons in each layer necessary for solving a problem. Here are some hints that can help in the design:

- Linearly separable problems can be solved using a single perceptron without hidden layers. This is the first topology to test.
- Multilayer perceptrons with one hidden layer can map only monotonic functions. This corresponds to classification problems that are convexly separable. A general rule is that the hidden layer must contain at least $2n + 1$ neurons, where n is the number of input units. This result is derived from Kolmogorov's representation theorem [55].
- A multilayer perceptron with two hidden layers can map any function if there are enough neurons in the hidden layers. Nevertheless, backpropagation becomes less efficient when the number of layers grows.
- The number of degrees of freedom of a network must be carefully chosen to avoid overfitting (overlearning) of the training input.

B.3 Weight initialization

The normal procedure is to initialize randomly the weights with small values. Several trials are recommended in order to overcome the local minima problem [16].

B.4 Weight update.

Pure gradient descent with constant learning rate can lead to a very slow convergence. In order to avoid this difficulty several methods are proposed:

- Add a momentum term to the weight update formula

$$\Delta w_{ij}(t+1) = -\epsilon \frac{\partial E}{\partial w_{ij}} + \alpha \Delta w_{ij}(t)$$

where ϵ is the learning rate and α the so called momentum term. This momentum term averages over last directions of gradient descent and eliminates oscillations. This method is a limiting case of the conjugate gradient method without line search. Good choices are $\epsilon = 0.1$ and $\alpha = 0.9$.

- Adaptive ϵ . This method provides very good results. The idea is to vary dynamically the learning rate in response to the actual behaviour of the error surface. The heuristic rules used are:
 - If the error gradient averaged over the last updates is low increase the learning rate. Maybe the error surface is too flat.
 - If the averaged error is low decrease the learning rate.
 - Increase the learning rate for patterns that have an error greater than the mean error in the last updates.

These rules can be implemented using

$$\epsilon_k = \epsilon_0 \frac{E_k}{\langle E \rangle \langle \nabla^2 E \rangle}$$

where ϵ_k is the learning rate used to update the weights after the presentation of pattern k , E_k is the actual error in classifying the pattern k and the averages are taken over a statistically significant sample of pattern presentations. The new free parameter is ϵ_0 . Initially must be put to a value around 10^{-3} and adaptatively lowered when the mean error does not decrease over several updates.

Appendix C

Mean Field equations for a Hopfield Network.

The Statistical Mechanics of a discrete Hopfield network is given by the partition function

$$Z = \sum_S e^{-E(S)/T} \quad (C.1)$$

where the function E is given by eq 4.2, and the summation runs over all possible neuron configurations $S = (S_1 \dots S_n)$. A proof that is beyond the scope of this work [49] shows that the discrete sum in 10.1 can be replaced by multiple nested integrals over the continuous variables $U_i = (U_1 \dots U_n)$ and $V_i = (V_1 \dots V_n)$ as

$$Z = C \prod_{j=1}^N \int_{-\infty}^{\infty} dV_j \int_{-\infty}^{\infty} dU_j e^{-E'(\vec{V}, \vec{U}, T)} \quad (C.2)$$

where C is a complex constant, $\prod \int$ denote multiple integrand, and

$$E'(\vec{V}, \vec{U}, T) = E(\vec{V})/T + \sum_{i=1}^N (U_i V_i - \log(\cosh(U_i))) \quad (C.3)$$

The integrals in C.2 can be analyzed using a saddle point expansion of $E'(\vec{V}, \vec{U}, T)$ that yields

$$Z = C e^{-E'(\vec{V}_0, \vec{U}_0, T)} (1 + o(N)) \quad (C.4)$$

where $\vec{V}_0$ and $\vec{U}_0$ are the solutions to

$$\frac{\partial E'}{\partial V_i} = 0 \quad (C.5)$$

$$\frac{\partial E'}{\partial U_i} = 0 \quad (C.6)$$

and $o(N)$ is a linear function of the number of neurons. This shows that the contribution to the partition function comes principally from the exponential (E' is also proportional to the number of neurons).

Then, the neural networks dynamics is governed by (from C.4)

$$V_i - \tanh(U_i) = 0 \quad (C.7)$$

$$\frac{1}{T} \frac{\partial E(\vec{V})}{\partial V_i} + U_i = 0 \quad (C.8)$$

that leads using 4.2

$$V_i = \tanh\left(\sum_{j=1}^N T_{ij} V_j / T\right) \quad (C.9)$$

These are the MFT equations. It can be shown that the continuous variables V_i approximate the mean of the discrete neuron variables at a given temperature [50]

$$V_i \approx \langle S_i \rangle_T \quad (C.10)$$

Thus, the statistical (i.e., non-zero temperature) behavior of the neural network in a simulated annealing framework is emulated by the sigmoid updating rule

$$V_i(t + \Delta t) = \tanh\left(\sum_{j=1}^N T_{ij} V_j(t) / T\right) \quad (C.11)$$

In contrast to simulated annealing, which is a stochastic algorithm, MFT is deterministic. Also it is not necessary to use an annealing schedule with the MFT equations. Whereas simulated annealing follows an equilibrium path from high to low temperatures, the MFT equations represent the equilibrium statistics of the neural network at a temperature T .

Replacing U_i in E' with $\tanh^{-1}(V_i)$ one obtains after some algebraic transformations

$$E'(\vec{V}, T) = E(\vec{V})/T + \sum_{i=1}^N \left[(1 + V_i) \frac{1}{2} \log(1 + V_i) + (1 - V_i) \frac{1}{2} \log(1 - V_i) \right] \quad (C.12)$$

Thus, when solving the MFT equations, we are optimizing the balance between the energy (the first term) and the entropy (second term), rather than minimizing the internal energy, which is the case when the update procedure is applied over the discrete neurons.

Appendix D

Sources of information about NN

These is a subset of the the most Frequently Asked Questions (FAQ) in the internet Newsgroup comp.ai.neural-net. The reason for including this Appendix is to provide information for people interested in to apply the methods discussed in this Thesis.

Comments are from people that have contributed .

1. Good introductory literature about Neural Networks
2. Any journals and magazines about Neural Networks
3. The most important conferences concerned with Neural Networks
4. Other sources of information about NNs
5. Freely available software packages for NN simulation
6. Commercial software packages for NN simulation
7. Neural Network hardware
8. Databases for experimentation with NNs

D.1 Good introductory literature about Neural Networks

D.1.1 Books for the beginner:

- Aleksander, I. and Morton, H. (1990). An Introduction to Neural Computing. Chapman and Hall. (ISBN 0-412-37780-2). Comments: "This book seems to be intended for the first year of university education."

