FERMILAB-Conf-90/94

Neural Network Tutorial for High Energy Physicists*

Bruce Denby
Fermi National Accelerator Laboratory
P.O. Box 500
Batavia, Illinois 60510

May 1990

* Presented at the International Workshop of Software Engineering, Artificial Intelligence and Expert Systems for High Energy and Nuclear Physics, Lyon, France, March 19-24, 1990.

Neural Network Tutorial for High Energy Physicists*

Bruce Denby

Fermi National Accelerator Laboratory
Batavia, IL 60510
U.S.A.

Abstract
Neural Networks are introduced through analogies to data analysis and electronic techniques used in high energy physics.

I. Introduction

Neural networks are data processing architectures constructed from large numbers of highly interconnected simple processors called neurons. The neurons in fact are not really processors at all but are rather more akin to single gates. These networks resemble qualitatively structures found in brains, which are known to consist of large numbers of simple neuron interconnected by means of dendrites, axons, and synapses. Brain structure is still poorly understood, but there exist today some simple models of neural networks which already share some of the characteristics of brains, in particular, fast pattern recognition in the presence of noise, and what might be called "reasoning", i.e., the finding of optimal solutions to a problem when constraints are present.

The field of applications of neural networks to high energy physics is still new, but this marriage of disparate domains seems eminently reasonable both from the viewpoint of high energy physicists and from that of researchers in neural networks. High energy physics needs very fast pattern recognition for triggering applications and to speed up offline processing. These needs today are being driven by such new endeavors as SSC, LHC, and RHIC, in which event rates and event complexity will reach unprecedented levels. The high energy physics applications may prove attractive to researchers in neural networks because 1) the problems to solve are relatively clear cut,2) high energy physics really needs a 'real time' application of neural nets, while simulations maybe sufficient for applications in some other fields, 3) high energy physics is in the forefront of scientific research, and, finally, 4) the $8 billion price tag, e.g., of the SSC may give the hopefully true impression that funds will be available for research into these applications.

II. Analogy to Fast Electronics

* Talk presented by Bruce Denby at International Workshop of Software Engineering, Artificial Intelligence, and Expert Systems for High Energy and Nuclear Physics, March 19-24, 1990, Lyon, France.

Consider a beam line scintillator telescope with 4 scintillators, as shown very schematically in figure 1. Signals from the scintillators pass via wires (we will ignore timing here) to a junction where they are summed. The summed signals then pass into a saturating amplifier (i.e., a discriminator) which gives a high output if all scintillators are hit and a low output if they are not.

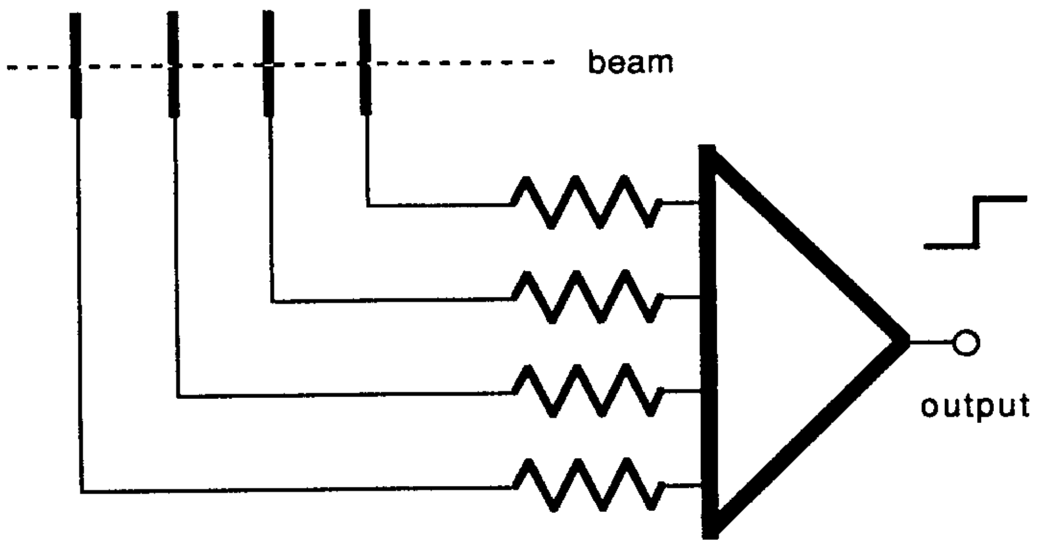


Figure 1. Scintillator Beam Telescope

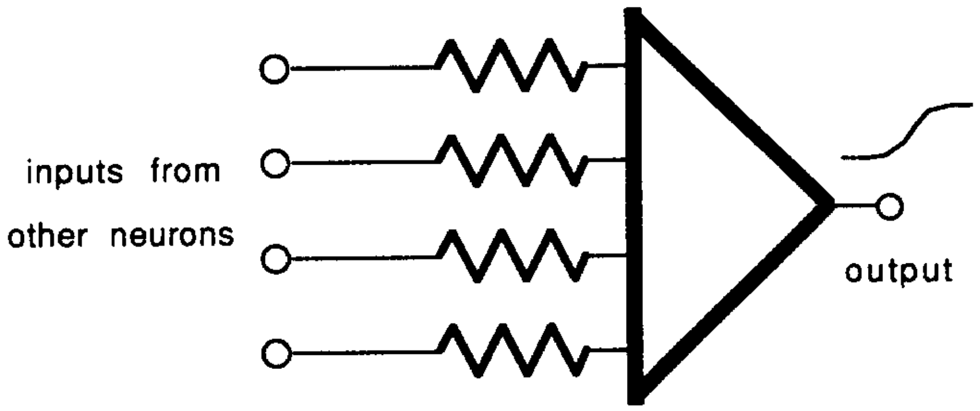


Figure 2. Hopfield model of a neuron

In figure 2 we show the Hopfield model of a neuron 1), the basic building block of artificial neural networks. Notice that it looks exactly like the discriminator of figure 1. This is because the neuron performs a similar function. Signals from other neurons are summed at the input of the neuron, which then responds to this sum. The response function of neurons in artificial neural networks is called a 'sigmoid' function (see figure), which is quite similar to the output function of a discriminator. It turns out that the more rounded, non-linear shape of the sigmoid is important, as we shall see later.

