



**Fermi National Accelerator Laboratory**

**FERMILAB-Conf-92/121-E**

# **Tutorial on Neural Network Applications in High Energy Physics: A 1992 Perspective**

B. Denby

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

April 1992

Invited tutorial to be published in the proceedings of the *Second International Workshop on Software Engineering, Artificial Intelligence and Expert Systems for High Energy and Nuclear Physics*, La Londe les Maures, France, January 1992.



## **Disclaimer**

*This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.*

**TUTORIAL  
ON  
NEURAL NETWORK APPLICATIONS  
IN HIGH ENERGY PHYSICS:  
A 1992 PERSPECTIVE\***

**BRUCE DENBY**  
*Fermi National Accelerator Laboratory  
M.S. 318  
Batavia, Illinois 60510 U.S.A.  
denby@fnal.bitnet*

**ABSTRACT**

Feed forward and recurrent neural networks are introduced and related to standard data analysis tools. Tips are given on applications of neural nets to various areas of high energy physics. A review of applications within high energy physics and a summary of neural net hardware status are given.

**Outline of Paper**

- 1. Architectures**
  - 1.1 Feed Forward Networks*
    - 1.1.1 Event Selection*
      - 1.1.1.1 Example 1 - Square Root Net*
      - 1.1.1.2 Example 2 - Tau Particle Selection*
      - 1.1.1.3 Example 3 - Secondary Vertex Selection*
    - 1.1.2 Function Mapping*
      - 1.1.2.1 Example 1 - Sin(x) Net*
      - 1.1.2.2 Example 2 - Drift Chamber z Position*
  - 1.2 Recurrent Nets and Tracking*
    - 1.2.1 Tracking with a Recurrent Net*
    - 1.2.2 Improvements to Tracking*
      - 1.2.2.1 Rotor Tracking*
      - 1.2.2.2 Elastic Tracking*
- 2. Tips on Applying NN in HEP**
  - 2.1 Choice of Variables*
    - 2.1.1 Offline Applications*
    - 2.1.2 Trigger Applications*
  - 2.2 Training*
    - 2.2.1 General Comments*
    - 2.2.2 Neural Network Software Packages*
    - 2.2.3 Low Level Pattern Recognition*
    - 2.2.4 Physics Process Determination*
    - 2.2.5 Example of a Hybrid Application - Isolation*
- 3. Survey of NN Applications in HEP**
  - 3.1 Track Segment and Vertex Finding*
  - 3.2 Quark/Gluon Separation*

---

\* Invited tutorial presented at the *Second International Workshop on Software Engineering, Artificial Intelligence, and Expert Systems for High Energy and Nuclear Physics*, La Londe les Maures, France, January 1992. To be published in the proceedings.

- 3.3 *Kink Recognition*
- 3.4 *An Assortment of Backprop Approaches*
- 3.5 *A Few New Approaches*
  - 3.5.1 *Z Branching Ratio*
  - 3.5.2 *Resonance Search*
  - 3.5.3 *Mass Reconstruction*
  - 3.5.4 *Non-Backprop Applications*
    - 3.5.4.1 *Introduction*
    - 3.5.4.2 *Jet Identification with Topological Map*
- 4. Neural Network Hardware**
  - 4.1 *Introduction*
  - 4.2 *Current Hardware Status*
    - 4.2.1 *True Analog Approaches*
    - 4.2.2 *Digital Approaches*
  - 4.3 *Prognosis for Triggers*
- 5. Acknowledgements**
- 6. References**

# 1. Architectures

## 1.1 Feed Forward Networks

### 1.1.1 Event Selection

Normally in high energy physics 'cuts' are used to select events of interest. Figure 1 shows the distribution of the variable,  $x$ , for two classes of events, class 'a' and class 'b'. We would like to use the variable  $x$  to allow us to classify a randomly chosen event as belonging to class 'a' or class 'b'. One way to do this is to simply place a cut at  $x'$  in the figure, and call everything to the left class 'a' and everything to the right class 'b'. The cut can be interpreted as a step function  $U(x-x')$  which has value 0 for  $x < x'$  and value 1 for  $x > x'$ . In cases where events are characterized by more than one variable, the optimal choice of cuts is less obvious. In figure 2, two consecutive cuts, 1 and 2, on the single variables  $X$ ,  $Y$  are inefficient at selecting class 'a', however, a single cut, 3, on a linear combination of  $X$  and  $Y$  is efficient. This can be interpreted as using a two dimensional step function  $U(aX + bY + c)$  as a classifier.

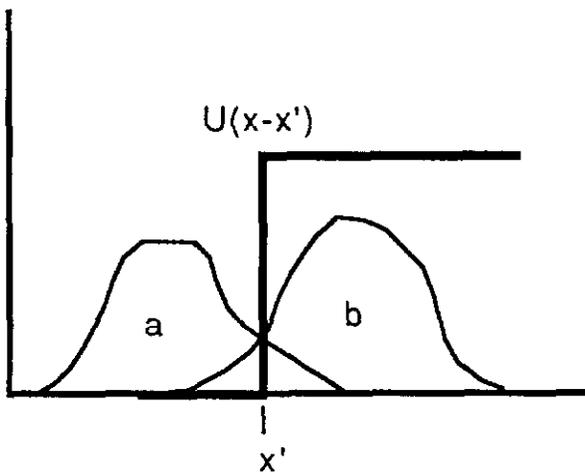


Figure 1

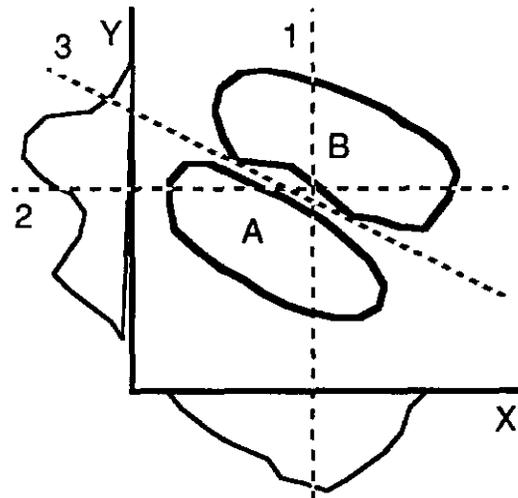


Figure 2

Figure 3 shows an even more complicated case: the boundary between the two classes is highly nonlinear. In this case, no linear cut can efficiently separate class 'a' from class 'b'. The ideal cut would be one like the first frame in figure 3, i.e., a cut with a curved edge.

One way to make such a cut is to build it up out of an ensemble of linear cuts. Frames 2 through 4 of figure 3 show the value over the  $X$ - $Y$  plane of a function made by summing one, two, and finally three step function cuts which approximate the curved edge cut we desire. In frame 5 we simply subtract 2 everywhere in the plane, and in the final frame, apply a final step function to the result. The function we have constructed is thus:

$$D = U( U(a_1x + b_1y + c_1) + U(a_2x + b_2y + c_2) + U(a_3x + b_3y + c_3) - 2)$$

This function has value 1 in the region containing the 'b' events and value 0 elsewhere; thus, it approximates well our desired nonlinear discriminant function.

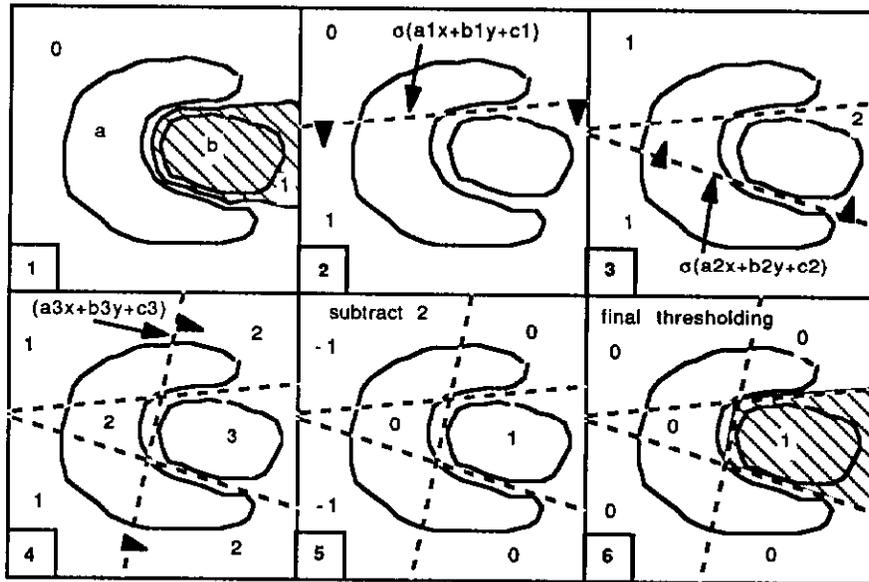


Figure 3

We can represent this function diagrammatically as in figure 4. The input variables  $x$  and  $y$  get multiplied by the coefficients  $a_1, b_2$ , etc. stored on the lines connecting units together. Summation occurs at the inputs to the step function units. The outputs of these units are again multiplied by coefficients, in this case 1, before the final summing and thresholding.

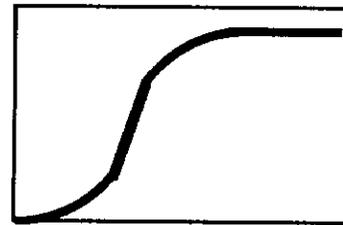
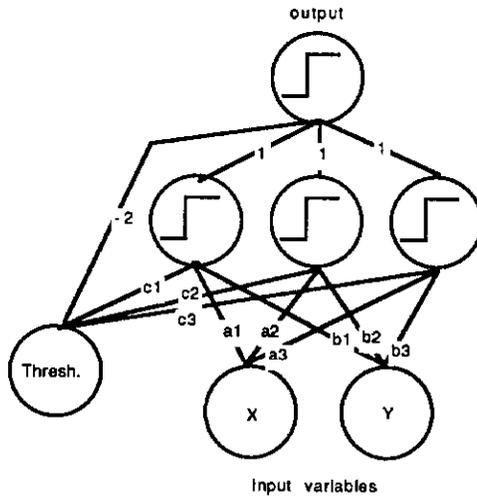


Fig. 4. Discriminant Function Architecture      Fig. 5. Sigmoid Function

Figure 4 looks very much like a standard three layer feed forward neural network. In the language of neural networks, this one has two units in the input layer, one each for  $x$  and  $y$ , three units in the hidden layer (represented by the step function units), one unit in the output layer which produces the final discriminant function, and one 'bias' unit, labelled 'thresh', for producing offsets. 'Units' are also called 'neurons'. The coefficients  $a_1, a_2$ , etc. form the matrix of neural network weights,  $w(j,i)$ , where  $i,j$  are the indices of neurons. In a feed forward neural network, each neuron performs the function  $t_j = \sigma(\sum_i w(j,i) t_i)$  where  $t$  is the output of a neuron,  $w(j,i)$  is the weight from neuron  $i$  to neuron  $j$ , and  $\sigma$  is the neuron

transfer function; in fig. 4, this is just the step function U. The difference between figure 4 and a standard neural network is that in the neural net, the hard step function U is replaced by a smoother, sigmoid transfer function as shown in figure 5.

The reasons for the sigmoid transfer function are twofold. First, the standard training procedure for neural networks, backpropagation<sup>1</sup>, requires that the derivative of the neuron transfer function exist. Neural networks can be trained, i.e., a data set which has already been classified, Monte Carlo data for example, can be used to derive the best values for the  $w(j,i)$ . If the initial values of coefficients,  $a_1, b_1$ , etc., are not quite right, backpropagation allows us to make corrections by varying the weights slightly and seeing the effect on the error the net makes. The sigmoids ensure that the 'error function' (defined below) varies smoothly rather than jumping from one value to another as would happen with a step transfer function.

The error function, E, is the sum over the network output units and over a training sample, of the deviation of the output values from their desired values. Gradient descent (a minimisation procedure) 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. We have:

$$E = \sum_p E(p) = \sum_{jp} [d(p,j) - t(p,j)]^2$$

where p is the index of an input pattern (i.e., an event in the training set), j is the index of an output neuron,  $d(p,j)$  is the desired output of neuron j in pattern p, and  $t(p,j)$  is its true output. The gradient with respect to a weight  $w(j,i)$  from hidden neuron i to output neuron j is then

$$E'(p,i,j) = [d(p,j) - t(p,j)] \sigma'(j) t(p,i)$$

where  $s'(j)$  is the derivative of the sigmoid function of neuron j. If  $w(j,i)$  instead is from an input unit i to a hidden unit j, we have

$$E'(p,i,j) = [\sum_k [d(p,k) - t(p,k)] \sigma'(k) w(k,j)] \sigma'(j) t(p,i)$$

where k runs over the output units. The prescription of backpropagation then is that, in each iteration

$$\Delta_p w(j,i) = -\epsilon E'(p,i,j) + \alpha * \text{previous } \Delta_p w(j,i)$$

where  $\Delta_p w(j,i)$  is the change in  $w(j,i)$  for this iteration,  $\epsilon$  is the distance to move along the gradient, also called the 'learning coefficient', and the term containing  $\alpha$ , the 'momentum' coefficient, is a smoothing term. Note that this expression explicitly contains the derivative of the transfer function  $\sigma'(j)$ . The total weight change is just the sum of the weight changes for the patterns presented. In practice, the weights are often updated after only a small number of 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. Typically several passes through the training set are necessary before E is minimised and a good set of  $w(j,i)$  are obtained. The minimum of E is only a local minimum, and different results may be obtained if a different set of starting weights is used. Normally, though, overall performance is reasonably independent of the initial weights.

Quality of training must always be judged based on performance of the network on a data set which is independent of the training set. Otherwise there is the danger of overtraining, in which the network begins fitting to noise in the training set. This is discussed further in section 2.2.1.