- Beale, R. and Jackson, T. (1990). *Neural Computing, an Introduction*. Adam Hilger, IOP Publishing Ltd : Bristol. (ISBN 0-85274-262-2). Comments: "It's clearly written. Lots of hints as to how to get the adaptive models covered to work (not always well explained in the original sources). Consistent mathematical terminology. Covers perceptrons, error-backpropagation, Kohonen self-org model, Hopfield type models, ART, and associative memories."
- Dayhoff, J. E. (1990). *Neural Network Architectures: An Introduction*. Van Nostrand Reinhold: New York. Comments: "Like Wasserman's book, Dayhoff's book is also very easy to understand".
- McClelland, J. L. and Rumelhart, D. E. (1988). *Explorations in Parallel Distributed Processing: Computational Models of Cognition and Perception* (software manual). The MIT Press. Comments: "Written in a tutorial style, and includes 2 diskettes of NN simulation programs that can be compiled on MS-DOS or Unix (and they do too !); "The programs are pretty reasonable as an introduction to some of the things that NNs can do."; "There are *two* editions of this book. One comes with disks for the IBM PC, the other comes with disks for the Macintosh".
- McCord Nelson, M. and Illingworth, W.T. (1990). *A Practical Guide to Neural Nets*. Addison-Wesley Publishing Company, Inc. (ISBN 0-201-52376-0). Comments: "No formulas at all"; "It does not have much detailed model development (very few equations), but it does present many areas of application. It includes a chapter on current areas of research. A variety of commercial applications is discussed in chapter 1. It also includes a program diskette with a fancy graphical interface (unlike the PDP diskette)".
- Orchard, G.A. & Phillips, W.A. (1991). *Neural Computation: A Beginner's Guide*. Lawrence Earlbaum Associates: London. Comments: "Short user-friendly introduction to the area, with a non-technical flavour.
- Wasserman, P. D. (1989). *Neural Computing: Theory & Practice*. Van Nostrand Reinhold: New York. (ISBN 0-442-20743-3) Comments: "Wasserman flatly enumerates some common architectures from an engineer's perspective ('how it works') without ever addressing the underlying fundamentals ('why it works') - important basic concepts such as clustering, principal components or gradient descent are not treated. It's also full of errors, and unhelpful diagrams drawn with what appears to be PCB board layout software from the '70s. For anyone who wants to do active research in the field I consider it quite inadequate"; "Okay, but too shallow"; "Quite easy to understand"; "The best bedtime reading for Neural Networks. I have given this book to numerous colleagues who want to know NN basics, but who never plan to implement anything."

D.1.2 The classics:

- Kohonen, T. (1984). *Self-organization and Associative Memory*. Springer-Verlag: New York. (2nd Edition: 1988; 3rd edition: 1989). Comments: "The section on Pattern mathematics is excellent."
- Rumelhart, D. E. and McClelland, J. L. (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* (volumes 1 & 2). The MIT Press. Comments: "As a computer scientist I found the two Rumelhart and McClelland books really heavy going and definitely not the sort of thing to read if you are a beginner."; "It's quite readable, and affordable (about \$65 for both volumes)."; "THE Connectionist bible."

D.1.3 The best:

- Hecht-Nielsen, R. (1990). *Neurocomputing*. Addison Wesley. Comments: "A good book", "comprises a nice historical overview and a chapter about NN hardware. Well structured prose. Makes important concepts clear."
- Hertz, J., Krogh, A., and Palmer, R. (1991). *Introduction to the Theory of Neural Computation*. Addison-Wesley: Redwood City, California. ISBN 0-201-50395-6 (hardbound) and 0-201-51560-1 (paperbound) Comments: "My first impression is that this one is by far the best book on the topic. And it's below \$30 for the paperback."; "Well written, theoretical (but not overwhelming)"; "It provides a good balance of model development, computational algorithms, and applications. The mathematical derivations are especially well done"; "Nice mathematical analysis on the mechanism of different learning algorithms"; "It is NOT for mathematical beginner. If you don't have a good grasp of higher level math, this book can be really tough to get through."

D.1.4 Introductory journal articles:

- Hinton, G. E. (1989). Connectionist learning procedures. *Artificial Intelligence*, Vol. 40, pp. 185-234. Comments: "One of the better neural networks overview papers, although the distinction between network topology and learning algorithm is not always very clear. Could very well be used as an introduction to neural networks."
- Knight, K. (1990). Connectionist, Ideas and Algorithms. *Communications of the ACM*. November 1990. Vol.33 nr.11, pp 59-74. Comments: "A good article, while it is for most people easy to find a copy of this journal."
- Kohonen, T. (1988). An Introduction to Neural Computing. *Neural Networks*, vol. 1, no. 1. pp. 3-16. Comments: "A general review".

D.1.5 Not-quite-so-introductory literature:

- Anderson, J. A. and Rosenfeld, E. (Eds). (1988). *Neurocomputing: Foundations of Research*. The MIT Press: Cambridge, MA. Comments: "An expensive book, but excellent for reference. It is a collection of reprints of most of the major papers in the field.";
- Anderson, J. A., Pellionisz, A. and Rosenfeld, E. (Eds). (1990). *Neurocomputing 2: Directions for Research*. The MIT Press: Cambridge, MA. Comments: "The sequel to their well-known Neurocomputing book."
- Caudill, M. and Butler, C. (1990). *Naturally Intelligent Systems*. MIT Press: Cambridge, Massachusetts. (ISBN 0-262-03156-6). Comments: "I guess one of the best books I read"; "May not be suited for people who want to do some research in the area".
- Khanna, T. (1990). *Foundations of Neural Networks*. Addison-Wesley: New York. Comments: "Not so bad (with a page of erroneous formulas (if I remember well), and #hidden layers isn't well described)."; "Khanna's intention in writing his book with math analysis should be commended but he made several mistakes in the math part".
- Levine, D. S. (1990). *Introduction to Neural and Cognitive Modeling*. Lawrence Erlbaum: Hillsdale, N.J. Comments: "Highly recommended".
- Lippmann, R. P. (April 1987). An introduction to computing with neural nets. *IEEE Acoustics, Speech, and Signal Processing Magazine*. vol. 2, no. 4, pp 4-22. Comments: "Much acclaimed as an overview of neural networks, but rather inaccurate on several points. The categorization into binary and continuous-valued input neural networks is rather arbitrary, and may work confusing for the unexperienced reader. Not all networks discussed are of equal importance."
- Maren, A., Harston, C. and Pap, R., (1990). *Handbook of Neural Computing Applications*. Academic Press. ISBN: 0-12-471260-6. (451 pages) Comments: "They cover a broad area"; "Introductory with suggested applications implementation".
- Pao, Y. H. (1989). *Adaptive Pattern Recognition and Neural Networks* Addison-Wesley Publishing Company, Inc. (ISBN 0-201-12584-6) Comments: "An excellent book that ties together classical approaches to pattern recognition with Neural Nets. Most other NN books do not even mention conventional approaches."
- Rumelhart, D. E., Hinton, G. E. and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, vol 323 (9 October), pp.

533-536. Comments: "Gives a very good potted explanation of backprop NN's. It gives sufficient detail to write your own NN simulation."

- Simpson, P. K. (1990). Artificial Neural Systems: Foundations, Paradigms, Applications and Implementations. Pergamon Press: New York. Comments: "Contains a very useful 37 page bibliography. A large number of paradigms are presented. On the negative side the book is very shallow. Best used as a complement to other books".
- Zeidenberg, M. (1990). Neural Networks in Artificial Intelligence. Ellis Horwood, Ltd., Chichester. Comments: "Gives the AI point of view".
- Zornetzer, S. F., Davis, J. L. and Lau, C. (1990). An Introduction to Neural and Electronic Networks. Academic Press. (ISBN 0-12-781881-2) Comments: "Covers quite a broad range of topics (collection of articles/papers)."; "Provides a primer-like introduction and overview for a broad audience, and employs a strong interdisciplinary emphasis".

D.2 Any journals and magazines about Neural Networks

Title: Neural Networks

Publish: Pergamon Press

Address: Pergamon Journals Inc., Fairview Park, Elmsford, New York 10523, USA and Pergamon Journals Ltd. Headington Hill Hall, Oxford OX3, 0BW, England

Freq.: 6 issues/year (vol. 1 in 1988)

Cost/Yr: Free with INNS membership (\$45?), Individual \$65, Institution \$175

ISSN : 0893-6080

Remark: Official Journal of International Neural Network Society (INNS).