Suppose we expand our beam line telescope to respond to tracks of a variety of orientations.We could construct a hodoscope array and link together into discriminators the outputs of counters which will be hit by a particular track, as shown in figure 3 The device thus constructed, our first 'neural network', produces a 'high' output for each track present in a given event.

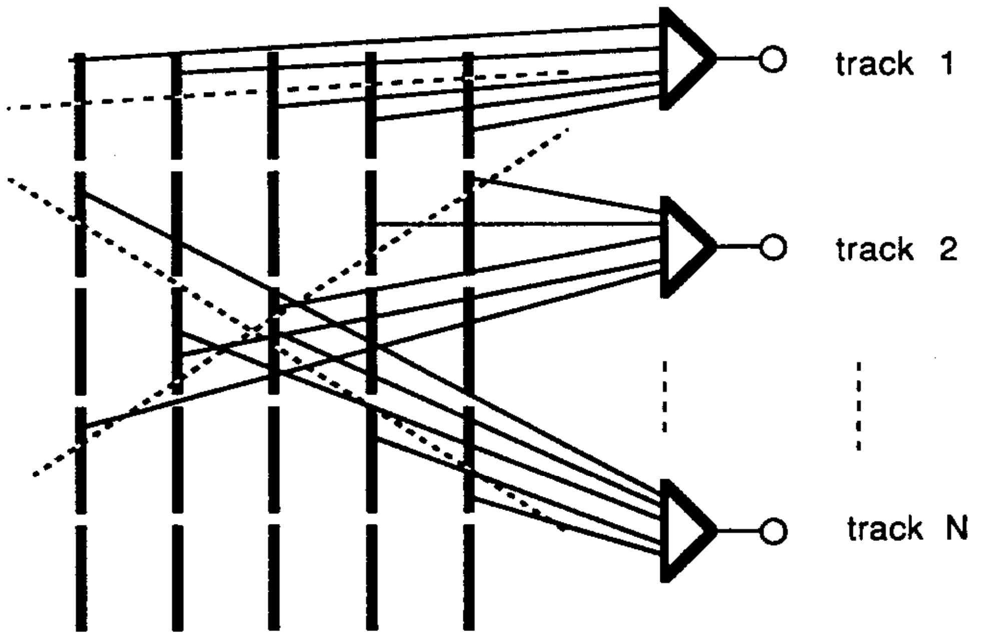
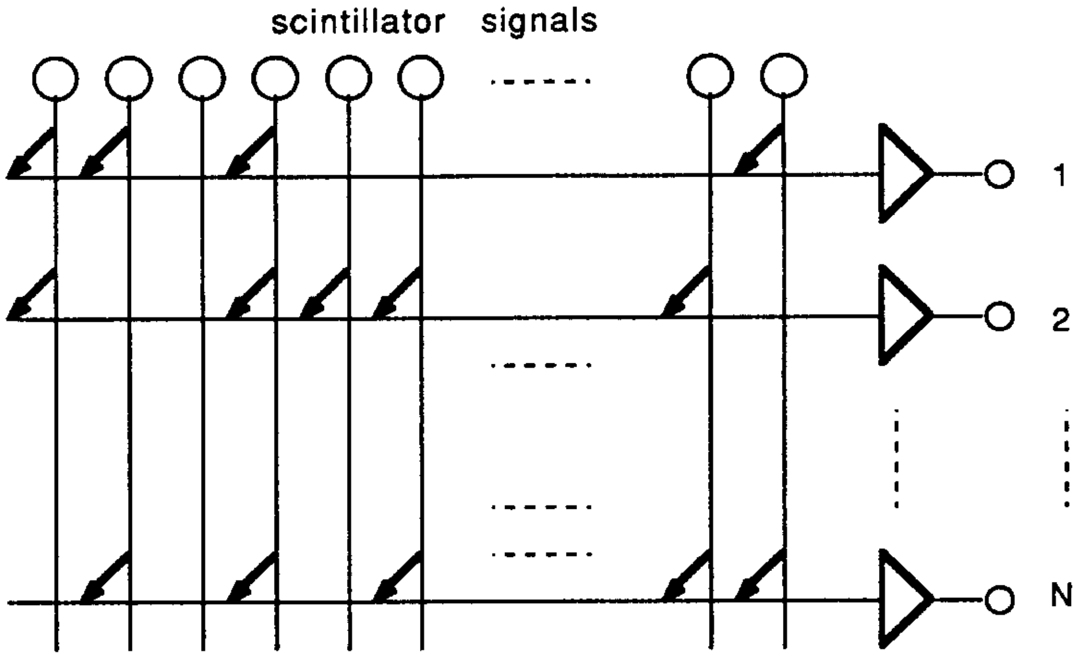


Figure 3. Hodoscope associative memory

It is worth noting here another analogy to biology. In the visual system there exist cells which respond to straight lines in the visual field $^{2)}$. These neurons are found to simply sum the outputs of light sensitive cells which happen to physically lie upon lines on the retina. These 'orientation sensitive' units thus form a sort of coincidence of the light sensitive cells.