The second reason for the sigmoid is that for overlapping classes, the sigmoids can be used to approximate the probability of an event belonging to one class or another. The optimal classifier which takes into account the probabilities of an event belonging to one class or another is called the Bayes classifier<sup>2</sup>. In such a classifier, the ranges of the input variables are finely binned, and the probability of an event in a particular bin belonging to, say, class 'a', is equal to the number of events in class 'a' in that bin divided by the total number of events in the bin. How does this relate to the neural network output? Recall that during training, an error function is minimized<sup>3</sup>:

$$E = \sum (t(i)-d(i))^2$$

$$E \cong \sum \alpha_a P_a(i) (t(i)-1)^2 + \alpha_b P_b(i) (t(i))^2$$

for two classes 'a' and 'b', where the desired output is 1 for class 'a' and 0 for class 'b', and where  $\alpha_a$  and  $\alpha_b$  are the fractions of classes 'a' and 'b' in the sample,  $P_a$  and  $P_b$  are the probabilities for an event  $i$  to belong to class 'a' and 'b' respectively. The second line is an approximation valid in the limit of a very large training set. If we differentiate with respect to  $t(i)$  and set each term to zero we get:

$$\alpha_a P_a(i) (t(i)-1) + \alpha_b P_b(i) t(i) = 0$$

$$t(i) = \alpha_a P_a(i) / (\alpha_a P_a(i) + \alpha_b P_b(i))$$

$$t(i) = n_a / (n_a + n_b) .$$

That is, the function  $t(i)$  which minimizes the error function is that which maps each event onto its Bayesian probability to be in class 'a'. Because a very large number of bins is sometimes required, a true Bayes classifier can be difficult to construct and use. The neural network will do the best job it can if the network output can be made as close as possible to the Bayesian probability. It can be shown that a three layer feed forward neural network trained with backpropagation approximates a Bayes classifier<sup>2</sup>; the accuracy of the approximation depends upon the number of hidden units, but normally a relatively small number is sufficient. We can now extend the ideas in figure 3 to the case of continuous valued sigmoid neurons. Rather than simply selecting a particular region of input variable space, the output of a feed forward neural network, constructed out of the sum of sigmoids, approximates, over the volume of input variable space, the probability of an event's being in each class.

#### *1.1.1.1 Example 1 - Square Root Net*

The coefficients  $a_1, b_1$ , etc., and thus the 'orientations' of the cuts, are determined from the data during the backpropagation process. It is instructive to see where the hidden units end up in a toy problem which was trained with backpropagation. The 5 hidden unit network shown in figure 6 was trained to recognize when its two inputs  $X$  and  $Y$  are related by  $Y = \sqrt{X}$ . The output should be 1 when this condition is satisfied and 0 when it is not. It is

interesting to note that a linear classifier (a cut on a single linear combination) will fail miserably on this problem.

Figure 7a shows the curve we are trying to select, isocontours of the trained net output, and lines labeled to indicate the hidden units they represent. Figure 7b shows the value of the network output over the plane. It is clear that the net has positioned the hidden units so as to select reasonably well the region where  $Y = \sqrt{X}$ . With additional hidden units, it could have done better by cutting out the regions at small X and large X where the network currently makes a mistake.

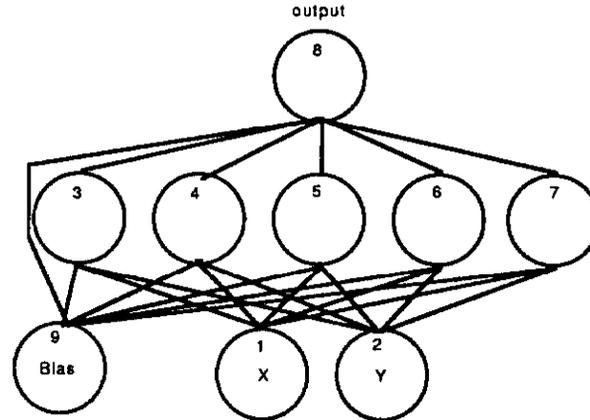


Figure 6. The square root net.

### 1.1.1.2 Example 2 - Tau Particle Selection

Figure 8 shows a two dimensional classification problem drawn from high energy physics<sup>3</sup>. A network was trained to choose the polarity of tau particles from their decay angles, psi and phi. Figure 8a shows the distribution of decays for positive helicity, 8b for negative helicity, and 8c the decision boundary found by a neural network with 4 hidden nodes. Shown also is the decision boundary of a Bayesian classifier for this problem. The neural net approximates very well the Bayes classifier.

### 1.1.1.3 Example 3 - Secondary Vertex Selection

Another example of event classification in a two dimensional space in high energy physics comes from the study of secondary decay vertices in the NAXX experiment at CERN (see figure 9)<sup>4</sup>. The silicon microstrips measure the trajectories of particles from primary and secondary vertices. Associative memories give a list of the track parameters, D (impact parameter), and phi (track angle) for the tracks in each event. The track parameter list can then be used to try to identify whether the event contains a secondary vertex. The study shown here is based on Monte Carlo data, but the motivation is to assess the feasibility of using a hardware neural network in a trigger to find secondary vertices online. Figure 10 shows the distribution in D-phi space of the tracks for three signal events (i.e., containing secondary vertices) and three background events. The signal and background are not very different but the signal events are somewhat broader in D. Two position independent moment variables,  $\eta_{02}$  and  $\eta_{11}$  were chosen, where

$$\eta_{02} = 1/M \sum (D-D^0)^2 \quad \text{and} \quad \eta_{11} = 1/M \sum (\phi-\phi^0)(D-D^0)$$

and  $D^0$  and  $\phi^0$  are the mean values of D and phi, and M is the number of tracks. Figures 11 a and b show the distributions in  $\eta_{02} - \eta_{11}$  space of the signal and background events. Note

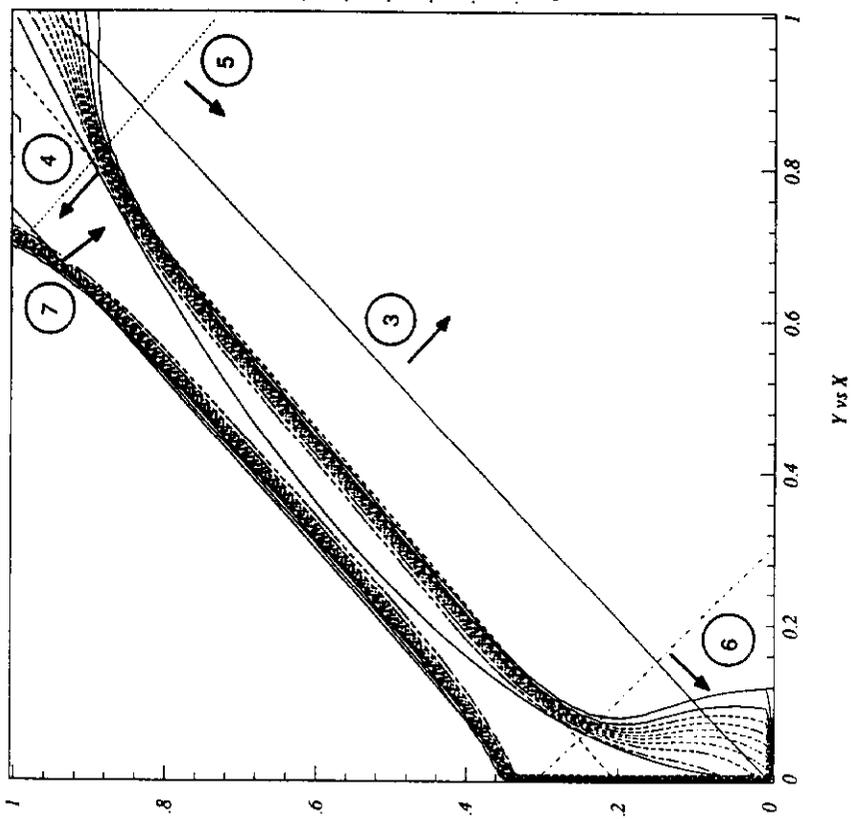


Figure 7a.

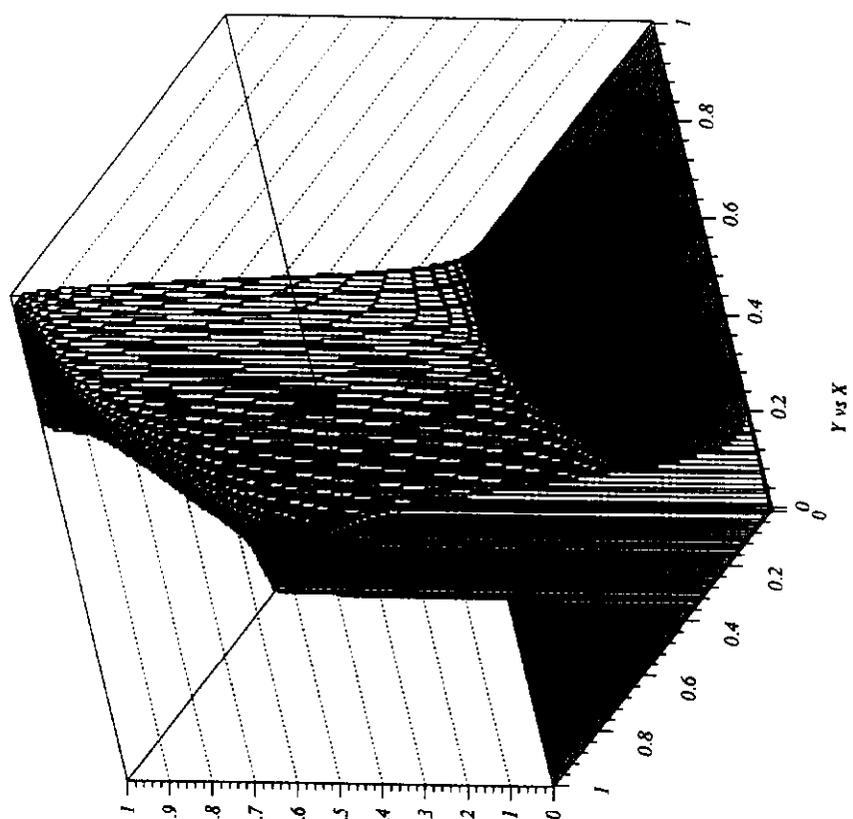
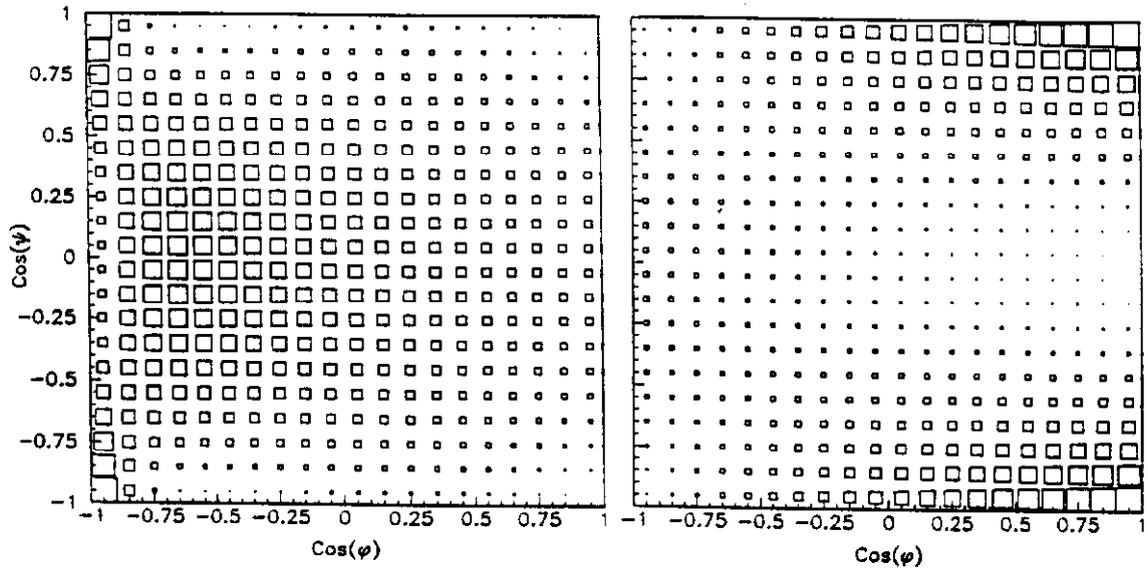


Figure 7b.



Angular distribution for a  $\tau^-$  with negative helicity. Angular distribution for a  $\tau^-$  with positive helicity

Figure 8a.

Figure 8b.

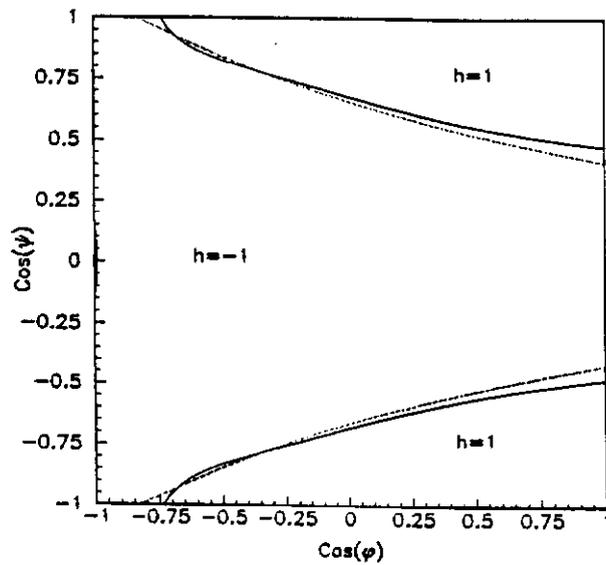


Figure 8c.