Contains Original Contributions, Invited Review Articles, Letters to Editor, Invited Book Reviews, Editorials, Announcements and INNS News, Software Surveys. This is probably the most popular NN journal. (Note: Remarks supplied by Mike Plonski "plonski@aero.org")

Title: Neural Computation

Publish: MIT Press

Address: MIT Press Journals, 55 Hayward Street Cambridge,

Freq.: Quarterly (vol. 1 in 1989)

Cost/Yr: Individual \$45, Institution \$90, Students \$35; Add \$9 Outside USA

ISSN : 0899-7667

Remark: Combination of Reviews (10,000 words), Views (4,000 words) and Letters

(2,000 words). I have found this journal to be of outstanding quality. (Note: Remarks supplied by Mike Plonski "plonski@aero.org")

Title: IEEE Transaction on Neural Networks

Publish: Institute of Electrical and Electronics Engineers (IEEE)

Address: IEEE Service Center, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ, 08855-1331 USA. Tel: (201) 981-0060

Cost/Yr: \$10 for Members belonging to participating IEEE societies

Freq.: Quarterly (vol. 1 in March 1990)

Remark: Devoted to the science and technology of neural networks which disclose significant technical knowledge, exploratory developments and applications of neural networks from biology to software to hardware. Emphasis is on artificial neural networks. Specific aspects include self organizing systems, neurobiological connections, network dynamics and architecture, speech recognition, electronic and photonic implementation, robotics and controls. Includes Letters concerning new research results. (Note: Remarks are from journal announcement)

Title: International Journal of Neural Systems

Publish: World Scientific Publishing

Address: USA: World Scientific Publishing Co., 687 Hartwell Street, Teaneck, NJ 07666. Tel: (201) 837-8858; Europe: World Scientific Publishing Co. Pte. Ltd., 73 Lynton Mead, Totteridge, London N20-8DH, England. Tel: (01) 4462461; Other: World Scientific Publishing Co. Pte. Ltd., Farrer Road, P.O. Box 128, Singapore 9128. Tel: 2786188

Freq.: Quarterly (Vol. 1 in 1990?)

Cost/Yr: Individual \$42, Institution \$88 (plus \$9-\$17 for postage)

ISSN : 0129-0657 (IJNS)

Remark: The International Journal of Neural Systems is a quarterly journal which covers information processing in natural and artificial neural systems. It publishes original contributions on all aspects of this broad subject which involves physics, biology, psychology, computer science and engineering. Contributions include research papers, reviews and short communications. The journal presents a fresh undogmatic attitude towards this multidisciplinary field with the aim to be a forum for novel ideas and improved understanding of collective and cooperative phenomena with computational capabilities. (Note: Remarks supplied by B. Lautrup (editor), "LAUTRUPReview is reported to be very slow.

Title: Neural Network News

Publish: AIWeek Inc.

Address: Neural Network News, 2555 Cumberland Parkway, Suite 299, Atlanta, GA 30339 USA. Tel: (404) 434-2187

Freq.: Monthly (beginning September 1989)

Cost/Yr: USA and Canada \$249, Elsewhere \$299

Remark: Commercial Newsletter

Title: Network: Computation in Neural Systems

Publish: IOP Publishing Ltd

Address: Europe: IOP Publishing Ltd, Techno House, Redcliffe Way, Bristol BS1 6NX, UK; IN USA: American Institute of Physics, Subscriber Services 500 Sunnyside Blvd., Woodbury, NY 11797-2999

Freq.: Quarterly (1st issue 1990)

Cost/Yr: USA: \$180, Europe: 110 pounds

Remark: Description: "a forum for integrating theoretical and experimental findings across relevant interdisciplinary boundaries." Contents: Submitted articles reviewed by two technical referees paper's interdisciplinary format and accessibility." Also Viewpoints and Reviews commissioned by the editors, abstracts (with reviews) of articles published in other journals, and book reviews. Comment: While the price discourages me (my comments are based upon a free sample copy), I think that the journal succeeds very well. The highest density of interesting articles I have found in any journal. (Note: Remarks supplied by brandt kehoe "kehoe@csufres.CSUFresno.EDU")

Title: Connection Science: Journal of Neural Computing, Artificial Intelligence and Cognitive Research

Publish: Carfax Publishing

Address: Europe: Carfax Publishing Company, P. O. Box 25, Abingdon, Oxfordshire OX14 3UE, UK. USA: Carfax Publishing Company, 85 Ash Street, Hopkinton, MA 01748

Freq.: Quarterly (vol. 1 in 1989)

Cost/Yr: Individual \$82, Institution \$184, Institution (U.K.) 74 pounds

Title: International Journal of Neural Networks

Publish: Learned Information

Freq.: Quarterly (vol. 1 in 1989)

Cost/Yr: 90 pounds

ISSN : 0954-9889

Remark: The journal contains articles, a conference report (at least the issue I have), news and a calendar. (Note: remark provided by J.R.M. Smits "anjos@sci.kun.nl")

Title: Concepts in Neuroscience

Publish: World Scientific Publishing

Address: Same Address (?) as for International Journal of Neural Systems

Freq.: Twice per year (vol. 1 in 1989)

Remark: Mainly Review Articles(?)
(Note: remarks by Osamu Saito "saito@nttica.NTT.JP")

Title: International Journal of Neurocomputing
Publish: ecn Neurocomputing GmbH
Freq.: Quarterly (vol. 1 in 1989)
Remark: Commercial journal, not the academic periodicals (Note: remarks by Osamu Saito "saito@nttica.NTT.JP") Review has been reported to be fast (less than 3 months)

Title: Neurocomputers
Publish: Gallifrey Publishing
Address: Gallifrey Publishing, PO Box 155, Vicksburg, Michigan, 49097, USA Tel: (616) 649-3772
Freq. Monthly (1st issue 1987?)
ISSN : 0893-1585
Editor: Derek F. Stubbs
Cost/Yr: \$32 (USA, Canada), \$48 (elsewhere)
Remark: I only have one exemplar so I cannot give you much detail about the contents. It is a very small one (12 pages) but it has a lot of (short) information in it about e.g. conferences, books, (new) ideas etc. I don't think it is very expensive but I'm not sure. (Note: remark provided by J.R.M. Smits "anjos@sci.kun.nl")

Title: JNNS Newsletter (Newsletter of the Japan Neural Network Society)
Publish: The Japan Neural Network Society
Freq.: Quarterly (vol. 1 in 1989)
Remark: (IN JAPANESE LANGUAGE) Official Newsletter of the Japan Neural Network Society(JNNS) (Note: remarks by Osamu Saito "saito@nttica.NTT.JP")

Title: Neural Networks Today
Remark: I found this title in a bulletin board of october last year. It was a message of Tim Pattison, timpatt@augean.OZ (Note: remark provided by J.R.M. Smits "anjos@sci.kun.nl")

Title: Neural Computing and Applications
Freq.: Quarterly
Publish: Springer Verlag
Cost/yr: 120 Pounds
Remark: Is the journal of the Neural Computing Applications Forum. Publishes original research and other information in the field of practical applications of neural computing.

D.3 The most important conferences concerned with Neural Networks

1. Dedicated Neural Network Conferences:

- (a) Neural Information Processing Systems (NIPS) Annually in Denver, Colorado; late November or early December
- (b) International Joint Conference on Neural Networks (IJCNN) co-sponsored by INNS and IEEE
- (c) Annual Conference on Neural Networks (ACNN)
- (d) International Conference on Artificial Neural Networks (ICANN) Annually in Europe(?), 1992 in Brighton Major conference of European Neur. Netw. Soc. (ENNS)

2. Other Conferences

- (a) International Joint Conference on Artificial Intelligence (IJCAI)
- (b) Intern. Conf. on Acustics, Speech and Signal Processing (ICASSP)
- (c) Annual Conference of the Cognitive Science Society

3. Pointers to Conferences

- (a) The journal "Neural Networks" has a long list of conferences, workshops and meetings in each issue. This is quite interdisciplinary.
- (b) There is a regular posting on comp.ai.neural-nets from Paultje Bakker: "Upcoming Neural Network Conferences", which lists names, dates, locations, contacts, and deadlines.