Our network can also be thought of as a 'content addressable memory'. The hit patterns corresponding to tracks are 'stored' in the memory through the interconnection pattern to the discriminators. When a pattern is imposed on the memory, it gives a 'high' output if that pattern corresponds to one that has been thus stored. Content addressable memories, which are more usually referred to as 'associative memories' are usually depicted in a slightly different way, as shown in figure 4. Here, the outputs of the scintillators are arranged vertically, and one summing line is laid horizontally across them for each pattern to be stored. The pattern is encoded by connecting the scintillators contributing to a given pattern to its corresponding summing line via a resistor (the arrows in the figure). This type of circuit, also called a 'pattern matching' circuit, was in fact exactly the type of circuit one referred to as a 'neural network' in the early days of neural network research in the 1950's. A CMOS implementation of such a circuit, referred to as a 'neural network associative memory', has been recently constructed at ATT Bell labs [3]. Although one would probably not want to build a large tracking system from scintillators using this technique, a similar thing has been done using silicon microstrips as inputs rather than scintillators. These are the associative memory track finding chips being developed at INFN Pisa [4].



Another view of scintillator associative memory

Figure 4.

II. Feed Forward Networks

In general, the resistors we use in our summing circuitry could be different, for example to correct for miscalibrations or to weight certain counters more heavily than others in the sum. Let's explicitly exploit this option in an application to calorimetry. Consider an idealized, one dimensional calorimeter with 5 cells. A monoenergetic beam containing electrons and hadrons will impinge near, but not necessarily exactly at, the center of the array of cells. In the upper corner of figure 5 we show how the energy deposits in the calorimeter might look for a typical electron and hadron. The electron's energy will be fully contained within the five cells by virtue of the intrinsic narrowness of electron showers, but for the hadron, significant energy will typically leak out. Thus the energy deposited in the calorimeter can be used as to identify the type of particle. All we need do is sum the cell energies into a discriminator and set the threshold correctly, that is to say, use the cells as inputs to a neuron. We may of course at the same time perform an arbitrary number of other linear combinations of the cell energies. For example, we might want to compute the barycenter of the distribution in order to know fairly precisely the position of impact of the particle. This we can easily do by connecting the cells to another neuron via resistors weighted according to the position of the center of each cell. Our network is now as shown in figure 5, but here we have adopted yet another formalism to draw the net.. Neurons are represented by circles, and weights simply by lines connecting the neurons. The lines are understood to contain weights which multiply the signals passing through them. This is the formalism used in most of the literature on neural networks. Note that if for some reason we did not know what values to assign to the weights, if the counters were uncalibrated, for instance, we could determine them by taking some test data of pure electrons and pure hadrons and then using some kind of iterative fitting technique to calculate the values of the weights.

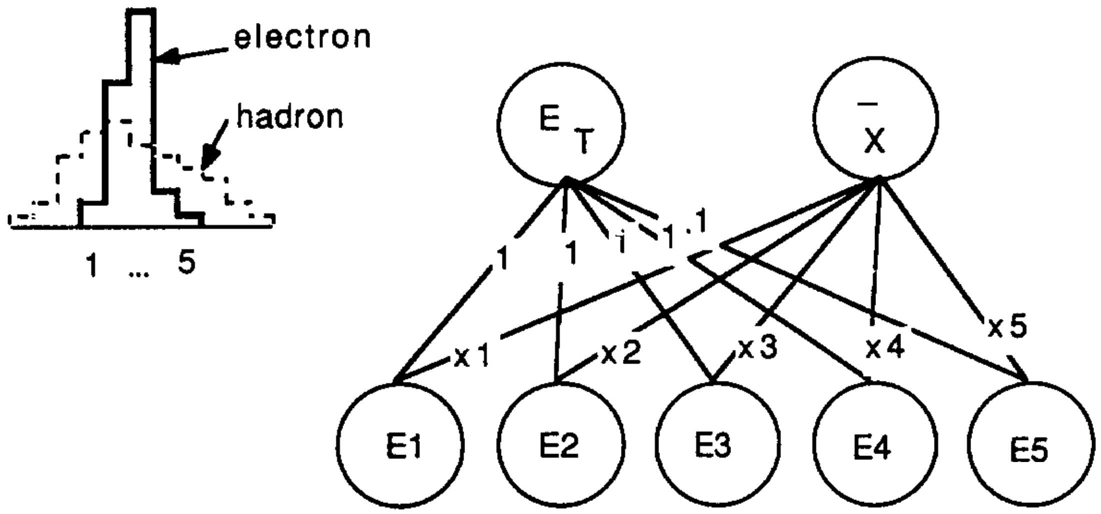


Figure 5. Calorimeter Network

This network has a two layer structure, i.e., a layer of input signals and an output layer containing the calculated quantities. Thus far we have just considered linear combinations of the inputs. In fact, this kind of weighted summing of calorimeter energies is a fairly standard technique for making fast calorimeter triggers. Why did we need to introduce neural networks? There are at least two reasons. The first is that today very many researchers in a wide range of fields are anxious to have neural net hardware which will permit them to program in large numbers of different combinations of their input data in parallel. This demand will hopefully drive chip manufacturers to produce large scale massively interconnected, programmable hardware. Even if, in high energy physics, one only retained exactly the type of trigger circuit one is already accustomed to , these new circuits should allow a much more compact and flexible implementation with several different types of trigger sums performed on the same chip. But there is also another very important reason to introduce neural networks. Linear summing networks are, of course, only sensitive to linear correlations in the input data. If there were non-linear functions of the input data that were of interest in triggering, how would we be able to access them?