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

the logarithmic scale. Table I shows the classification results, based on a test set of 2400 events, for a neural network trained to tell signal from background based upon the two moment variables.

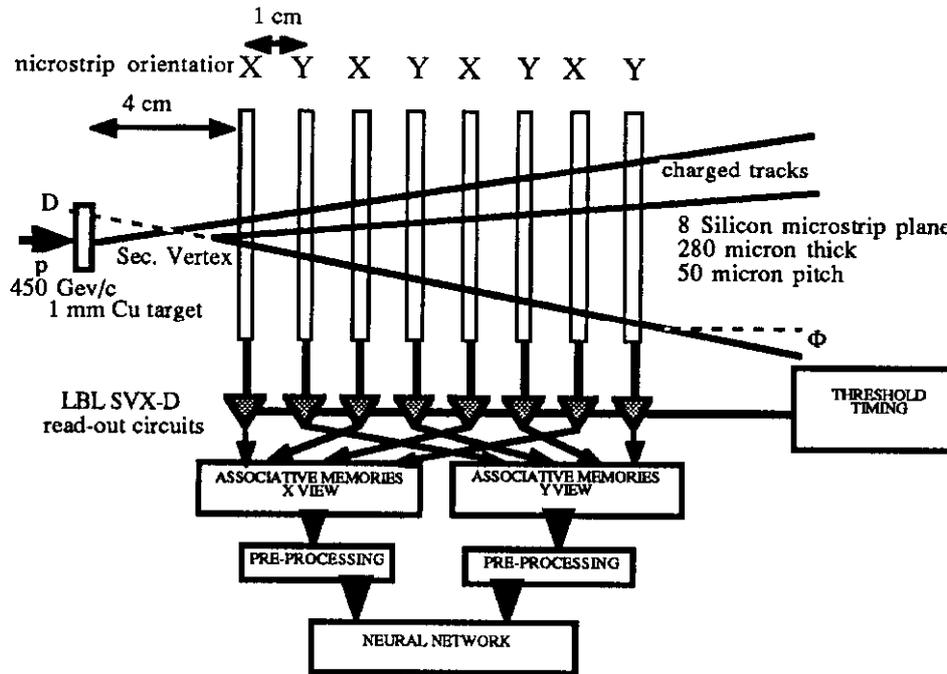


Figure 9. NAXx layout and readout scheme

Table I

	$P_f$	$P_m$
Neural Network	7/2400	461/2400
Nearest Neighbor	27/2400	523/2400

$P_f$  is the probability of calling a background event a signal event and  $P_m$  is the probability of calling a signal event a background. Since the background is many times more common, it is important to keep  $P_f$  as small as possible even at the expense of increasing  $P_m$ . In Table I, the neural network result is also compared with a standard nearest neighbor classifier<sup>2</sup>; the network performance is better.

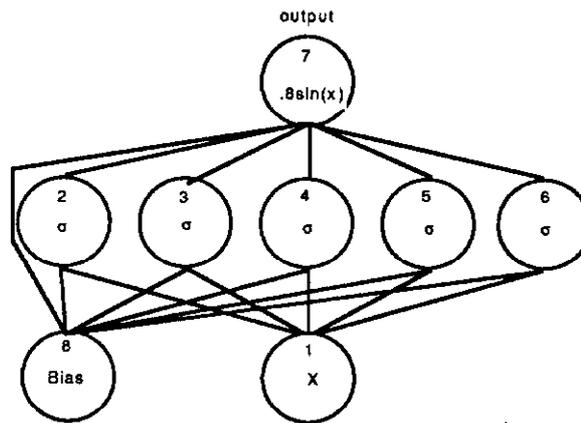
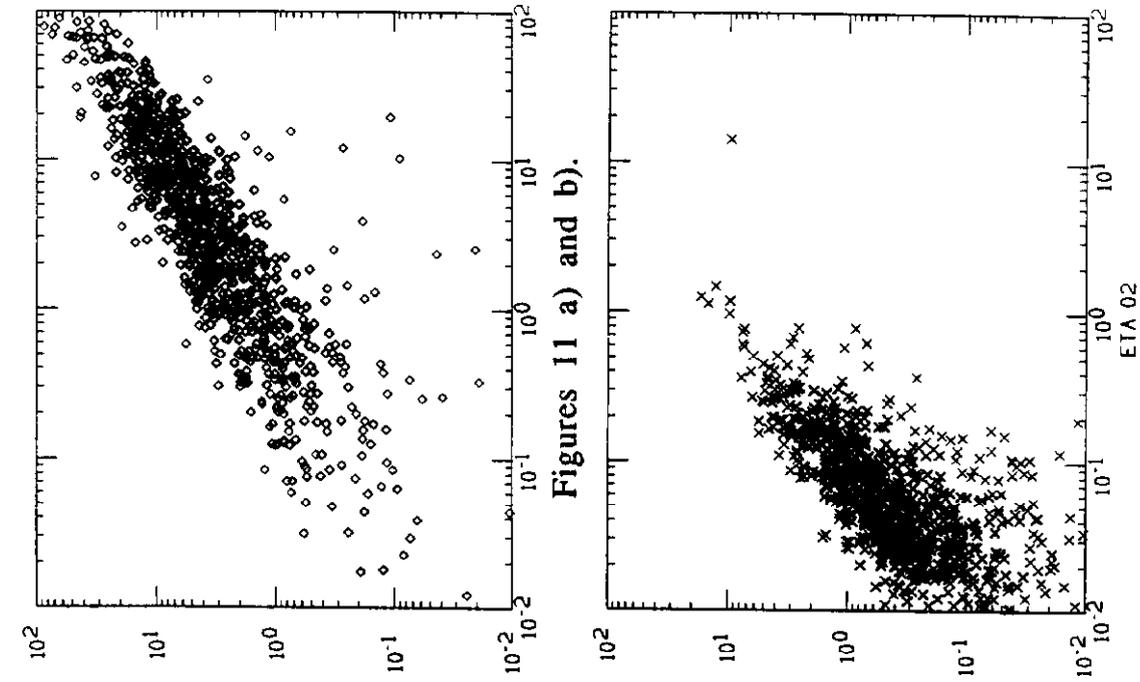


Figure 12. Net to make  $\sin(x)$  from  $x$ .



Figures 11 a) and b).

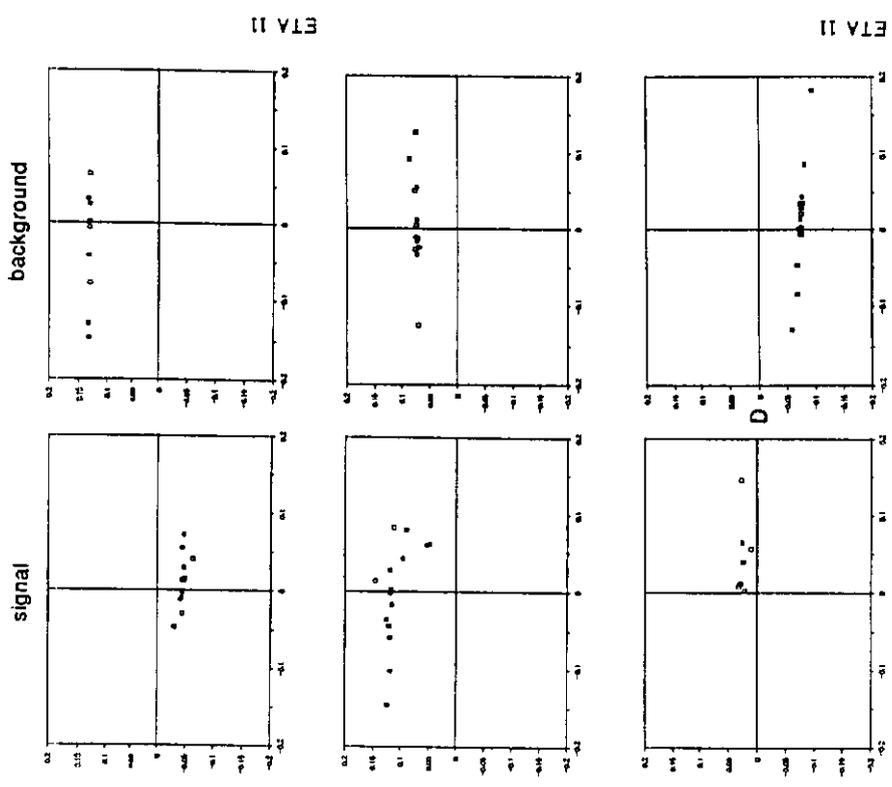


Figure 10.

### 1.1.2 Function Mapping

Neural networks can also be used to map inputs into functions of the inputs. In this case the role of the hidden units is rather different. It turns out that the network, during training, uses the shapes of the sigmoid units to build up the desired function.

#### 1.1.2.1 Example 1 - Sin(x) Net

A simple example of this is taken from reference 5 in which a network with 5 sigmoid hidden units was trained to map  $x$  to  $\sin(x)$ . The architecture used is shown in figure 12. The network was trained with backpropagation. Figure 13 show the function of each of the 5 sigmoids in building the final  $\sin(x)$  function. Most of the work is done by three sigmoids which have been translated to form the bumps in the sin function. The remaining two hidden units just correct for small variations.

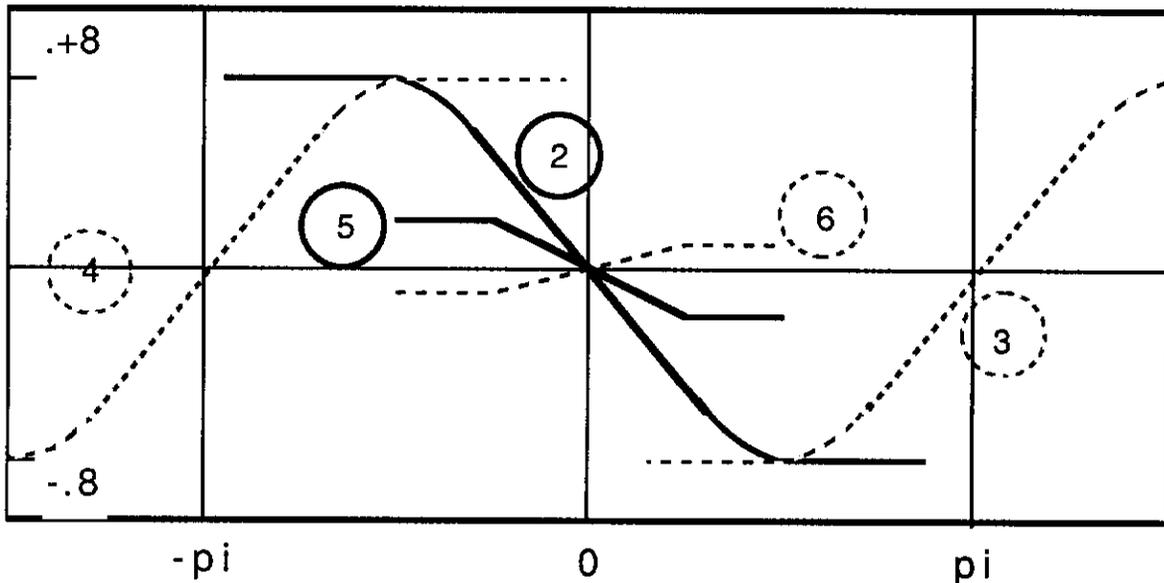


Figure 13. The function of the hidden units for  $\sin(x)$  problem.

#### 1.2.2.2 Example 2 - Drift Chamber z Position

A second example of function mapping with neural networks is a high energy physics application: determining the  $z$  position of a track which passes through a drift chamber<sup>6</sup>. The position perpendicular to the wire is well determined by the drift time, however, induced charge in cathode pads must be used to infer information about the position along the wire,  $z$ . The geometry of the chambers is shown in figure 14.

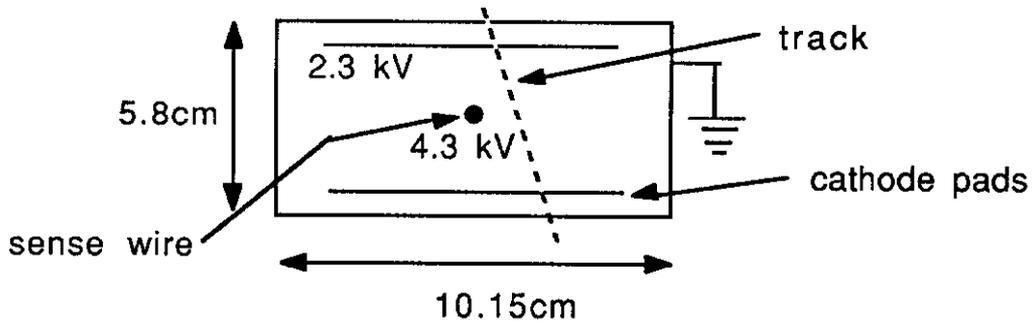


Figure 14. Drift chamber geometry

The pads are etched in a diamond pattern as shown in figure 15a. When a particle passes through the chamber, the avalanche induces differing amounts of charge on the inner ( $q_a$ ) and outer ( $q_b$ ) pads.