D.4 Other sources of information about NNs

- Neuron Digest Internet Mailing List. From the welcome blurb: "Neuron-Digest is a list (in digest form) dealing with all aspects of neural networks (and any type of network or neuromorphic system)" Moderated by Peter Marvit. To subscribe, send email to neuron-request@cattell.psych.upenn.edu comp.ai.neural-net readers also find the messages in that newsgroup in the form of digests.

- Usenet groups comp.ai.neural-nets and comp.theory.self-org-sys There is a periodic posting on comp.ai.neural-nets sent by srctran@world.std.com (Gregory Aharonian) about Neural Network patents.
- Central Neural System Electronic Bulletin Board Modem: 509-627-6CNS; Sysop: Wesley R. Elsberry; P.O. Box 1187, Richland, WA 99352; Available through FidoNet, RBBS-Net, and other EchoMail compatible bulletin board systems as NEURAL_NET echo.
- Neural ftp archive site funic.funet.fi Is administrating a large collection of neural network papers and software at the Finnish University Network file archive site funic.funet.fi in directory /pub/sci/neural Contains all the public domain software and papers that they have been able to find. ALL of these files have been transferred from FTP sites in U.S. and are mirrored about every 3 months at fastest. Contact: magi@funic.funet.fi or magi@utu.fi
- USENET newsgroup comp.org.issnnet Forum for discussion of academic/student-related issues in NNs, as well as information on ISSNNet (see A13) and its activities.
- AI CD-ROM Network Cybernetics Corporation produces the "AI CD-ROM". It is an ISO-9660 format CD-ROM and contains a large assortment of software related to artificial intelligence, artificial life, virtual reality, and other topics. Programs for OS/2, MS-DOS, Macintosh, UNIX, and other operating systems are included. Research papers, tutorials, and other text files are included in ASCII, RTF, and other universal formats. The files have been collected from AI bulletin boards, Internet archive sites, University computer departments, and other government and civilian AI research organizations. Network Cybernetics Corporation intends to release annual revisions to the AI CD-ROM to keep it up to date with current developments in the field. The AI CD-ROM includes collections of files that address many specific AI/AL topics including: [... some stuff deleted...] - Neural Networks: Source code and executables for many different platforms including Unix, DOS, and Macintosh. ANN development tools, example networks, sample data, and tutorials are included. A complete collection of Neural Digest is included as well. [... lots of stuff deleted...] The AI CD-ROM may be ordered directly by check, money order, bank draft, or credit card from: Network Cybernetics Corporation 4201 Wingren Road Suite 202 Irving, TX 75062-2763 Tel 214/650-2002 Fax 214/650-1929 The cost is \$129 per disc + shipping (\$5/disc domestic or \$10/disc foreign)

D.5 Freely available software packages for NN simulation

- Rochester Connectionist Simulator
A quite versatile simulator program for arbitrary types of neural nets. Comes with a backprop package and a X11/Sunview interface.
anonymous FTP from cs.rochester.edu (192.5.53.209)
directory : pub/simulator
files: README (8 KB)
(documentation:) rcs_v4.2.justdoc.tar.Z (1.6 MB)
(source code:) rcsi_v4.2.justsrc.tar.Z (1.4 MB)

- UCLA-SFINX
ftp 131.179.16.6 (retina.cs.ucla.edu)
Name: sfinxftp
Password: joshua
directory: pub/
files : README
sfinx_v2.0.tar.Z
Email info request : sfinx@retina.cs.ucla.edu

- NeurDS
Simulator for DEC systems supporting VT100 terminal. OR anonymous ftp
gatekeeper.dec.com [16.1.0.2] directory: pub/DEC
file: NeurDS031.tar.Z (please check may be NeurDSO31.tar.Z)

- PlaNet5.7 (also known as SunNet)
ftp 133.15.240.3 (tutserver.tut.ac.jp)
pub/misc/PlaNet5.7.tar.Z
or
ftp 128.138.240.1 (boulder.colorado.edu)
pub/generic-sources/PlaNet5.7.tar.Z (also the old PlaNet5.6.tar.Z)
A popular connectionist simulator with versions to run under X Windows, and non-graphics terminals created by Yoshiro Miyata (Chukyo Univ., Japan). 60-page User's Guide in Postscript. Send any questions to miyata@sccs.chukyo-u.ac.jp

- GENESIS
anonymous ftp 131.215.135.64 (genesis.cns.caltech.edu)
Register first via telnet genesis.cns.caltech.edu login as: genesis

- Mactivation
anonymous ftp from bruno.cs.colorado.edu [128.138.243.151]
directory: /pub/cs/misc
file: Mactivation-3.3.sea.hqx

- CMU Connectionist Archive
There is a lisp backprop simulator in the connectionist archive.
unix> ftp b.gp.cs.cmu.edu (or 128.2.242.8)
Name: ftpguest
Password: cmunix
ftp> cd connectionists/archives
ftp> get backprop.lisp

- Cascade Correlation Simulator
There is a LISP and C version of the simulator based on Scott Fahlman's Cascade Correlation algorithm, who also created the LISP version. The C version was created by Scott Crowder.
Anonymous ftp from pt.cs.cmu.edu (or 128.2.254.155)
directory /afs/cs/project/connect/code
files cascor1.lisp (56 KB)
cascor1.c (108 KB)

- Quickprop
A variation of the back-propagation algorithm developed by Scott Fahlman. A LISP and C version can be obtained in the same directory as the cascade correlation simulator above. (25 KB)

- DartNet
DartNet is a Macintosh-based Neural Network Simulator. It makes full use of the Mac's graphical interface, and provides a number of powerful tools for building, editing, training, testing and examining networks. This program is available by anonymous ftp from dartvax.dartmouth.edu [129.170.16.4] as /pub/mac/dartnet.sit.hqx (124 KB) Copies may also be obtained through email from bharucha@dartmouth.edu. Along with a number of interface improvements and feature additions, v2.0 is an extensible simulator. That is, new network architectures and learning algorithms can be added to the system by writing small XCMD-like CODE resources called nDEF's ("Network Definitions"). A number of such architectures are included with v2.0, as well as header files for creating new nDEF's. Contact: sean@coos.dartmouth.edu (Sean P. Nolan)

- SNNS

"Stuttgart Neural Network Simulator" from the University of Stuttgart, Germany. A luxurious simulator for many types of nets; with X11 interface: Graphical 2D and 3D topology editor/visualizer, training visualisation, etc. Currently supports backpropagation (vanilla, online, with momentum term and flat spot elimination, batch, time delay), counterpropagation, quickprop, backpercolation 1, generalized radial basis functions (RBF), RProp, ART1, ART2, ARTMAP, Cascade Correlation, Recurrent Cascade Correlation, Dynamic LVQ, Backpropagation through time (for recurrent networks), batch backpropagation through time (for recurrent networks), Quickpropagation through time (for recurrent networks), and is user-extendable.

ftp: ftp.informatik.uni-stuttgart.de [129.69.211.2]

directory /pub/SNNS

file SNNSv3.0.tar.Z OR SNNSv3.0.tar.Za[a-d] (826 KB)

manual SNNSv2.1.Manual.ps.Z (1270 KB)

SNNSv2.1.Readme (8052 Bytes)

- Aspirin/MIGRAINES

Aspirin/MIGRAINES 6.0 consists of a code generator that builds neural network simulations by reading a network description (written in a language called "Aspirin") and generates a C simulation. An interface (called "MIGRAINES") is provided to export data from the neural network to visualization tools. The system has been ported to a large number of platforms. The goal of Aspirin is to provide a common extendible front-end language and parser for different network paradigms. The MIGRAINES interface is a terminal based interface that allows you to open Unix pipes to data in the neural network. This replaces the NeWS1.1 graphical interface in version 4.0 of the Aspirin/MIGRAINES software. The new interface is not as simple to use as the version 4.0 interface but is much more portable and flexible. The MIGRAINES interface allows users to output neural network weight and node vectors to disk or to other Unix processes. Users can display the data using either public or commercial graphics/analysis tools. Example filters are included that convert data exported through MIGRAINES to formats readable by Gnuplot 3.0, Matlab, Mathematica, and xgobi. The software is available from two FTP sites: CMU's simulator collection on "pt.cs.cmu.edu" (128.2.254.155) in /afs/cs/project/connect/code/am6.tar.Z". and UCLA's cognitive science machine "ftp.cognet.ucla.edu" (128.97.50.19) in alexis/am6.tar.Z The compressed tar file is a little less than 2 megabytes.