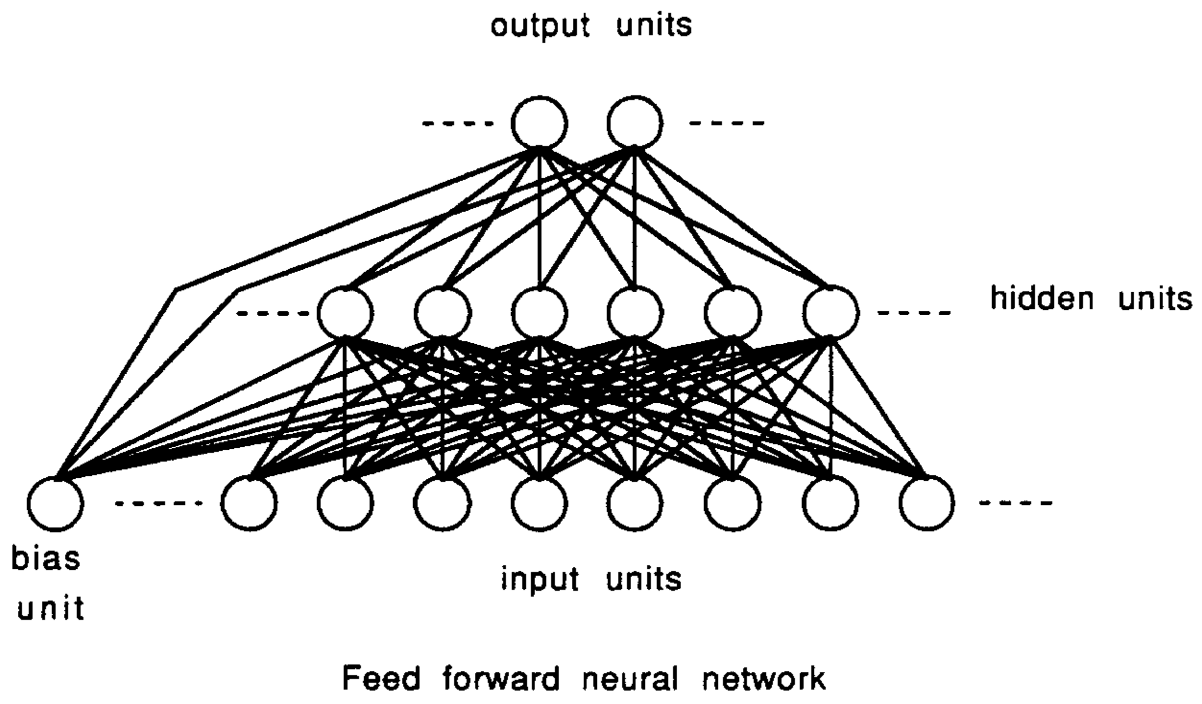


Figure 6.

III. Nonlinearity

It turns out that by making more complicated, multilayer configurations of neurons, we can make use of the nonlinear sigmoid functions of the neurons to handle nonlinear correlations in input data as well. The standard multilayer configuration is shown in figure 6. It consists of an input layer, an output layer, and an intermediate layer called the 'hidden' layer. Only the hidden and output layers are sigmoid neurons; the input units are linear. There is also a so called 'bias' unit at the input which has a constant value of 1 and which connects via weights to all hidden and output units. It is of course well known that any function can be represented as a sum of sinusoids, i.e., as a Fourier series. It is not unreasonable to suspect that a large variety of functions could also be represented as a sum of sigmoids. It has in fact been shown that any well behaved function mapping N real variables onto M real variables can be represented to any desired degree of accuracy by a neural network with N input units, M output units, and a single hidden layer [5]. The network does this by summing together many sigmoids which have been shifted, via the bias neuron and its weight, and scaled, by the weights connected to their outputs.

What kind of nonlinear function would we be interested in for high energy physics? The truth of the matter is that the majority of useful functions are probably nonlinear, but we give here an example from electron identification. A popular method of discriminating electrons from hadrons in real calorimeters is the covariance matrix technique. In one implementation of this technique [6], a χ^2 is defined for electrons and hadrons. These are given by:

$$\chi_e{}^2 = (P-P_e)H_e(P-P_e)$$

$$\chi_h{}^2 = (P-P_h)H_h(P-P_h)$$

Where P is the vector of pulse heights in the calorimeter, **H** is the covariance matrix of the calorimeter over test samples of electrons and hadrons, and P_h and P_e are the vectors of mean pulse heights for electrons and hadrons. Note that these χ^2 are quadratic in the calorimeter energies.

One strategy then would be to try to construct a neural network to implement these functions. It would have an input layer consisting of the calorimeter energies, a hidden layer of sigmoid units, and a two unit output layer representing our two χ^2. Our only task will be to determine the necessary weight matrices to install in order to make the network implement the desired functions. As long as we are going to go to all the trouble to do this, why not also take advantage of the following fact. The network does not care what function we tell it to implement. Instead of arbitrarily choosing to implement this particular covariance matrix technique, why not try to build a network to implement the best possible electron/hadron discrimination technique based on performance on a test data set. We don't care what the closed form of the function implemented is, we only care that the network works well.