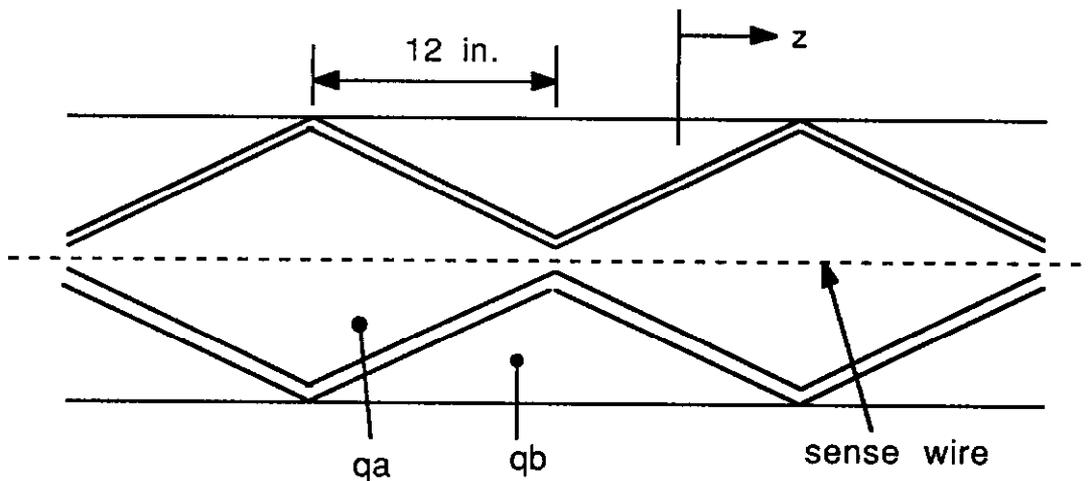


Figure 15a. The cathode pads.

The relationship between  $q_a$ ,  $q_b$ , and  $z$  has been measured and is shown in figure 15 b. The empirical functional relationship is:

$$z = -.136 + \text{sqrt}(.405 + .71(q_a - q_b)/(q_a + q_b))$$

Timing information tells which cusp of the function we are in; then, the charges can be used to get the final value of  $z$ .

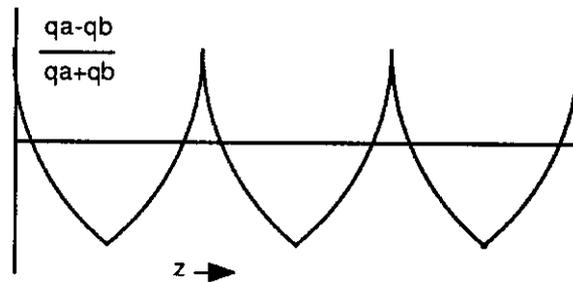


Figure 15b. Charge ratio as a function of  $z$

In this application, a feed forward net with five hidden units was trained to calculate the  $z$  function from the two charges  $q_a$  and  $q_b$ . Once the weights were determined, a circuit was built to execute the network, in which the weights were represented by resistors. The circuit is shown in figure 16.

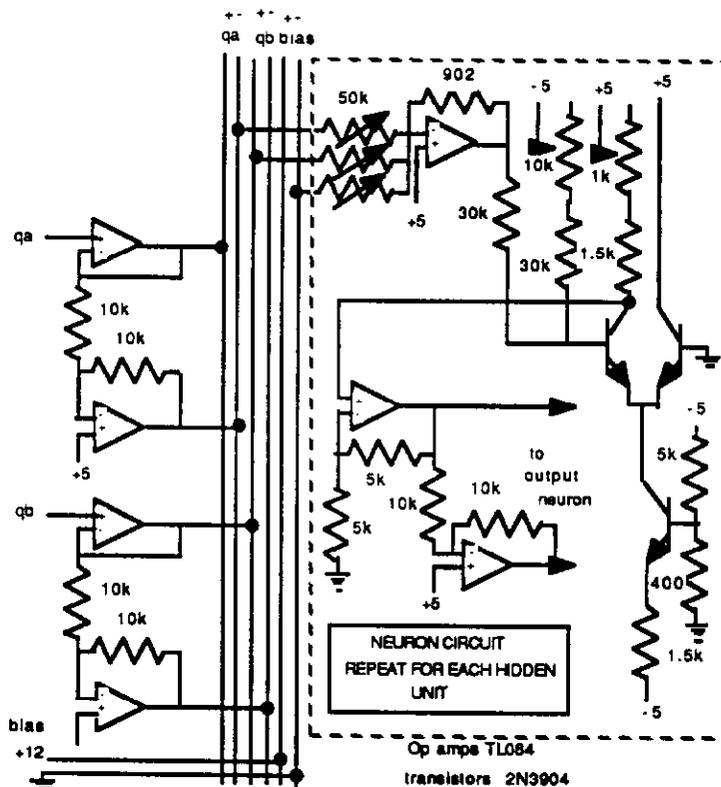


Figure 16. The pad net circuit.

Figure 17 shows a comparison of the  $z$  value calculated online by the neural network (using real particles) with that calculated offline using the two charges, which were digitized and stored on each event along with the neural net output. The relationship is quite linear.

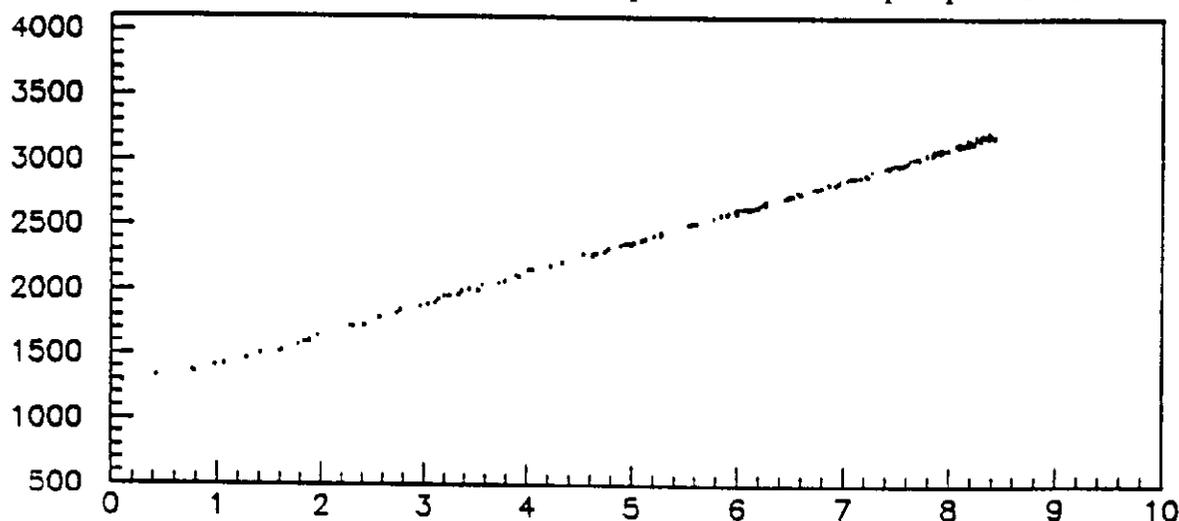


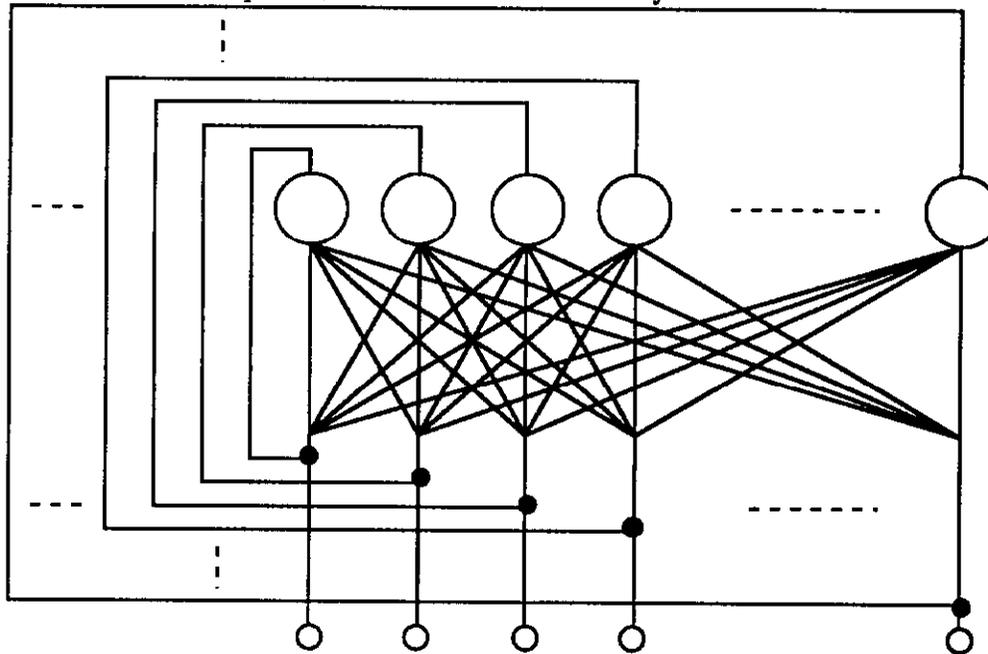
Figure 17. Neural Net position (counts) versus offline position (cm.).

### 1.2 Recurrent Networks and Tracking

The basic recurrent network architecture is shown in figure 18. The neuron outputs are fed back into the inputs. Recurrent networks have dynamical behaviour, with activation values settling to fixed values after a few cycles through the network. If we choose the connection strengths to be symmetric, i.e.,  $w_{ij} = w_{ji}$ , and non-diagonal, then an energy function,  $E = -1/2 \sum w_{ij} o_i o_j$ , where  $o_i$  is the output of neuron  $i$ , is constantly minimized as the network evolves. A clever choice of the  $w_{ij}$  can produce networks that evolve to a desired steady state solution. In practice, the evolution of the system is obtained by iteratively solving the update equations:

$$\tau \frac{du_i}{dt} = \sum_j w_{ij} o_j - u_i ; o_i = \text{sigmoid}(u_i).$$

On each iteration,  $dt$  is kept  $\ll \tau$ , the time constant of the system.



Recurrent Network inputs

Figure 18

#### 1.2.1 Tracking with a Recurrent Net

The most common application of recurrent networks in high energy physics is for track reconstruction, using an algorithm developed by Denby and independently by Peterson<sup>7,8</sup>. In this application a neuron is defined to be a directed link between two hits in a tracking detector. The weight connecting two neurons  $i$  and  $j$  is determined by the angle  $\theta_{ij}$  between them, (figure 19):

$$w_{ij} = A \cos^n \theta_{ij} / l_{ij}$$

where  $l_i$  and  $l_j$  are the lengths of the neurons (i.e., distance between hits), if  $i$  and  $j$  do not both point into or out of the same point, and  $w_{ij} = -B$  if  $i$  and  $j$  are head to head or tail to tail. The energy function will be smallest when the angles between close together neurons sharing points are small. This favors neurons lying along smooth trajectories such as those of particles moving in a magnetic field. The constraint term  $-B$  ensures a unique direction to the tracks to avoid a degeneracy which prevents settling of the network.

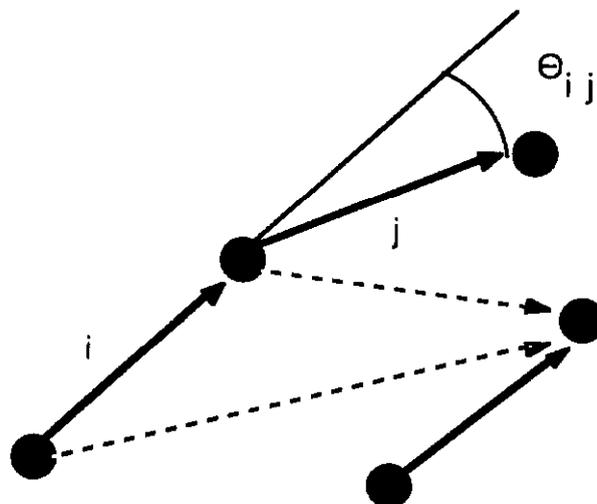


Figure 19 Neuron links in the Denby-Peterson Net

This method has now been used on real data at the ALEPH experiment at CERN<sup>9</sup>. Figure 20 shows  $r$ - $\phi$  (i.e., looking down beam line) and  $r$ - $z$  (side) views of a  $Z$  event with all links defined before network evolution (left side of figure), and the event after settling of the network, with tracks found (right side). The efficiency is as good as the conventional track reconstruction program but the neural net algorithm is somewhat faster. In this same reference, a study was made of execution time for the neural net and conventional algorithms as a function of track multiplicity. This is plotted in figure 21. The advantage of the neural algorithm is shown to increase with multiplicity.

Considerable effort was needed to make the neural algorithm competitive with the conventional algorithm in execution time. Another way to speed up the original algorithm is to vectorize it<sup>10</sup>. However, there does not seem to be a straightforward way to implement this algorithm in the fast hardware that would be needed to make it applicable at the trigger level. The main reason for this is that the weights (i.e., the  $\cos^n \theta_{ij} / l_i l_j$ ) must be recalculated for each event. Also, the number of neurons and weights is high. In addition, it is clear that the algorithm does not take advantage of all the available information, such as that tracks are known to be nearly perfect helices. This makes the algorithm more susceptible to noise since it will be less able to reject outliers. This algorithm, however, is very appropriate for applications where the tracks are not easily parametrized, such as in non-uniform magnetic field, or in the case of decays in flight.

## 1.2.2 Improvements to Tracking

### 1.2.2.1 Rotor Tracking

One proposed improvement on the neural tracking algorithm is the so-called rotor tracking<sup>11</sup>. In this formalism, each hit is assigned a Potts neuron<sup>12</sup>, i.e., a little rotor with a length and an orientation (figure 22). The obvious advantage is that the number of neurons is now  $N$

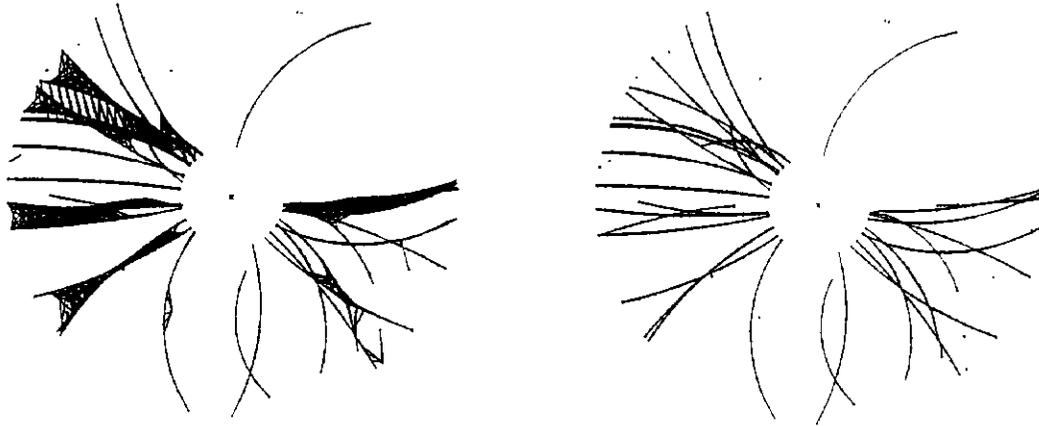


Figure 20.