- Adaptive Logic Network kit

Available from menaik.cs.ualberta.ca. This package differs from the traditional nets in that it uses logic functions rather than floating point; for many

tasks, ALN's can show many orders of magnitude gain in training and performance speed.

Anonymous ftp from menaik.cs.ualberta.ca [129.128.4.241]

README /pub/atree/atree.readme (7 KB)

unix source code and examples: /pub/atree/atree2.tar.Z (145 KB)

Postscript documentation: /pub/atree/atree2.ps.Z (76 KB)

MS-Windows 3.x executable: /pub/atree/a27exe.exe (412 KB)

MS-Windows 3.x source code: /pub/atree/atree27.exe (572 KB)

– NeuralShell

Available from FTP site quanta.eng.ohio-state.edu (128.146.35.1) in directory "pub/NeuralShell", filename "NeuralShell.tar".

- PDP The PDP simulator package is available via anonymous FTP at nic.funet.fi (128.214.6.100) in /pub/sci/neural/sims/pdp.tar.Z (0.2 MB) The simulator is also available with the book "Explorations in Parallel Distributed Processing: A Handbook of Models, Programs, and Exercises" by McClelland and Rumelhart. MIT Press, 1988. Comment: "This book is often referred to as PDP vol III which is a very misleading practice! The book comes with software on an IBM disk but includes a makefile for compiling on UNIX systems. The version of PDP available at nic.funet.fi seems identical to the one with the book except for a bug in bp.c which occurs when you try to run a script of PDP commands using the DO command. This can be found and fixed easily."

– Xerion

Xerion is available via anonymous ftp from ftp.cs.toronto.edu in the directory /pub/xerion. xerion-3.1.ps.Z (153 kB) and xerion-3.1.tar.Z (1322 kB) plus several concrete simulators built with xerion (about 40 kB each). Xerion runs on SGI and Sun machines and uses X Windows for graphics. The software contains modules that implement Back Propagation, Recurrent Back Propagation, Boltzmann Machine, Mean Field Theory, Free Energy Manipulation, Hard and Soft Competitive Learning, and Kohonen Networks. Sample networks built for each of the modules are also included. Contact: xerion@ai.toronto.edu

– Neocognitron simulator

An implementation is available for anonymous ftp at 128.194.15.32 tamsun.tamu.edu as

/pub/neocognitron.Z.tar or

129.12.21.7 unix.hensa.ac.uk as

/pub/uunet/pub/ai/neural/neocognitron.tar.Z

The simulator is written in C and comes with a list of references

which are necessary to read to understand the specifics of the implementation.

The unsupervised version is coded without (!) C-cell inhibition.

– Multi-Module Neural Computing Environment (MUME)

MUME is a simulation environment for multi-modules neural computing. It provides an object oriented facility for the simulation and training of multiple nets with various architectures and learning algorithms. MUME includes a library of network architectures including feedforward, simple recurrent, and continuously running recurrent neural networks. Each architecture is supported by a variety of learning algorithms. MUME can be used for large scale neural network simulations as it provides support for learning in multi-net environments. It also provide pre- and post-processing facilities. The modules are provided in a library. Several "front-ends" or clients are also available. MUME can be used to include non-neural computing modules (decision trees, ...) in applications. The software is written in 'C' and is being used on Sun and DEC workstations. Efforts are underway to port it to the Fujitsu VP2200 vector processor using the VCC vectorising C compiler. MUME is made available to research institutions on media/doc/postage cost arrangements. Contact: Marwan Jabri, SEDAL, Sydney University Electrical Engineering, NSW 2006 Australia, marwan@sedal.su.oz.au

– LVQ_PAK, SOM_PAK

These are packages for Learning Vector Quantization and Self-Organizing Maps, respectively. They have been built by the LVQ/SOM Programming Team of the Helsinki University of Technology, Laboratory of Computer and Information Science, Rakentajanaukio 2 C, SF-02150 Espoo, FINLAND There are versions for Unix and MS-DOS available from cochlea.hut.fi (130.233.168.48) in

/pub/lvqi_pak/lvqi_pak-2.1.tar.Z (340 kB, Unix)

/pub/lvq_pak/lvq_2r1.exe (310 kB, MS-DOS self-extract archive)

/pub/som_pak/som_pak-1.1.tar.Z (246 kB, Unix)

/pub/som_pak/som_plr1.exe (215 kB, MS-DOS self-extract archive)

– SESAME

(Software Environment for the Simulation of Adaptive Modular Systems)

SESAME is a prototypical software implementation which facilitates

Object-oriented building blocks approach.

Contains a large set of C++ classes useful for neural nets, neurocontrol and pattern recognition. No C++ classes can be used as stand alone, though!

C++ classes include CartPole, nondynamic two-robot arms, Lunar Lander, Backpropagation, Feature Maps, Radial Basis Functions, TimeWindows, Fuzzy Set Coding, Potential Fields, Pandemonium, and diverse utility building blocks.

A kernel which is the framework for the C++ classes and allows run-time manipulation, construction, and integration of arbitrary complex and hybrid

experiments.

Currently no graphic interface for construction, only for visualization.

Platform is SUN4, XWindows

Unfortunately no reasonable good introduction has been written until now.

We hope to have something soon. For now we provide papers (eg. NIPS-92), a reference manual (>220 pages), source code (ca. 35.000 lines of code), and a SUN4-executable by ftp only. Sesame and its description is available for anonymous ftp on

ftp ftp.gmd.de [129.26.8.90] in the directories gmd/as/sesame and gmd/as/paper
Questions please to sesame-request@gmd.de There is only very limited support available. Currently we can not handle many users.

– Nevada Backpropagation (NevProp)

NevProp is a user-friendly backpropagation program written in C for UNIX, Macintosh, and DOS. The original version was Quickprop 1.0 by Scott Fahlman, as translated from Common Lisp into C by Terry Regier. The quickprop algorithm itself was not substantively changed, but we inserted options to force gradient descent (per-epoch or per-pattern) and added generalization & stopped training, c index, and interface enhancements. The most updated version of NevProp will be made available by anonymous ftp from the University of Nevada, Reno:

"unssun.scs.unr.edu" [134.197.10.128]
directory "pub/goodman/nevpropdir"

Limited support is available from Phil Goodman (goodman@unr.edu), University of Nevada Center for Biomedical Research.

For some of these simulators there are user mailing lists. Get the packages and look into their documentation for further info.