IV. Training

How can we find the weight matrices we need to do this? There is a standard procedure, called 'backpropagation' [7] which is used to 'train' neural networks to perform desired tasks. There are other types of training algorithms but this is by the far the most common and is fairly easy to implement. What is done in backpropagation is to define an energy function which is the sum over the network and over a training sample of the deviation of the output values from their desired values. Gradient descent is then performed on this function with respect to the weights in order to minimise the deviation of the network response from the desired response. For pattern p,

$$E = \Sigma_p E(p) = \Sigma_{ip} [d(p,i)-t(p,i)]^2$$

$$E'(p,i,j) = [d(p,j)-t(p,j)]s'(j)t(p,i) \quad \text{(for output units)}$$

$$E'(p,i,j) = \Sigma_k[d(p,j)-t(p,j)]s'(k)w(k,j)s'(j)t(p,i) \quad \text{(for hidden units)}$$

where i,j are the indices of neurons, p is the index of a pattern, w(i,j) is the weight between neurons i and j, E'(p,i,j) is the derivative of the energy function (due to pattern p) with respect to w(i,j), d(p,j) is the desired output of neuron j in pattern p, t(p,j) is its true output, and s'(i) is the derivative of the sigmoid function of neuron i. The prescription of backpropagation then is that

$$\Delta w(i,j) = -\varepsilon E'(p,i,j) + \alpha(\text{last } \Delta w(i,j))$$

where $\Delta w(i,j)$ is the change in w(i,j) for this iteration, ε is the distance to move along the gradient, also called the 'learning coefficient', and the term containing α, the 'momentum' coefficient, is a smoothing term. In practice, the weights are updated after only one or a few presentations of training patterns, rather than after the whole set. This is not true gradient descent but is easier to implement and seems to work well. There are a number of commercial backpropagation simulators on the market, but they are also rather easy to code in house. A general purpose Fortran simulator has been written at Fermilab for high energy physics applications.

V. Electron Identification

What we are proposing to do then, is to construct a network with calorimeter energies as inputs, a layer of hidden units, and two output units, and, using a set of known electrons and hadrons, train the network to distinguish the two classes of events. We will run back propagation using the training sample to determine the weights we need so that when electron energy deposits are presented, the electron output bit goes high, and when hadronic deposits are presented , the hadronic bit goes high. A test of this method was made and published by Cutts et al. [8]. They considered a section of calorimeter with 5 depth segments and summed the energy in the 4 concentric rings of cells surrounding the entry point of the incident particle, which could be either an electron or a hadron. The input layer of their network thus had 20 units. Eight hidden units were used and there were two output units to specify electron or hadron. Some typical input patterns are shown in figure 7. The training data was generated using ISAJET to produce samples of electrons from decays of Z particles, and hadrons from two jet events. After training,the performance was tested on an independent set of events. A histogram of the output bits for this test sample is shown in figure 8,

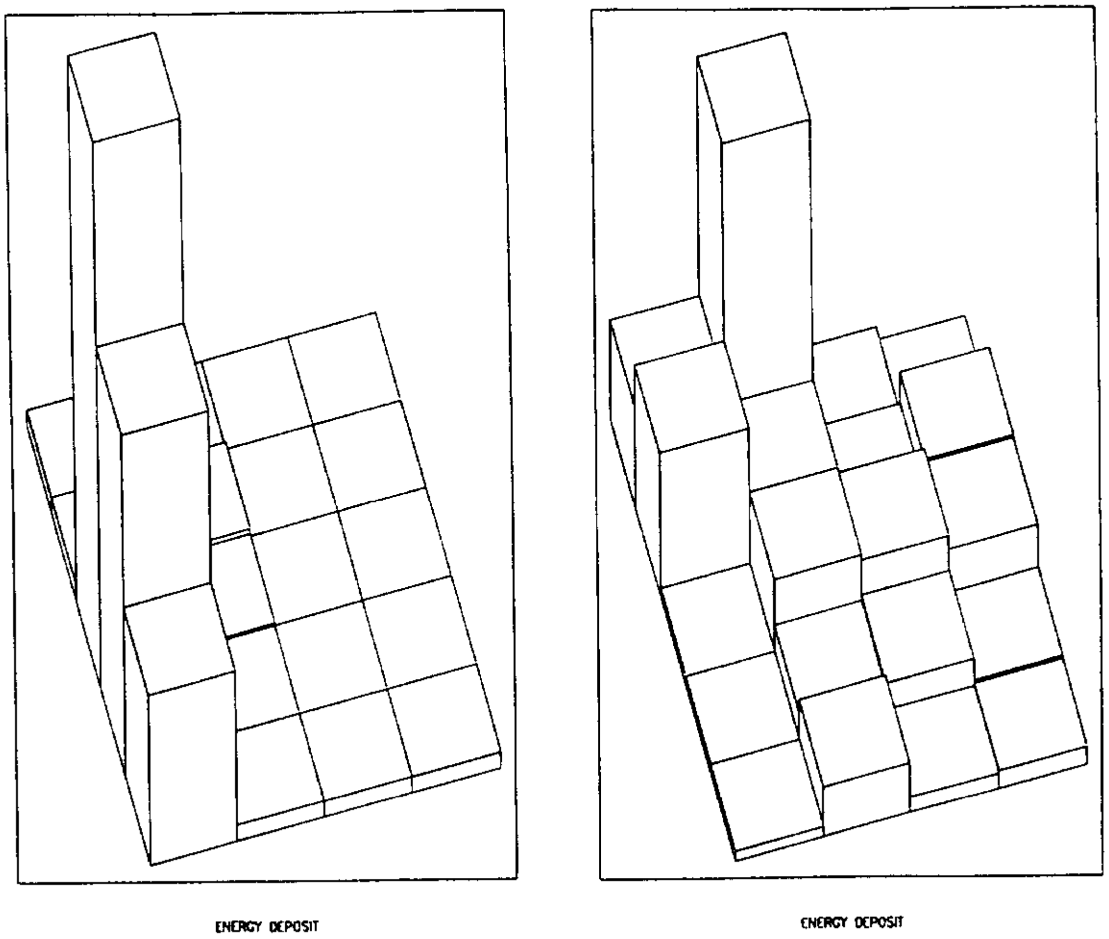


Figure 7. Electrons (left) and hadrons. Left axis is depth, bottom axis is distance from impact.