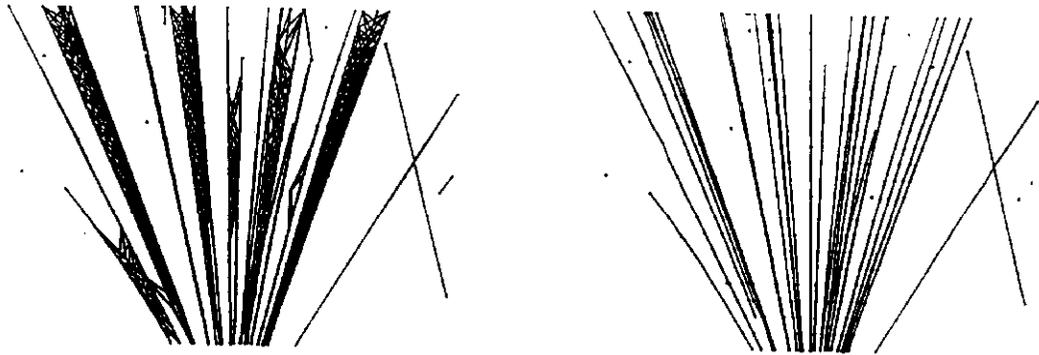
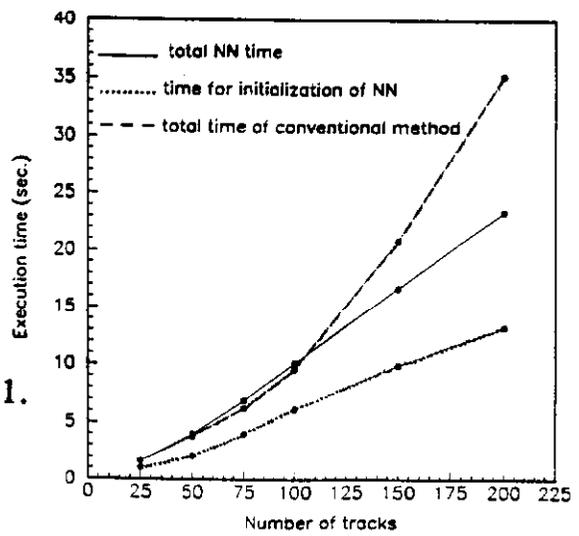


Figure 21.



instead of  $N^2$ , and the number of connections thus  $N^2$  instead of  $N^4$ . The interaction energy between the rotors is defined to be:

$$E = -1/2 \sum s_i \cdot s_j / |r_{ij}|^m + a (s_i \cdot r_{ij})^2 / |r_{ij}|^m ,$$

where  $s_i$  and  $s_j$  are the Potts variables,  $r_{ij}$  is the vector between hits  $i$  and  $j$ ,  $m$  is a constant of order 2-5, and  $a$  is a constant to be determined empirically (figure 22). The first term tends to align the rotors with one another, and the second term tends to make a rotor point at its neighbor's pivot point. The equations of motion, which must be solved iteratively, are:

$$u_i = -dE/ds_i \quad s_i = u_i / |u_i| \text{ sigmoid}(|u_i|)$$

followed by a 'greedy' algorithm<sup>11</sup> to select the links nearest the rotors.

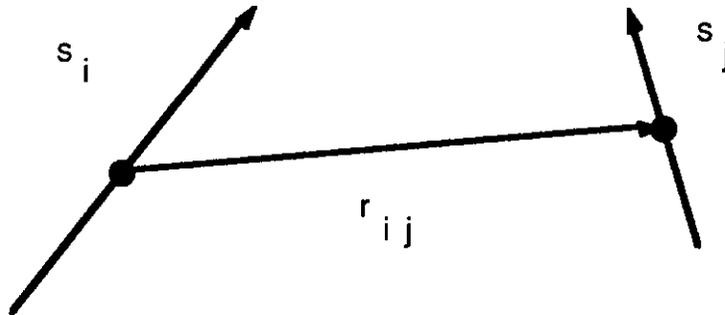


Figure 22. Rotor Neurons

The result of trying this method on a toy problem is shown in figure 23. Unfortunately, the interference between neighbors spoils the results for anything beyond toy problems. Further improvements are being worked out.

### 1.2.2.2 Elastic Tracking

A second improvement to the neural tracking is in the so-called elastic tracking<sup>13</sup> or deformable templates<sup>14</sup> approaches. In these approaches, a track is a helical object which settles into a shape which best fits the hits. The helix can be thought of as electrically charged and attracted to the hits which have opposite charge. Although these algorithms map the tracking problem onto dynamical systems, and are at least in principle parallelizable, they have lost some of the 'neural' flavor of the original Denby-Peterson net. Nonetheless, the efficiency and robustness to noise of the elastic methods are excellent. Figure 24 shows the result of applying a set of 'elastic arms' to a simulated event in the Delphi TPC. One interesting study<sup>13</sup> compared the robustness to noise of the standard 'roadfinder' method, the Denby-Peterson net, and the elastic tracking method. Figure 25, from this study shows the efficacy of each method as a function of number of tracks. All data have 20 percent noise and 3 percent error on position measurement. The roadfinder breaks down between 5-10 tracks, the Denby-Peterson net at 10-15 tracks, but the elastic tracking always finds the correct answer.

## 2. Tips on Applying NN in HEP

### 2.1 Choice of Variables

#### 2.1.1 Offline Applications

For offline applications of neural networks in HEP, it is advisable to choose input variables which are relevant to the problem from a physics standpoint. For instance, for quark/gluon separation, jet shape variables might be chosen due to theoretical ideas about color charge, or

from experience in electron-positron collider experiments. The reason for this is that, although a network can be trained to calculate any variable from whatever input variables one decides to use, if one has a priori knowledge, much can be gained in terms of network complexity, training time, and number of examples. It is also true that if the 'intelligent' variables do not enable to separate two classes, then 'unintelligent' variables probably will do no better. If on the other hand the intelligent variables do prove effective, subsequent nets can still at that point be used to calculate them from raw quantities if, for instance, it is desired to try to incorporate the decision making into an online trigger.

The network should always be kept as small as possible to ensure adequate training, especially if the training set is small. As a rough rule of thumb, for a net with 10 inputs, 10 hidden and 1 output, about 2000 examples are required. It may be tempting to include as many variables as possible in hopes that the network will find some 'secret' correlation that a human could not. It is better, though, to begin with intelligent variables and after some separation appears, to try adding additional variables to see if they help. Very large fully connected networks can have problems with training convergence. For large networks, it is advisable to use structured networks with local receptive fields<sup>51</sup>.

### ***2.1.2 Trigger Applications***

In the case of the trigger, input variables must be strictly limited to those which are definitely available at the trigger level and with the accuracy available at the trigger. It is no good designing a trigger which operates in 1 nanosecond if 3 seconds of Vax time are needed to calculate the inputs to the trigger. Projects involving neural nets for triggering are best done in conjunction with people who are intimately involved in the triggering of the experiment in question. It turns out that most triggers were built using tricks to make them more buildable; for instance, coarser granularity, time multiplexing of signals, multihit capabilities, special readout schemes. These can make just getting the signals to the neural net a formidable task.

Consider the level-2 trigger for the CDF experiment. Three neural network calorimeter triggers are being planned for the 1992 run of the CDF experiment at Fermilab<sup>15,16</sup> (one of these will be discussed in more detail in a later section). All will operate on clusters of energy found in the calorimeter, where each tower is represented by an analog voltage. In principle, the system is a 'single chip solution' since the pattern recognition necessary for triggering is available on a single neural network chip. Nevertheless, it has been necessary to build special signal tap cards to extract the signals from the existing trigger and bring them to special new matrix shifter boards which allow a cluster found by the cluster finder to be selected and presented to the neural network. Two additional control boards are also necessary to coordinate the timing of the trigger. In the trigger, calorimeter towers are ganged together into coarser towers which are used in the trigger tower array in order to provide a manageable array of 24 by 42 trigger towers. Finally, the use of analog voltages implies a limited precision on the energies.

These compromises limit the effectiveness of pattern recognition which networks might be able to do. For instance, most of the information on the lateral shape of electromagnetic showers is lost due to the coarse granularity. This information, if available, would have allowed the network to do very good electron identification in the trigger. Future triggers may be able to make use of such information.

## ***2.2 Training***

### ***2.2.1 General Comments***

A number of hidden units incommensurate with the size of the training set can lead to overtraining of the network, i.e., the net will begin to memorize the training set and will not generalize well. The indication of overtraining is a network which continues to improve in its

performance on the training set but whose performance on an independent test set begins to deteriorate. This is analogous to fitting a curve to a set of points. If the function used has many parameters, it will always be possible to fit exactly to all the points, including noise points. The curve found, however, will interpolate poorly between the data points if the true parent curve was actually a simpler, smoother function.

Even when great care is taken, limited training set size can be a problem. In reference 17, fifty-five input variables were used to attempt to separate hadronic top events from QCD background. Three hundred neurons were used in training. The result was that, because there was not enough data to adequately train the network, the net performance was 4 times worse than that obtained using a simple linear discriminant. It is unfortunately the case that even though the neural net is a powerful classifier, if it is not possible to generate a sufficiently large training set for the net, the network will not be trainable and will perform poorly. A simpler, linear classifier may do better even with its known limitations.

### ***2.2.2 Neural Network Software Packages***

The easiest way to begin to learn about neural networks is to play with one of the commercial packages which have nice graphics and user interfaces. MimeNice from the Mimetics Company<sup>18</sup> and NeuralWorks Professional II from NeuralWare Inc.<sup>19</sup> are good. BrainMaker from California Scientific Software<sup>20</sup> and DynaMind from NeuroDynamX<sup>21</sup> have the advantage of interfacing with the Intel INNTS Development System<sup>22</sup> for the ETANN chip which we will discuss later.

Most of these simulators are written in C and the source code is not always available. Up till now, most of the applications in high energy physics have been done using homemade simulators written in Fortran, such as the Fermilab simulator<sup>23</sup> and the Lund simulator JETNET<sup>24</sup>.

### ***2.2.3 Low Level Pattern Recognition***

Neural nets can be used to find low level patterns in detectors, such as clusters, tracks, etc., Normally, clusters and tracks are fairly well understood objects, and if the detector simulation is well understood as well, there will be no problem in generating enough data to train a network to recognize them. In many cases, real data will be available, e.g., from a test beam, for training. Some examples of these types of applications will be given later.

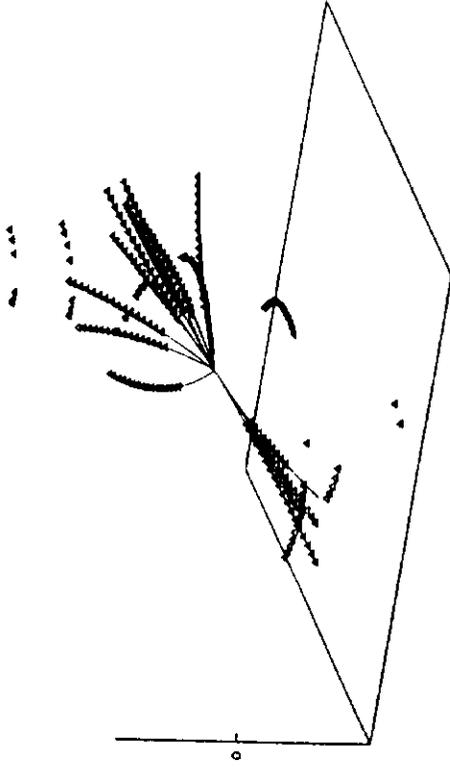
### ***2.2.4 Physics Process Determination***

In contrast, if the pattern recognition consists of recognizing a particular physics process which may or may not be present in an event, the situation is much more difficult. Often it is a rare, as yet unseen process upon which one wants to trigger. In this case, since the process has never been seen, one doesn't know exactly what to look for. The training set in this case must of necessity be based upon models of the process. The effect of this model dependence of the training is difficult to assess.

It may seem attractive, in order to reduce dependence upon Monte Carlo data, to use real data as the background sample in the training set, since usually backgrounds are much better understood. This however is dangerous since if, in the training set, some events are Monte Carlo (the signal) and some are real data (the background) the network may find a way of distinguishing Monte Carlo data from real data which has nothing to do with the desired discrimination between signal and background.