If you are using a small computer (PC, Mac, etc.) you may want to have a look at the Central Neural System Electronic Bulletin Board (see Answer 14) Modem: 509-627-6CNS; Sysop: Wesley R. Elsberry; P.O. Box 1187, Richland, WA 99352; welsberr@sandbox.kenn.wa.us There are lots of small simulator packages, the CNS ANNSIM file set. There is an ftp mirror site for the CNS ANNSIM file set at me.uta.edu (129.107.2.20) in the /pub/neural directory. Most ANN offerings are in /pub/neural/annsim.

D.6 Commercial software packages for NN simulation

The Number 1 of each volume of the journal "Neural Networks" has a list of some dozens of commercial suppliers of Neural Network things: Software, Hardware,

Support, Programming, Design and Service.

Here is a naked list of names of Simulators running on PC (and, partly, some other platforms, too):

1. NeuralWorks Professional 2+ (NeuralWare)
2. AIM
3. BrainMaker Professional
4. Brain Cel
5. Neural Desk
6. Neural Case
7. Neuro Windows
8. Explorenet 3000
9. NeuroShell (Systems Group)
10. DynaMind, DynaMind Developer (NeuroDynamX)

Another thing:

MATLAB Neural Network Toolbox (for use with Matlab 4.x) Contact: The MathWorks, Inc. Phone: 508-653-1415 24 Prime Park Way FAX: 508-653-2997 Natick, MA 01760 email: info@mathworks.com

(Comment by Richard Andrew Miles Outerbridge, RAMO@UVPHYS.PHYS.UVIC.CA)
Matlab is spreading like hotcakes (and the educational discounts are very impressive). The newest release of Matlab (4.0) answers the question "if you could only program in one language what would it be?". The neural network toolkit is worth getting for the manual alone. Matlab is available with lots of other toolkits (signal processing, optimization, etc.) but I don't use them much - the main package is more than enough. The nice thing about the Matlab approach is that you can easily interface the neural network stuff with anything else you are doing.

D.7 Neural Network hardware

The Number 1 of each volume of the journal "Neural Networks" has a list of some dozens of suppliers of Neural Network support: Software, Hardware, Support, Programming, Design and Service.

Here is a list of companies contributed by xli@computing-maths.cardiff.ac.uk:

- HNC, INC.
5501 Oberlin Drive
San Diego
California 92121
(619) 546-8877
and a second address at
7799 Leesburg Pike, Suite 900
Falls Church, Virginia
22043
(703) 847-6808
Note: Australian Dist.: Unitronics
Tel : (09) 4701443
Contact: Martin Keye
HNC markets:
'Image Document Entry Processing Terminal' - it recognises handwritten documents and converts the info to ASCII.
'ExploreNet 3000' - a NN demonstrator
'Anza/DP Plus' - a Neural Net board with 25MFlop or 12.5M peak interconnects per second.

- SAIC (Science Application International Corporation)
10260 Campus Point Drive
MS 71, San Diego
CA 92121
(619) 546 6148
Fax: (619) 546 6736

- Micro Devices
30 Skyline Drive
Lake Mary
FL 32746-6201
(407) 333-4379
MicroDevices makes MD1220 - 'Neural Bit Slice'
Each of the products mentioned so far have very different usages. Although this sounds similar to Intel's product, the architectures are not.

- Intel Corp
2250 Mission College Blvd
Santa Clara, Ca 95052-8125
Attn ETANN, Mail Stop SC9-40
(408) 765-9235

Intel is making an experimental chip:

80170NW - Electrically trainable Analog Neural Network (ETANN)

It has 64 'neurons' on it - almost fully internally connected and the chip can be put in an hierarchical architecture to do 2 Billion interconnects per second.

Support software has already been made by

California Scientific Software

10141 Evening Star Dr 6

Grass Valley, CA 95945-9051

(916) 477-7481

Their product is called 'BrainMaker'.

- NeuralWare, Inc

Penn Center West

Bldg IV Suite 227

Pittsburgh

PA 15276

They only sell software/simulator but for many platforms.

- Tubb Research Limited

7a Lavant Street

Peterfield

Hampshire

GU32 2EL

United Kingdom

Tel: +44 730 60256

- Adaptive Solutions Inc

1400 NW Compton Drive

Suite 340

Beaverton, OR 97006

U. S. A.

Tel: 503 - 690 - 1236 FAX: 503 - 690 - 1249

- NeuroDynamX, Inc.

4730 Walnut St., Suite 101B

Boulder, CO 80301

Voice: (303) 442-3539 Fax: (303) 442-2854

Internet: techsupport@ndx.com

NDX sells a number neural network hardware products:

NDX Neural Accelerators: a line of i860-based accelerator cards for the PC that give up to 45 million connections per second for use with the DynaMind

neural network software.

iNNTS: Intel's 80170NX (ETANN) Neural Network Training System. NDX's president was one of the co-designers of this chip.

And here is an incomplete list of Neurocomputers (provided by jon@kongle.idt.unit.no (Jon Gunnar Solheim)):

Overview over known Neural Computers with their newest known reference.

Digital

Special Computers

AAP-2 Takumi Watanabe, Yoshi Sugiyama, Toshio Kondo, and Yoshihiro Kitamura. Neural network simulation on a massively parallel cellular array processor: AAP-2. In International Joint Conference on Neural Networks, 1989.

ANNA B.E.Boser, E.Sackinger, J.Bromley, Y.leChun, and L.D.Jackel. Hardware Requirements for Neural Network Pattern Classifiers. In *IEEE Micro*, 12(1), pages 32-40, February 1992.

Analog Neural Computer Paul Mueller et al. Design and performance of a prototype analog neural computer. In *Neurocomputing*, 4(6):311-323, 1992.

APx – Array Processor Accelerator

F.Pazienti.

Neural networks simulation with array processors. In *Advanced Computer Technology, Reliable Systems and Applications; Proceedings of the 5th Annual Computer Conference*, pages 547-551. IEEE Comput. Soc. Press, May 1991. ISBN: 0-8186-2141-9.

ASP – Associative String Processor

A.Krikelis.

A novel massively associative processing architecture for the implementation artificial neural networks.

In *1991 International Conference on Acoustics, Speech and Signal Processing*, volume 2, pages 1057-1060. IEEE Comput. Soc. Press, May 1991.

BSP400 Jan N.H. Heemskerk, Jacob M.J. Murre, Jaap Hoekstra, Leon H.J.G. Kemna, and Patrick T.W. Hudson. The bsp400: A modular neurocomputer assembled from 400 low-cost microprocessors. In *International Conference on Artificial Neural Networks*. Elsevier Science, 1991.

BLAST

J.G.Elias, M.D.Fisher, and C.M.Monemi.

A multiprocessor machine for large-scale neural network simulation. In *IJCNN91-Seattle: International Joint Conference on Neural Networks*, volume 1, pages 469-474. IEEE Comput. Soc. Press, July 1991. ISBN: 0-7883-0164-1.

CNAPS Neurocomputer

H. McCartor

Back Propagation Implementation on the Adaptive Solutions CNAPS Neurocomputer.

In *Advances in Neural Information Processing Systems*, 3, 1991.

MA16 – Neural Signal Processor U. Ramacher, J. Beichter, and N. Bruls.

Architecture of a general-purpose neural signal processor.

In *IJCNN91-Seattle: International Joint Conference on Neural Networks*, volume 1, pages 443-446. IEEE Comput. Soc. Press, July 1991. ISBN: 0-7883-0164-1.

Mindshape Jan N.H. Heemskerk, Jacob M.J. Murre Arend Melissant, Mirko Pelgrom, and Patrick T.W. Hudson. Mindshape: a neurocomputer concept based on a fractal architecture. In *International Conference on Artificial Neural Networks*. Elsevier Science, 1992.

mod 2 Michael L. Mumford, David K. Andes, and Lynn R. Kern. The mod 2 neurocomputer system design. In *IEEE Transactions on Neural Networks*, 3(3):423-433, 1992.