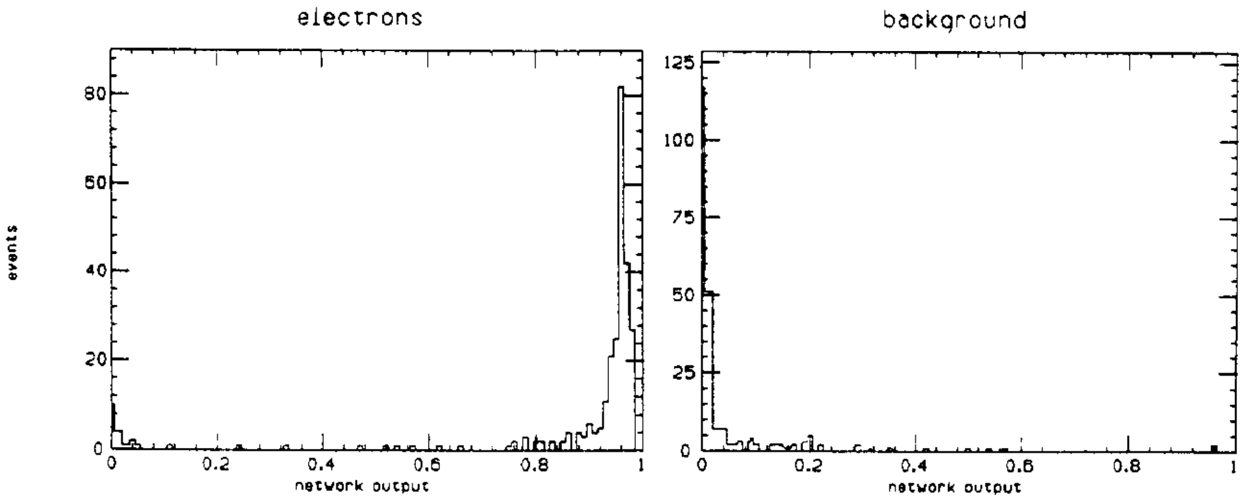


Figure 8. Network response to electrons (left) and hadrons.

showing the clear separation of the classes. The network was able to correctly identify 90 percent of the electrons and only called .9 percent of the hadrons electrons, which is comparable to the results they found using a standard electron identification algorithm. The neural net algorithm, however, unlike a standard algorithm, may be implemented as an analog circuit or some other fast hardware implementation.

VI. Recurrent Networks

Up to now we have discussed feed forward networks. Signals enter through the input layer, pass through the hidden layer, and exit through the output layer. There is also another standard neural network architecture called the 'recurrent network' in which the outputs can be fed back into the inputs, so that signals can cycle repeatedly through the network. The basic architecture is shown in figure 9.