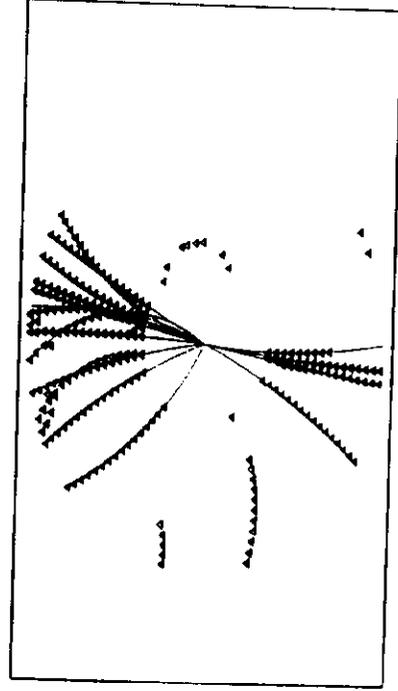
It is thus 'safer', when trying to create a discriminant, to use Monte Carlo both for the signal and the background. Here again, though, any results will be dependent upon the model

3-D view



(a)

xy view



(b)

rotor tracking

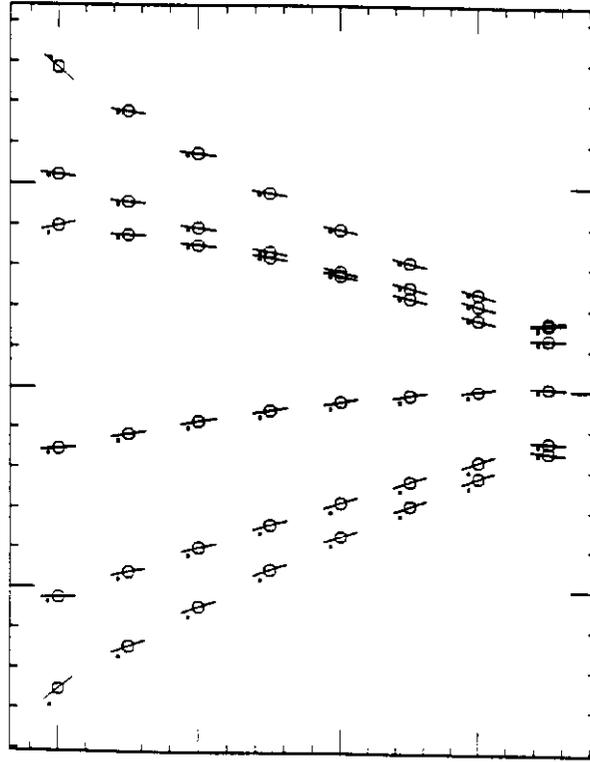


Figure 23.

Figures 24 a) and b).

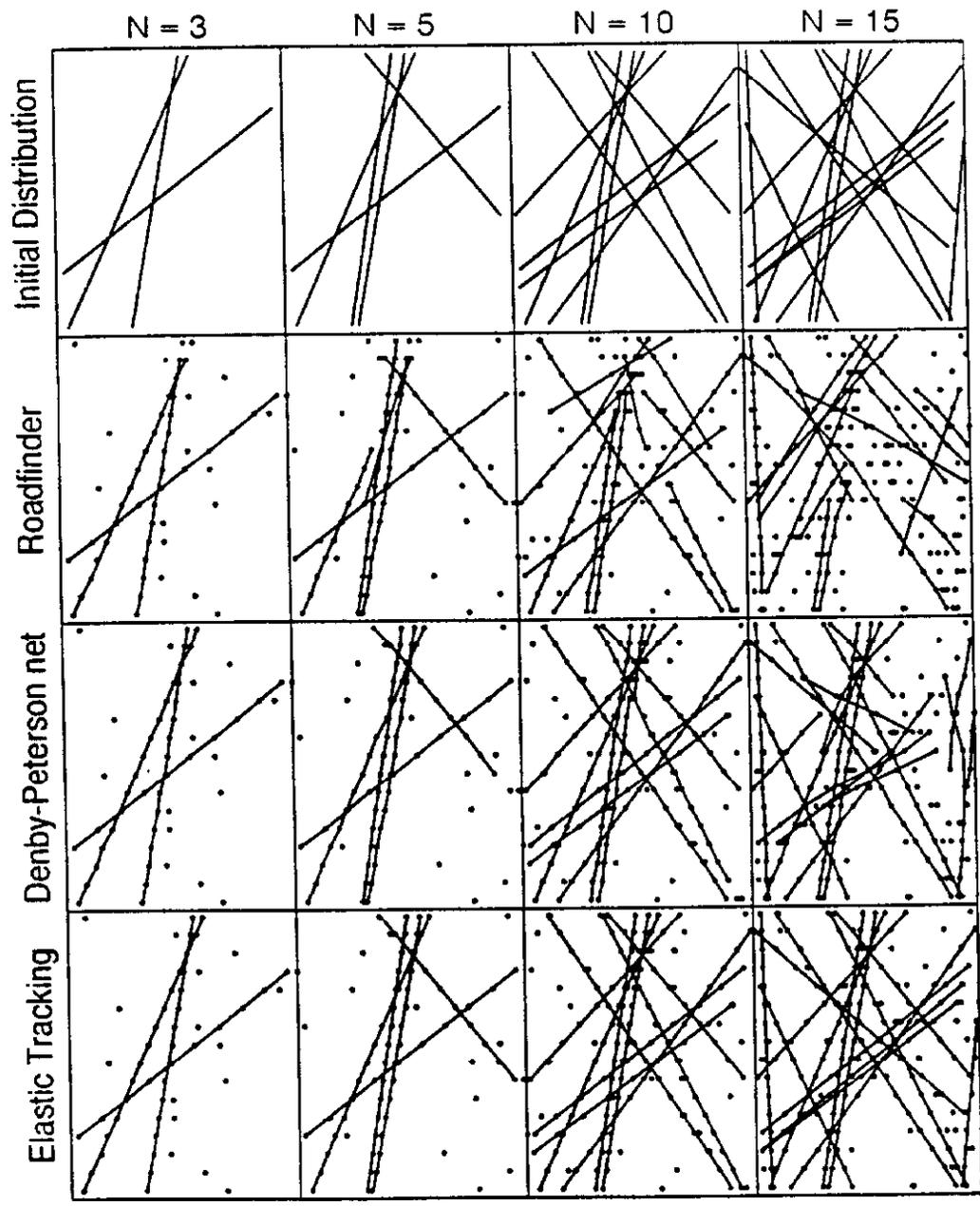


Figure 25.

used. This is perhaps the most serious problem to be faced in the use of neural networks in physics process determination. Any algorithm based upon Monte Carlo will be model dependent, but because a neural network stores the parameters of its discrimination process in a matrix of weights, interpretation of this dependence may be much more difficult. Not enough work has been done to determine how to understand the model dependence when using neural networks.

Another 'disadvantage' of neural networks is that a relatively large amount of training data is required in order to assure good generalization. Thus, a great deal of Monte Carlo data must be generated to assure adequate training. Sometimes the generation of this data is quite time consuming, as was seen in<sup>17</sup>. Here, the QCD multijet background to top production took 5 CPU hours of Cray X-MP/48 time.

### 2.2.5 Example of a Hybrid Application - Isolation

We show here a hybrid application: an isolated endplug electron trigger<sup>25</sup>. This is one of three neural network triggers to be installed at CDF for 1992. Electrons from W decay are normally isolated in the calorimeter. The approach is hybrid in the sense that it attempts to select a particular physics process, namely leptonic decay of the W, but bases the discriminant upon a rather low level pattern, i.e., that of an isolated electromagnetic cluster. In the endplug, tracking is not available in the trigger, so other means must be used to bring down the rate of background from pi-zeroes in jets. In the past, a higher energy threshold was used in the plug for this purpose. In 1992, isolation will be tried in the level 2 trigger to allow the same rate at a lower threshold. As we shall see, the neural network executes exactly the algorithm normally used offline for isolation (except that it operates on trigger towers rather than offline towers).

The trigger operates (as will all of the CDF neural net triggers) upon a 5 by 5 cell array of trigger tower energies. Four templates are defined as shown in figure 26. The cells have size  $15^\circ$  by  $.2$  units of rapidity.

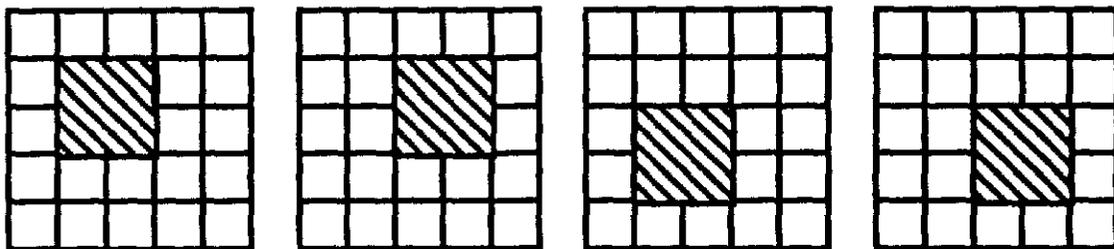


Figure 26. Isolation templates for plug electron trigger.

The dark central region is meant to contain the electromagnetic shower. Four templates are necessary since the shower may spill over in to 2 to 4 towers and since the center of the tower as found by the cluster finder may not perfectly center it in the 5 by 5 array in all cases. Each template is represented as a hidden unit in the network. Each tower has a weight connecting it to a hidden unit in the neural network, figure 27. Cells in the central region have a weight of 'frac', and cells in the outer region have a weight of -1. Thus, the quantity presented to the hidden units, which are used just as comparators, is

$$\text{frac} * E_{\text{inner}} - E_{\text{outer}}$$

If this quantity is negative, the hidden unit will not 'fire': the energy outside the electromagnetic shower was greater than some fixed fraction of the shower energy and the shower is thus not isolated. If the quantity is positive, the neuron fires, indicating an isolated

cluster. If any of the templates fires, the cluster is isolated. In typical offline applications,  $\text{frac}$  has a value of .1 to .2. The value  $\text{frac} = .16$  was found to be optimum in the present application.

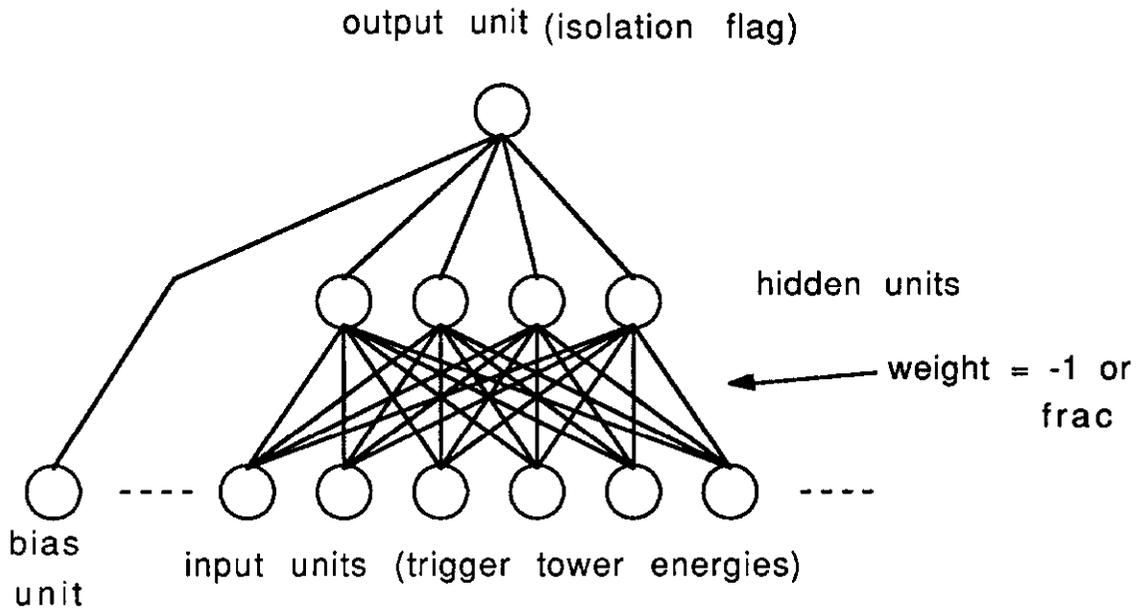


Figure 27. Net architecture for isolation trigger

A simulation of this trigger operating on real data from a previous run indicated a 4 fold reduction in background while retaining 95% efficiency on electrons from W.

### 3. Survey of NN Applications in HEP

In this section we survey applications of NN in HEP. We have in fact already discussed several: track reconstruction, secondary vertex finding, an isolation trigger, finding z position from drift chamber pads, polarization of the tau particle, finding top amidst the QCD background. The reports on those applications presented at this conference will be brief; refer to the proceedings for details.

#### 3.1 Track Segment and Vertex Finding

Several papers have covered this subject<sup>26,27,28,29</sup>. Here we discuss reference 27 in which real data from a collider experiment, E735 at Fermilab, were fed to a simulated neural network trained to find the primary vertex of the event based upon drift times in the z-chamber, a drift chamber with three layers of wires placed near the beam pipe. Figure 28 shows the hits in the chamber for a typical event; here, only the hit wires are shown, not the drift times. By eye, the hits seem to emerge from a point on or near the beam line.