NERV

R. Hauser, H. Horner, R. Maenner, and M. Makhaniok.

Architectural Considerations for NERV - a General Purpose Neural Network Simulation System.

In *Workshop on Parallel Processing: Logic, Organization and Technology – WOPLOT 89*, pages 183-195. Springer Verlag, Mars 1989. ISBN: 3-5405-5027-5.

NP – Neural Processor

D.A. Orrey, D.J. Myers, and J.M. Vincent.

A high performance digital processor for implementing large artificial neural networks.

In *Proceedings of the IEEE 1991 Custom Integrated Circuits Conference*, pages 16.3/1-4. IEEE Comput. Soc. Press, May 1991. ISBN: 0-7883-0015-7.

RAP – Ring Array Processor

N. Morgan, J. Beck, P. Kohn, J. Bilmes, E. Allman, and J. Beer.

The ring array processor: A multiprocessing peripheral for connectionist applications.

In *Journal of Parallel and Distributed Computing*, pages 248-259, April 1992.

RENNS – REconfigurable Neural Networks Server

O. Landsverk, J. Greipsland, J.A. Mathisen, J.G. Solheim, and L. Utne.

RENNS - a Reconfigurable Computer System for Simulating Artificial Neural Network Algorithms.

In *Parallel and Distributed Computing Systems, Proceedings of the ISMM 5th International Conference*, pages 251-256. The International Society for Mini and Microcomputers - ISMM, October 1992. ISBN: 1-8808-4302-1.

SMART – Sparse Matrix Adaptive and Recursive Transforms

P.Bessiere, A.Chams, A.Guerin, J.Herault, C.Jutten, and J.C.Lawson.

From Hardware to Software: Designing a “Neurostation”.

In *VLSI design of Neural Networks*, pages 311-335, June 1990.

SNAP – Scalable Neurocomputer Array Processor E.Wojciechowski.

SNAP: A parallel processor for implementing real time neural networks.

In *Proceedings of the IEEE 1991 National Aerospace and Electronics Conference; NAECON-91*, volume 2, pages 736-742. IEEE Comput.Soc.Press, May 1991.

Toroidal Neural Network Processor

S.Jones, K.Sammut, C.Nielsen, and J.Staunstrup.

Toroidal Neural Network: Architecture and Processor Granularity Issues.

In *VLSI design of Neural Networks*, pages 229-254, June 1990.

SMART and SuperNode P. Bessi'ere, A. Chams, and P. Chol. **MENTAL** : A virtual machine approach to artificial neural networks programming. In *NERVES*, ESPRIT B.R.A. project no 3049, 1991. (The report archived on neuroprose)

Standard Computers

EMMA-2

R.Battiti, L.M.Briano, R.Cecinati, A.M.Colla, and P.Guido.

An application oriented development environment for Neural Net models on multi-processor Emma-2.

In *Silicon Architectures for Neural Nets; Proceedings for the IFIP WG.10.5 Workshop*, pages 31-43. North Holland, November 1991. ISBN: 0-4448-9113-7.

iPSC/860 Hypercube

D.Jackson, and D.Hammerstrom

Distributing Back Propagation Networks Over the Intel iPSC/860 Hypercube

In *IJCNN91-Seattle: International Joint Conference on Neural Networks*, volume 1, pages 569-574. IEEE Comput. Soc. Press, July 1991. ISBN: 0-7083-0164-1.

SCAP – Systolic/Cellular Array Processor

Wei-Ling L., V.K.Prasanna, and K.W.Przytula.

Algorithmic Mapping of Neural Network Models onto Parallel SIMD Machines.

In *IEEE Transactions on Computers*, 40(12), pages 1390-1401, December 1991. ISSN: 0018-9340.

D.8 Databases for experimentation with NNs

- The nn-bench Benchmark collection accessible via anonymous FTP on
"pt.cs.cmu.edu"
in directory
"/afs/cs/project/connect/bench"
or via the Andrew file system in the directory
"/afs/cs.cmu.edu/project/connect/bench"
In case of problems email contact is "nn-bench-request@cs.cmu.edu". The data sets in this repository include the 'nettalk' data, the 'two spirals' problem, a vowel recognition task, and a few others.
- UCI machine learning database
accessible via anonymous FTP on
"ics.uci.edu" [128.195.1.1]
in directory
"/pub/machine-learning-databases"
- NIST special databases of the National Institute Of Standards And Technology:
NIST special database 2:
Structured Forms Reference Set (SFRS)

The NIST database of structured forms contains 5,590 full page images of simulated tax forms completed using machine print. THERE IS NO REAL TAX DATA IN THIS DATABASE. The structured forms used in this database are 12 different forms from the 1988, IRS 1040 Package X. These include Forms 1040, 2106, 2441, 4562, and 6251 together with Schedules A, B, C, D, E, F and SE. Eight of these forms contain two pages or form faces making a total of 20 form faces represented in the database. Each image is stored in bi-level black and white raster format. The images in this database appear to be real forms prepared by individuals but the images have been automatically derived and synthesized using a computer and contain no "real" tax data. The entry field values on the forms have been automatically generated by a computer in order to make the data available without the danger of distributing privileged tax information. In addition to the images the database includes 5,590 answer files, one for each image. Each answer file contains an ASCII representation of the data found in the entry fields on the corresponding image. Image format documentation and example software are also provided. The uncompressed database totals approximately 5.9 gigabytes of data.

- NIST special database 3:

Binary Images of Handwritten Segmented Characters (HWSC)

Contains 313,389 isolated character images segmented from the 2,100 full-page images distributed with "NIST Special Database 1". 223,125 digits, 44,951 upper-case, and 45,313 lower-case character images. Each character image has been centered in a separate 128 by 128 pixel region, error rate of the segmentation and assigned classification is less than 0.1%. The uncompressed database totals approximately 2.75 gigabytes of image data and includes image format documentation and example software.

NIST special database 4:

8-Bit Gray Scale Images of Fingerprint Image Groups (FIGS)

The NIST database of fingerprint images contains 2000 8-bit gray scale fingerprint image pairs. Each image is 512 by 512 pixels with 32 rows of white space at the bottom and classified using one of the five following classes: A=Arch, L=Left Loop, R=Right Loop, T=Tented Arch, W=Whirl. The database is evenly distributed over each of the five classifications with 400 fingerprint pairs from each class. The images are compressed using a modified JPEG lossless compression algorithm and require approximately 636 Megabytes of storage compressed and 1.1 Gigabytes uncompressed (1.6 : 1 compression ratio). The database also includes format documentation and example software.

More short overview:

Special Database 1 - NIST Binary Images of Printed Digits, Alphas, and Text

Special Database 2 - NIST Structured Forms Reference Set of Binary Images

Special Database 3 - NIST Binary Images of Handwritten Segmented Characters

Special Database 4 - NIST 8-bit Gray Scale Images of Fingerprint Image Groups

Special Database 6 - NIST Structured Forms Reference Set 2 of Binary Images

Special Database 7 - NIST Test Data 1: Binary Images of Handprinted Segmented Characters

Special Software 1 - NIST Scoring Package Release 1.0

Special Database 1 - \$895.00

Special Database 2 - \$250.00

Special Database 3 - \$895.00

Special Database 4 - \$250.00

Special Database 6 - \$250.00

Special Database 7 - \$1,000.00

Special Software 1 - \$1,150.00

The system requirements for all databases are a 5.25" CD-ROM drive with software to read ISO-9660 format.