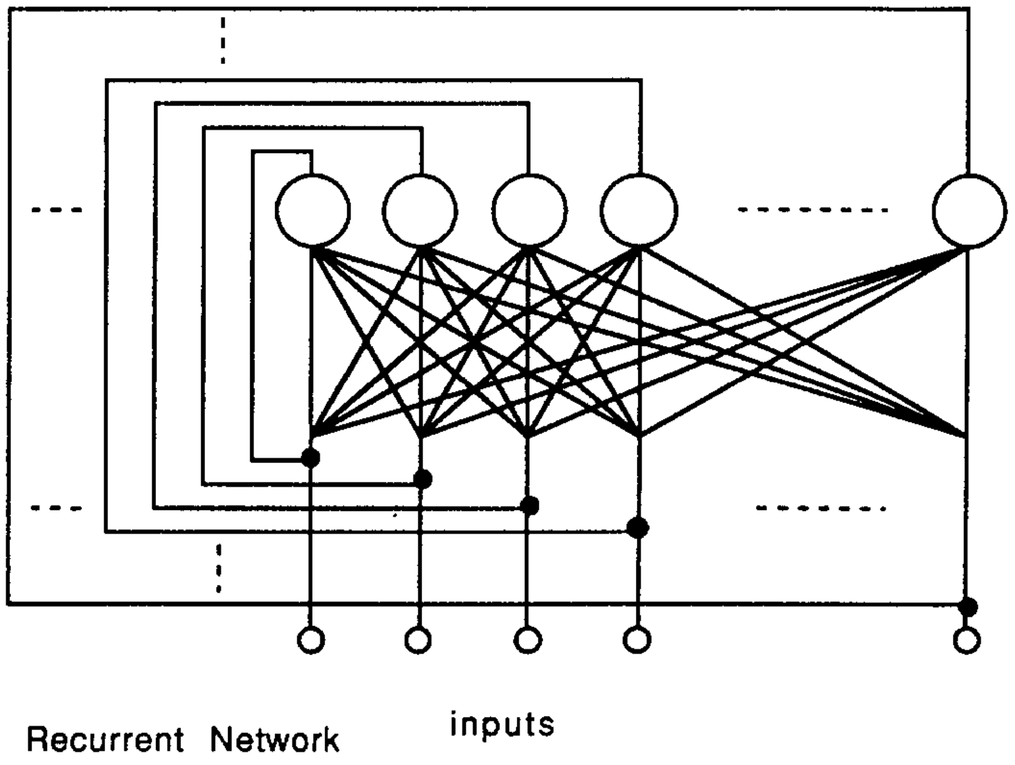


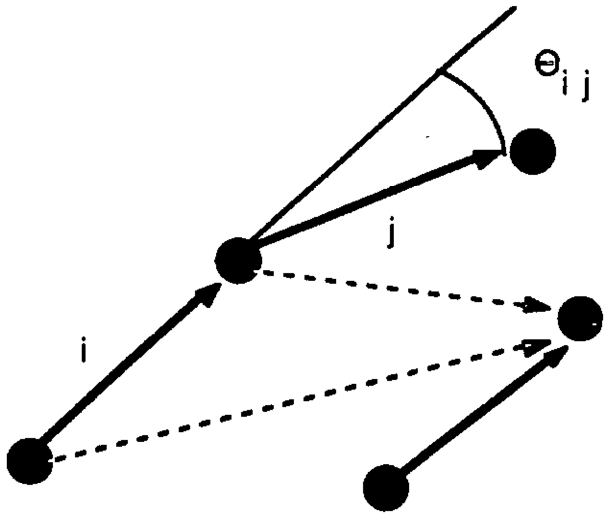
Figure 9.

In recurrent networks the neurons are not divided into input, hidden, and output. Rather, the data are used to specify an initial set of levels at the neural inputs (which are in fact tied to the outputs), and the network evolves to some final set of levels. It turns out that if the weights are chosen to be symmetric, i.e., w(i,j)=w(j,i), then the system will always evolve to a stable final state, and that, furthermore, in so doing, an energy function,

$$E = -1/2 \Sigma_{ij} w(i,j)t(i)t(j)$$

will attain its minimum value [9]. Although it is possible to do back propagation to determine the weights needed for recurrent networks, what is more commonly done is to make use of the existence of the energy function in order to guess a form for the weights. That is, one tries to post the problem to be solved as a minimization problem and then deduces from the energy function the form the coefficients must have.

Recurrent networks have been used in high energy physics in applications to tracking [10], photon combinatorics [11], and secondary vertex finding [12]. In the tracking application, the neurons were identified with directed links connecting together two hits in a drift chamber as in figure 10.



Neuron links

Figure 10.

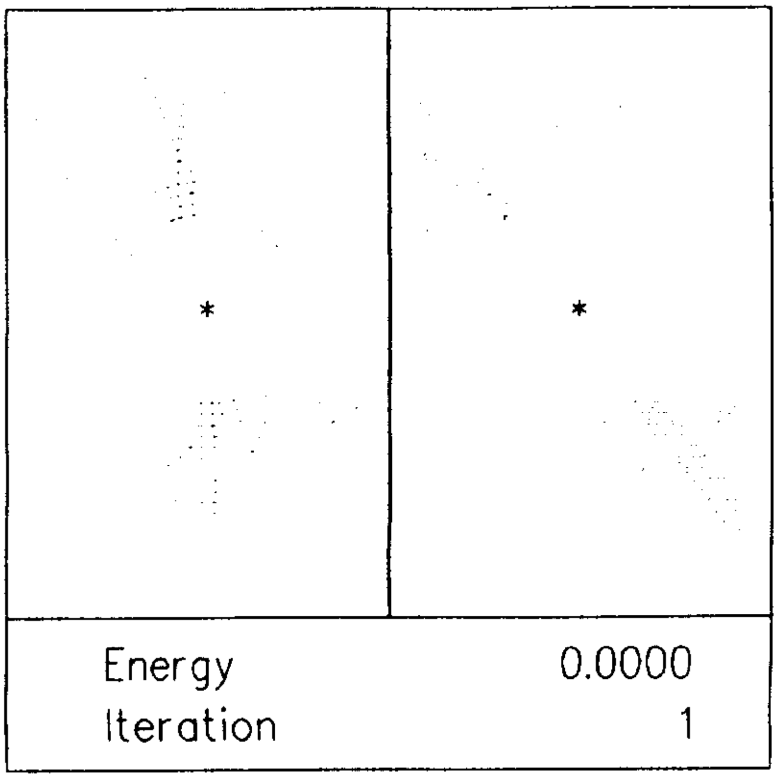
The weights were defined as

$$w(i,j) = \cos^n\theta_{ij} / l_i l_j$$

for neurons sharing a hit and in a head-tail configuration, where θ_{ij} is the angle between the neurons, and l_i and l_j are the lengths of the neurons. For neurons in head-head or tail-tail configurations a negative coefficient is applied,

$$w(i,j) = -\text{const.}$$

It is only necessary to connect neurons to others in a small neighborhood, and very long neurons need not be considered since they will never exist in a real track. Thus neurons which have similar directions will reinforce each other, as long as they are in a head-tail configuration, and other neurons sharing hits will inhibit each other. The network will minimize the sum of the $\cos^n$ terms