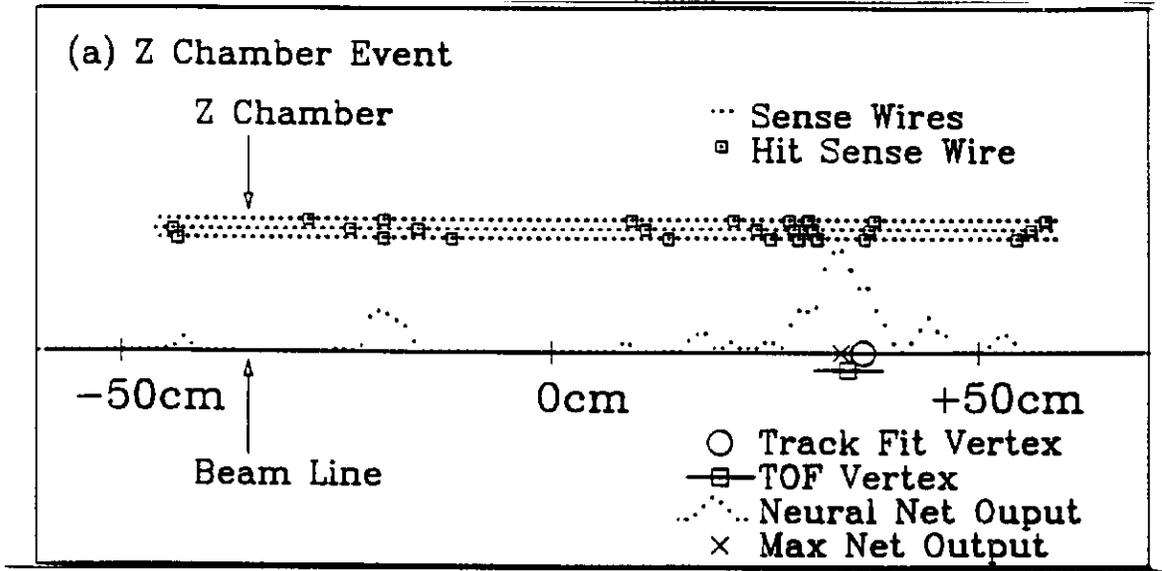
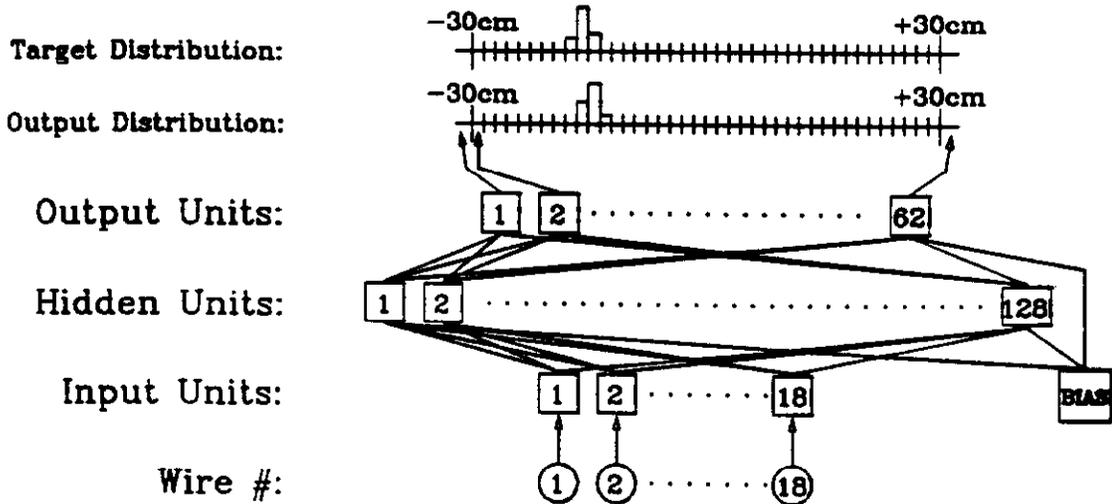


Figure 28

The chamber was broken into 18 wire subsections (3 layers of 6 wires each) for processing by the network. The 18 drift times became inputs to a three layer neural network as shown in figure 29.

Input = 18 Sense Wire Drift Times  
 Output = 60 1.0cm Bins From -30cm to +30cm  
 + 1 Bin for  $Z < -30\text{cm}$  + 1 Bin for  $Z > +30\text{cm}$



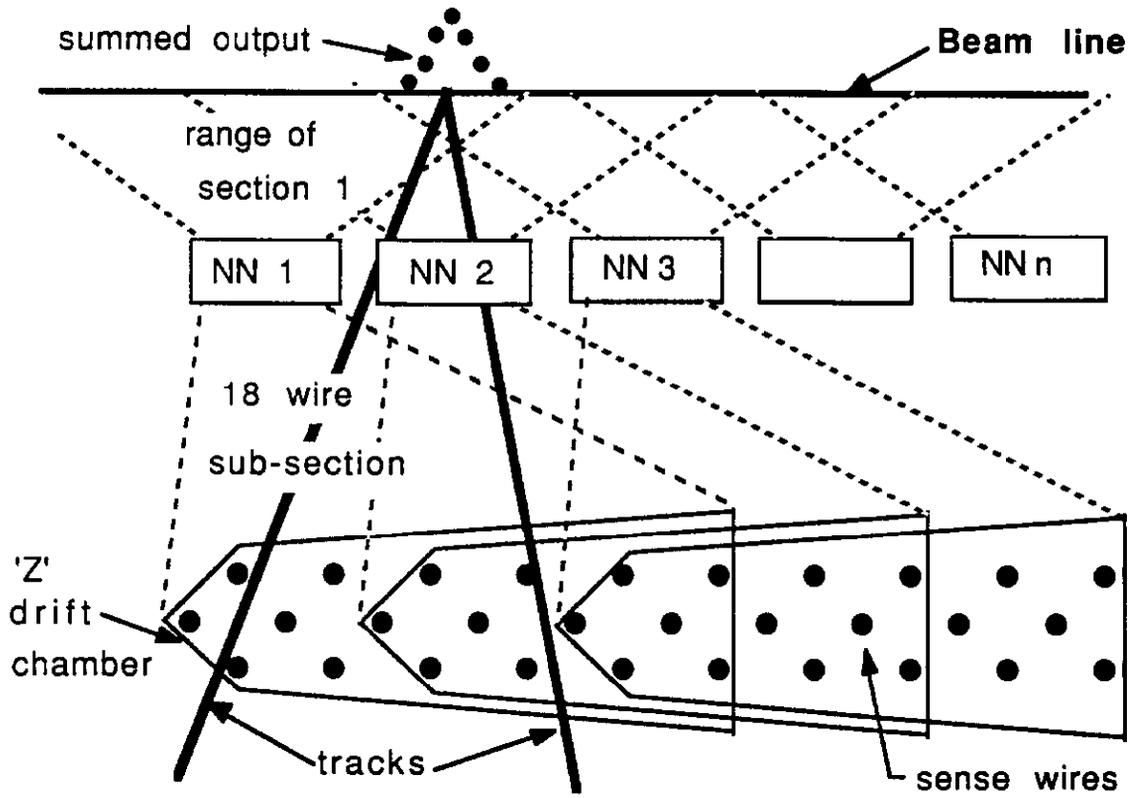


Figure 30.

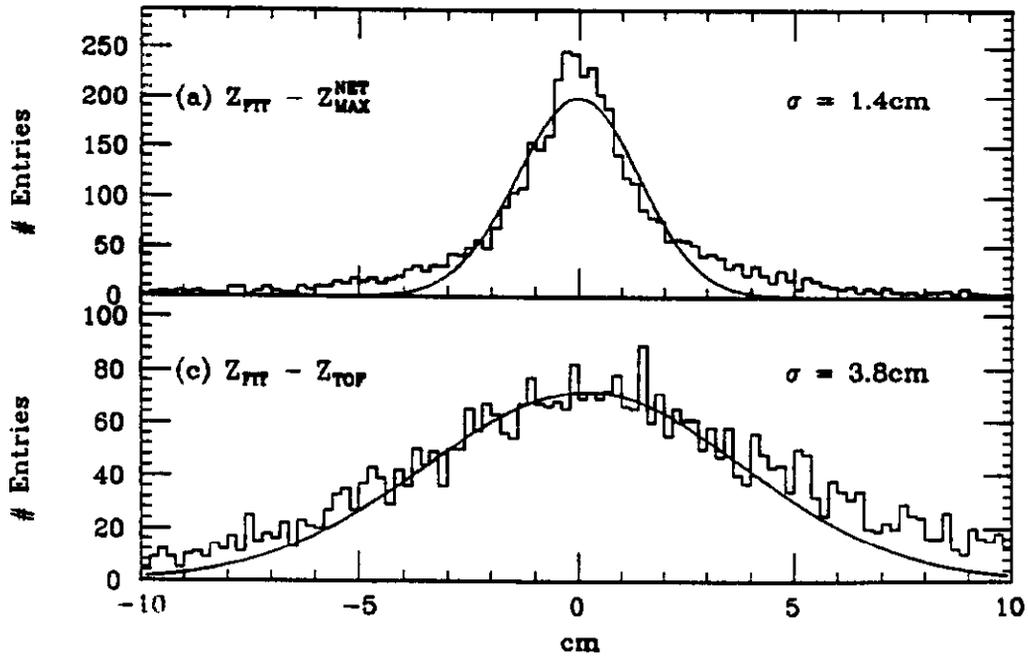


Figure 31.

The output layer had 62 units, 60 representing 1.0 centimeter bins from -30 cm to 30 cm. and 2 'overflow' units. The network was trained to represent the vertex position by a little

Gaussian histogram in the output units. This gives good vertex position resolution with relatively few output units. The 18 wire subsections were chosen so as to overlap in order not to miss tracks which may span subsections. The outputs of the subnets are then simply added. This is illustrated in figure 30. The resulting vertex distribution is shown in figure 28 for a typical event. The neural net output is shown as a dotted line. The maximum net output, represented by a cross, agrees well with the vertex position calculated offline as well as with the time-of-flight (TOF) vertex. Figure 31 compares the distribution of  $Z_{\text{offline}}-Z_{\text{NN}}$  to that of  $Z_{\text{offline}}-Z_{\text{TOF}}$ . TOF might possibly be implementable online. The neural net does much better, and can certainly be improved, while the TOF resolution can probably not be improved.

A similar technique has recently been applied, using a VLSI neural network, to online reconstruction of slope and intercept of muon tracks in a drift chamber<sup>29</sup>. The test beam setup is shown in figure 32. The setup is discussed in more detail in reference 30. Four more online reconstructed events are shown in figure 33 which superimposes the neural net result with that from the offline fit. Normally the net gets the correct answer. In those instances when it does not, it is usually in cases where there are two tracks with reasonable fits, as in the fourth frame of figure 33.

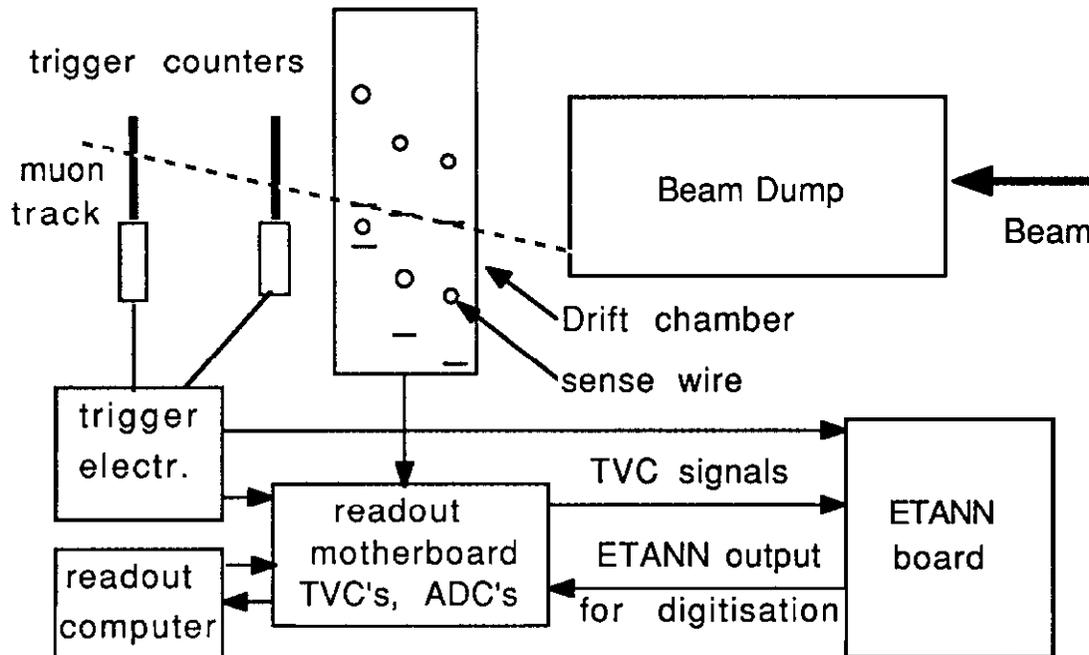
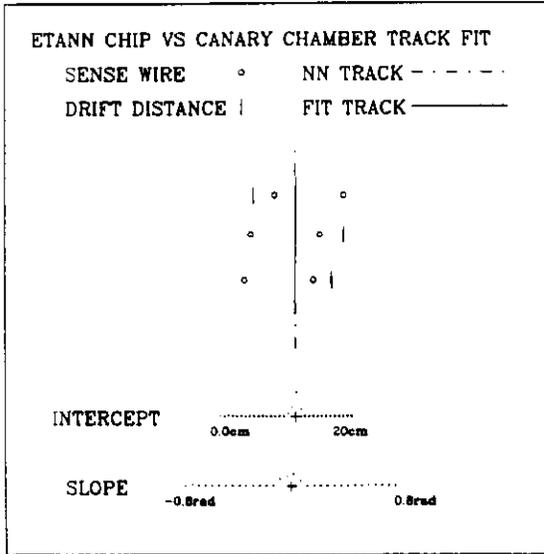


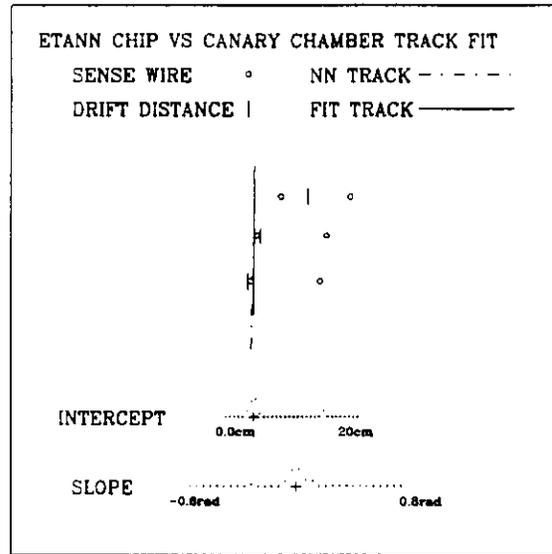
Figure 32.