Contact: Darrin L. Dimmick
dld@magi.ncsl.nist.gov (301)975-4147

If you wish to order the database, please contact:

Standard Reference Data
National Institute of Standards and Technology
221/A323
Gaithersburg, MD 20899
(301)975-2208 or (301)926-0416 (FAX)

– CEDAR CD-ROM 1: Database of Handwritten
Cities, States, ZIP Codes, Digits, and Alphabetic Characters

The Center Of Excellence for Document Analysis and Recognition (CEDAR) State University of New York at Buffalo announces the availability of CEDAR CDROM 1: USPS Office of Advanced Technology The database contains handwritten words and ZIP Codes in high resolution grayscale (300 ppi 8-bit) as well as binary handwritten digits and alphabetic characters (300 ppi 1-bit). This database is intended to encourage research in off-line handwriting recognition by providing access to handwriting samples digitized from envelopes in a working post office. Specifications of the database include:

- + 300 ppi 8-bit grayscale handwritten words (cities, states, ZIP Codes)
 - o 5632 city words
 - o 4938 state words
 - o 9454 ZIP Codes
- + 300 ppi binary handwritten characters and digits:
 - o 27,837 mixed alphas and numerics segmented from address blocks
 - o 21,179 digits segmented from ZIP Codes
- + every image supplied with a manually determined truth value
- + extracted from live mail in a working U.S. Post Office
- + word images in the test set supplied with dictionaries of postal words that simulate partial recognition of the corresponding ZIP Code.
- + digit images included in test set that simulate automatic ZIP Code segmentation. Results on these data can be projected to overall ZIP Code recognition performance.
- + image format documentation and software included

System requirements are a 5.25" CD-ROM drive with software to read ISO-9660 format. For any further information, including how to order the database, please contact: Jonathan J. Hull, Associate Director, CEDAR, 226 Bell Hall

State University of New York at Buffalo, Buffalo, NY 14260 hull@cs.buffalo.edu
(email)

- AI-CD-ROM (see above under "other sources of information about NNs")

Bibliography

- [1] Denby, B. *Comp. Phys. Comm.* 49 (1988) 429.
- [2] Soucek B. *Neural and Concurrent Real-Time Systems*, (John Wiley) 1989,
- [3] Stimpfl G., Garrido Ll., *Comp. Phys. Comm.* (1991) 183.
- [4] Rumelhart D.E. et al., *Nature* 323 (1986) 533.
- [5] Hecht-Nielsen R., *Neurocomputing*, (Addison-Wesley) 1990.
- [6] McCulloch W.S., Pitts W., *Bulletin of Math. Bio.* 5 (1943) 115-133.
- [7] Hebb D. *The Organization of Behavior*, (Wiley) 1949.
- [8] Rosenblatt F., *Psychol. Rev.* 65 (1958) 386-408.
- [9] Widrow G., Hoff M.E., *Adaptative switching circuits*, (I.R.E.) 1960.
- [10] Minsky M., Papert S., *Perceptrons* (MIT Press) 1969.
- [11] Grossberg S., *Jou. Math. Psychol.* 6 (1969) 209-239.
- [12] Kohonen T. *IEEE Trans. Comp.* C-23 (1974) 444.
- [13] Hopfield J.J. *Proc. Nac. Acad. Sci. USA* 76 , (1982) 2554.
- [14] Werbos P. *PhD Thesis, Harvard* (1974)
- [15] Gora M., Tesi A., *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14, 1991, 76.
- [16] Garrido et al., 1992, (in preparation).
- [17] Johansson E.M. et al., (1991) Submitted to *IEEE Transactions on Neural Networks*.
- [18] Gawthrop P.J. et al. *Complex Systems* 4 (1990) 51.

- [19] Eadie W.T. et al, "Statistical Methods in experimental physics". North Holland Publishing Company (1971).
- [20] Denker J. et al. ,Complex Systems 1 (1987) 877-922.
- [21] Duda R., Hart P. Pattern Classification and Scene Analysis John Wiley (1973).
- [22] Jadach S. et al. "The tau polarization measurement" in *Z Physics at LEP 1* CERN 89-08 (1989) .
- [23] Orteu S., PhD thesis, Universitat Autònoma de Barcelona,(1989).
- [24] Fasano G., Franceschini A. Monthly Notices of the Royal Astronomical Society 1987 255.
- [25] Garrido Ll. and Gaitan V. (1991) International Journal of Neural Systems Vol 2. No. 3 .
- [26] The Aleph Collaboration, Nucl. Inst. Meth. A 294 (1990) 121.
- [27] Aarnio P. et al. (DELPHI coll.) Nucl. Instr. Meth.1991, A303.
- [28] Adeva B. et al. (L3 coll.) Nucl. Instr. Meth.1990, A289.
- [29] Ahmet K. et al. (OPAL coll.) Nucl. Instr. Meth.1991 A305.
- [30] Particle Data Group. Review of Particle Properties. Phy.Lett. 1990,B239.
- [31] Saund E., IEEE Transactions on Pattern Analysis and Machine Intelligence, 11, 1989,304.
- [32] Kendall M.G. "A course in Multivariate Analysis", Griffin & Co., (1957) London.
- [33] Sanger T.D., Neural Networks, 2, 1989,459 .
- [34] Oja E., Artificial Neural Networks. In: Kohonen T., Mäkisara K., Simula O., Kangas J. (eds.) Proceedings of the 1991 International Conference on Artificial Neural Networks. North-Holland, Amsterdam, p. 737
- [35] Garrido Ll. et al, in preparation (1993)
- [36] Mezard M., Parisi G., Virasoro M.A., Spin Glass Theory and Beyond (World Scientific, Singapur) 1987.
- [37] Hopfield J.J., Tank D.W., Biol. Cybernetics 52 (1985) 141.
- [38] Fu Y. and Anderson P.W., Jour. Phys. A19 (1986), 1605-1620.
- [39] Decamp D. et al., Zeit. fur Physik C53 (1992) 1.

- [40] Decamp D. et al, ALEPH Collab., Phys. Lett. B265 (1991), 430.
- [41] Alemany R., Tesina, Universitat Autònoma de Barcelona (1990).
- [42] Decamp D. et al., ALEPH Collab., Phys. Rep. 216 (1992) 253.
- [43] Numerical Recipes in Fortran
- [44] Bock R.K. et al, Data analysis techniques for high energy physics experiments. (Cambridge University Press, Cambridge) 1990.
- [45] Dahl-Jensen E., CERN/R807/8 internal note.
- [46] Shaw G.L., Palm G., Brain Theory (World Scientific, Singapore) 1988.
- [47] Humpert B., Comp. Phys. Comm. 58 (1990) 223,
Treleaven P., Vellasco M., Comp. Phys. Comm. 57 (1989) 543.
- [48] Denby B., Linn S.L., Comp. Phys. Comm. 56 (1990) 293.
- [49] Peterson C., Nucl. Instr. Meth. A 279 (1989) 537.
- [50] Peterson C., Anderson J.R., Complex Systems 1 (1987) 995.
- [51] Mermikides M.E., internal note ALEPH-TPCDET 84-69 (1984).
- [52] Carrabina J., PhD. Universitat Autònoma de Barcelona.
- [53] Pérez Vicente C.J., Carrabina J., Valderrama E., (1992). Submitted to Network: Computation in Neural Systems.
- [54] Carrabina J. et al IWANN93 Proc. 1993.
- [55] Kolmogorov A.N. Dokl. Akad. Nauk USSR 114 (1957) 953-956.
- [56] Richard M.D. and Lippmann R.P. Neural computation 3, 1991, 461-483.
- [57] Klir G.J. and Folger T.A. "Fuzzy Sets, Uncertainty, and Information". Prentice-Hall International Editions, 1988.