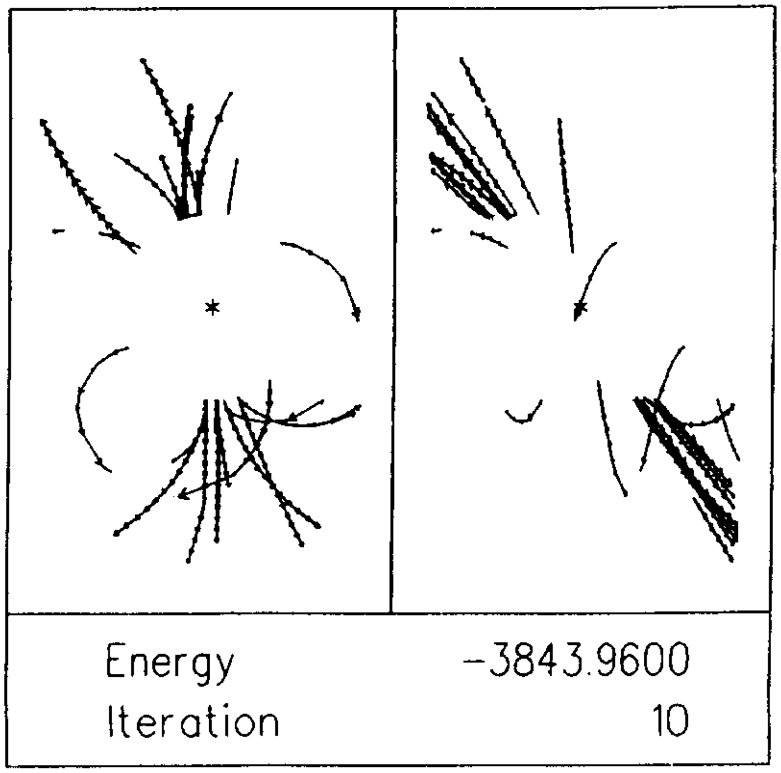


Figure 11. Hits in TPC from e^+e^- event (top) and reconstructed event (below).

between neighbouring neurons so that neurons which point along tracks will be reinforced and other neurons will be inhibited. Figure 11 shows a simulated electron positron event reconstructed with this method.

The tracking algorithm has a couple of drawbacks. The first is that it requires a very large number of neurons in order to accommodate the very high resolution of drift chambers and the like. The second is that it is necessary to calculate new w(i,j) for each event. This second point means that even if one did have a neural net with enough neurons to do the tracking, it would be necessary to recalculate the coefficients and reload them on each event, so that the speed advantage of the network would be lost. There may be ways around these difficulties and work is continuing here. In any case, the neural tracking algorithm is highly parallel and hence vectorizable, and the calculation of the w(i,j) is vectorizable as well, so that this may present a good vectorizable tracking algorithm even if for the moment we can not implement it in hardware [13]. A similar algorithm has been proposed for determining the trajectories of airplanes. Here the resolution required in not so high [14]. In general recurrent networks are used for optimization problems where there is a cost function to be minimized in the presence of constraints. Recurrent nets have been used for the Travelling Salesman Problem [15], for load balancing on parallel computers [16], for scheduling classes [17], for graph partitioning, [18] and the like, often with very good results.

VI. Conclusion

Simple feed forward neural networks are circuits that look not much different from fast electronic circuits that high energy physicists are familiar with. Far more powerful circuits will be possible using neural networks by exploiting the nonlinearity in the neuron transfer function. It should be possible to make complicated trigger decisions very quickly. Recurrent networks are appropriate to problems requiring optimization in the presence of constraints. Hardware feed forward networks are just beginning to appear on the market. As yet no commercial recurrent networks have appeared, but they should not be far behind..

REFERENCES

[1] J. J. Hopfield and D. W. Tank, *Biological Cybernetics* **52** (1985) p. 141.

[2] D. H. Hubel and T. N. Wiesel, *J. Physiol.* **160** (1962) pp. 106-154.

[3] L. D. Jackel, H. P. Graf, R. E. Howard, *Applied Optics* (1987) 5077.

[4] M. Dell'Orso and L. Ristori, VLSI Structures for Track Finding, proceedings of Int. Conference on the Impact of Digital Microelectronics and Microprocessors on Particle Physics, Trieste, Italy, 28-30 March, 1988, World Scientific Pub. Co.

[5] R. Hecht-Nielsen, Theory of Backpropagation Neural Networks, proc. of the International Joint Conference on Neural Networks, vol. I, pp. 593-605, Washington, D.C.,18-22 June, 1989, IEEE Catalog no. 89CCH2765-6.

[6] M. G. Albrow et al., *Nucl. Inst. Meth.* **A256** (1987) 23-37.

[7] D. Rumelhart et al., Parallel Distributed Processing, Explorations in the Microstructure of Cognition, vol. I, ch. 8, MIT Press, Cambridge, Mass.

[8] D. Cutts et al., The Use of Neural Networks in the D0 Data Acquisition System, presented at the conference, Real Time '89, Williamsburg, VA (May 1989), proceedings to be published.

[9] ibid. ref. 1

[10] B. Denby, *Comp. Phys. Comm.* **49** (1988) 429-448.
C. Peterson, *Nucl. Inst. Meth.* **A279** (1989) 537.

[11] T. Awes, *Nucl. Inst. Meth.* **A276** (1989) 468-481.

[12] B. Denby et al., Neural Networks for Triggering, FERMILAB-Conf-90/20.

[13] B. Denby and S. Linn, *Comp. Phys. Comm.* **56** (1990) 293-297.

[14] M. Oyster, Hughes Aircraft Corporation, in The DARPA Neural Network Study, AFCEA International Press, Fairfax, VA (1988) p. 451.

[15] J. J. Hopfield and D. W. Tank, *Science* **233** (1986) 625.

[16] G. C. Fox and W. Furmanski, Load Balancing by a Neural Network, Caltech C^3P363, September, 1986.

[17] C. Peterson, private communication.

[18] ibid. ref. 10 (Peterson).