### 3.2 Quark/Gluon Separation

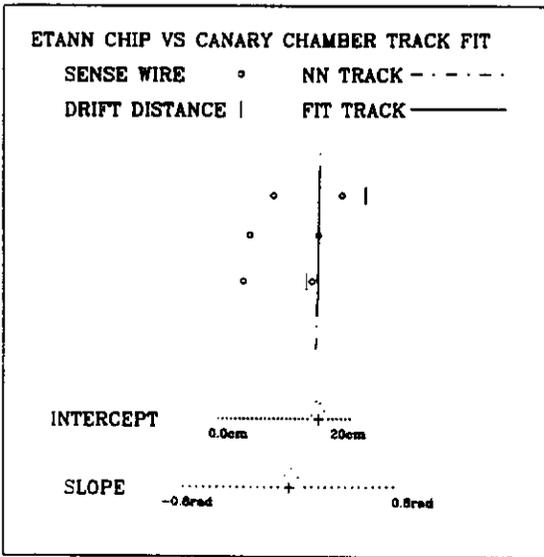
The identification of the parton ancestors of jets using neural networks has been treated in a number of references<sup>31,32,33,34</sup>. Most of these have treated electron-positron collider data and have used Monte Carlo data. Results are that 70 to 85% of the jets are correctly identified. Another Monte Carlo study for the proton-antiproton collider environment, with a crude detector simulation, got about 70% correct identification for quarks<sup>33</sup>. Reference 35 gives a result for proton antiproton collider with full detector simulation and also mentions the effect of the trained net on real data from the CDF experiment. The real data seems to contain components which resemble the quarks and gluons of the Monte Carlo.



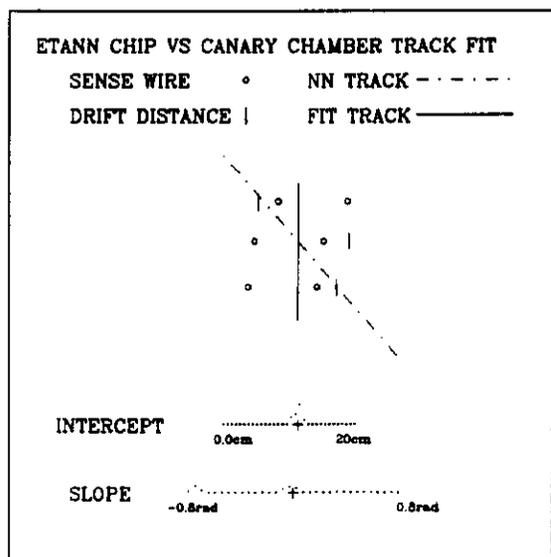
EVENT 2



EVENT 4



EVENT 9



EVENT 10

Figure 33.

### **3.3 Kink Recognition**

In this work<sup>36</sup> tracks in a TPC are examined to try to determine whether they are particles which have decayed and therefore contain a 'kink'. Two approaches are tried. In the first, the track is fit in an inner region, 1, and an outer region, 2. The track parameters in the two regions are used as input in to a neural network which tells whether or not this track is due to a decay. In the second approach, a single fit is done to the track, and the residuals of the fit are used as input to the neural network. Both of the neural methods are found to have higher efficiency than the standard chi-squared method, and are about a factor of 20 faster.

### **3.4 An Assortment of Backprop Approaches**

Numerous groups have used neural networks for tagging of B quarks. Typically this is done for electron positron colliders<sup>37,38,39,40,41</sup> although some work with jets at hadron colliders has also been reported<sup>15</sup>. A number of presentations on this subject will be found in the proceedings of this workshop<sup>42,43,44</sup>.

Other applications include identification of Cherenkov rings using the positions of the photons in the ring<sup>45,46</sup>, and determination of the quark jet charge using the z and charge of the leading particle in the jet.<sup>47</sup>

### **3.5 A Few New Approaches**

#### **3.5.1 Z Branching Ratio**

Reference 48 is a very nice result using real data to do the first high precision measurement of the Z branching ratio into the five known quark flavors. Event topology variables such as sphericities and invariant masses were the inputs to the neural net. This may be the first application of neural networks that has produced a new physics result.

#### **3.5.2 Resonance Search**

This work<sup>49</sup> comes from the E735 proton antiproton collider experiment at Fermilab. For two prong events, the inputs to a feed forward network are the three-momenta of the two particles, and the Z position of the vertex. The network is trained to tell 'signal', Monte Carlo generated rho, K, and Lambda particles which decay to two particles, from background simulated by same sign particle pairs. All of the particles are put through a detailed detector simulation. When a mass plot of opposite charge pairs is made using real data, clear signals for rho, K, and Lambda are seen (figure 34) when the events are selected by the neural network. When no neural network selection is made, no peak is seen. Apparently the network has learned a good combination of cuts to make to enhance the signal. No other attempt to define a set of cuts to enhance these signals in this experiment has been as successful.

#### **3.5.3 Mass Reconstruction**

This is another example of using a neural net to do function mapping. In this work<sup>50</sup>, proton antiproton collisions producing W particles which decay to two jets are produced. Gluon radiation, underlying event, and detector effects are included. Based upon the information in a calorimeter alone, a neural network is trained to calculate the mass of the final state system. Two approaches were tried: 1) the net is trained with all raw calorimeter energies; and 2) the net is trained to reconstruct the mass using a set of 'intelligent variables', such as the di- and tri-jet invariant masses. The intelligent variables approach is found to work much better. Results are shown in figure 35. The neural net method is found to be superior to the traditional method in which a fixed cone size is used to define the jets.

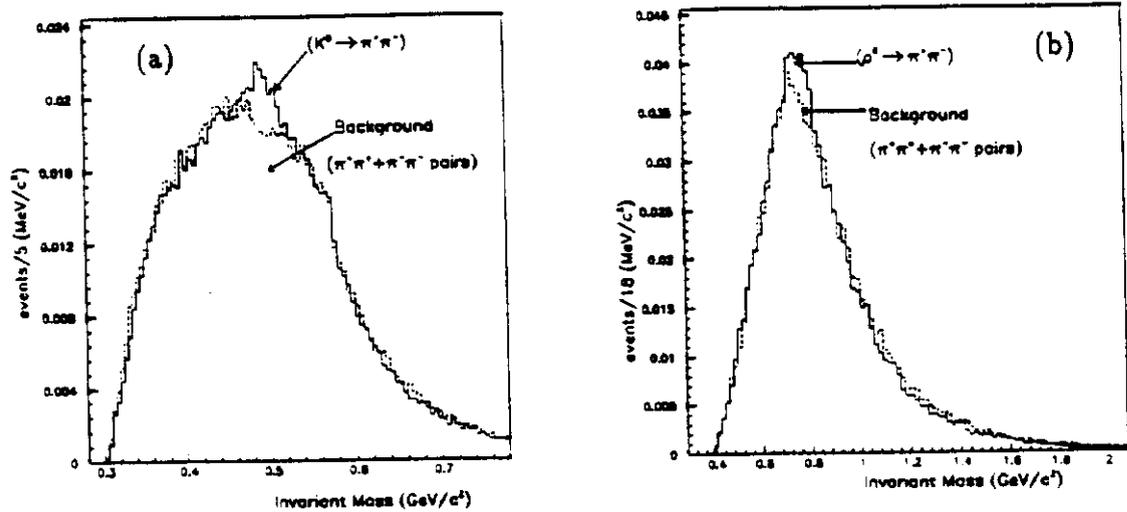
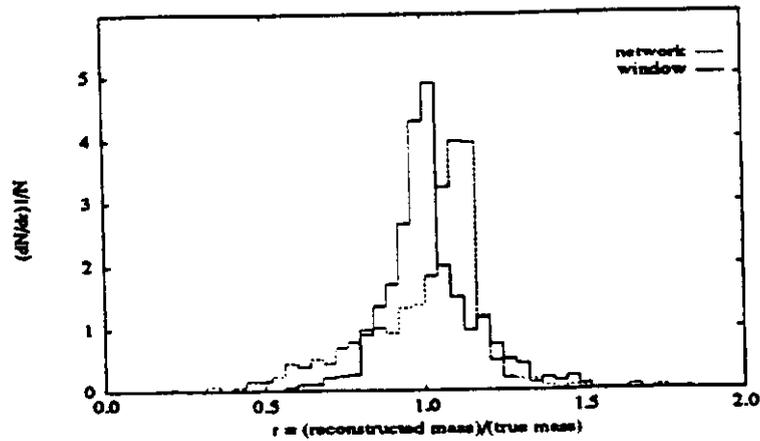


Figure 34.

a) Invariant mass distribution of  $\pi^+\pi^-$  pairs superposed with the corresponding invariant mass distribution of  $\pi^+\pi^+$  and  $\pi^-\pi^-$ . b) Invariant mass distribution of  $\pi^+\pi^-$  pairs superposed with the corresponding invariant mass distribution of  $\pi^+\pi^+$  and  $\pi^-\pi^-$ .



The reconstructed mass ( $M_{W,Z}$ ) divided by the true mass ( $M_{W,Z}^0$ ) using the neural network method (full line) and the conventional "window" method (dashed line) with  $R = 0.8$ .

Figure 35.

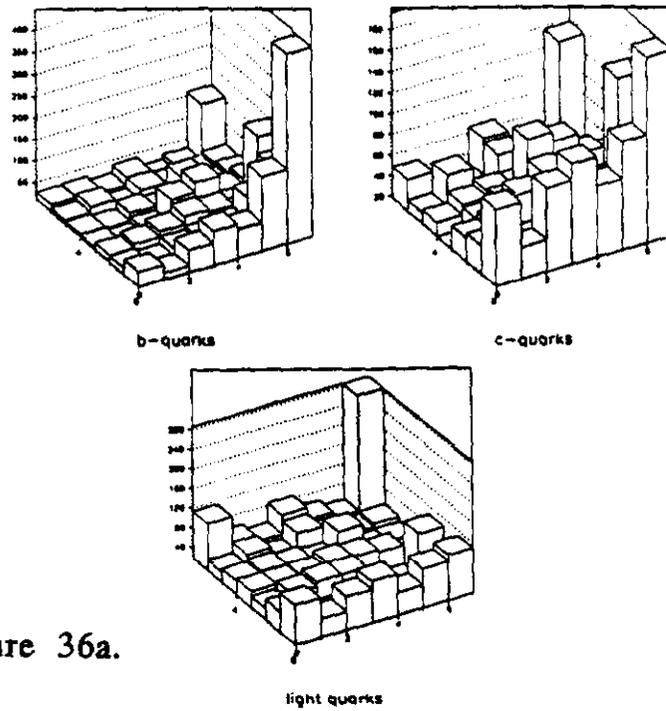


Figure 36a.

Distribution of b-quarks (a), c-quarks (b) and uds-quarks (c) over the self-organised 7 x 7 feature nodes.

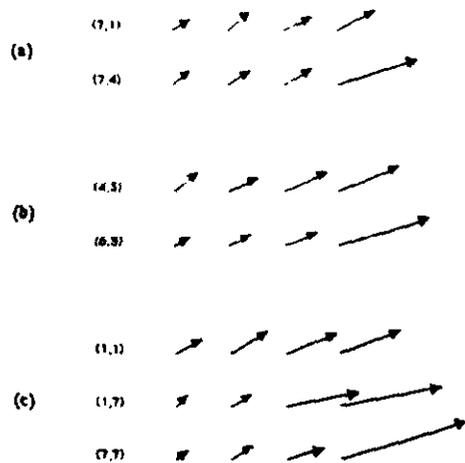


Figure 36b.

The weight vectors corresponding to the 4 leading hadrons for some of the units in the self-organising network; b-jets (a), c-jets (b) and uds-jets (c). The  $p_T$  component has been multiplied by a factor of 5 relative to the  $p_z$  component. Coordinates refer to the position in the map

### 3.5.4 Non-Backprop Applications

#### 3.5.4.1 Introduction

Recently some HEP applications of learning vector quantization (LVQ) and topological maps have appeared. The reader not familiar with LVQ and topological maps should refer to the tutorial of Fogelman<sup>51</sup> in the proceedings of this workshop. Learning vector quantization is a supervised learning algorithm like backpropagation and appears to give performance similar to that of backprop. It has been used in the quark/gluon separation of reference 34 and in the  $t\bar{t}$  recognition of reference 17. The topological map is an unsupervised learning algorithm and essentially performs a type of clustering.

#### 3.5.4.2 Jet Identification with a Topological Map

In this work<sup>52</sup>, a topological map with 49 units arranged in a 7 by 7 grid is used to identify quark flavor in jets. Eight input variables consisting of the the longitudinal and transverse momenta of the 4 leading particles in the jet were used as input to the topological map. The results obtained for jet classification were similar to those gotten with backpropagation, however, the topology of the solutions was found to give physical insight into the criteria the network used to classify the jets. Figure 36a shows the average response of the net to b, c, and light quarks. Certain groups of nodes are seen to be responsive to particular quark flavors. Figure 36b shows the weight vectors corresponding to the 4 leading hadrons for nodes which are specific to particular quark flavors. Nodes sensitive to uds quarks have weights which favor one or two leading particles which carry most of the momentum. Nodes for the heavier quarks favor a more uniform equipartition of momentum over the particles in the jet, in agreement with current notions about fragmentation.

## 4. Neural Network Hardware

### 4.1 Introduction

Most high energy physicists are familiar with discriminators. A neuron is essentially a low quality discriminator with a sluggish turn on function<sup>53</sup> (the sigmoid), see figure 37. Linear summing of signals is common in high energy physics: total energy, e.g. Triggers which weight calorimeter energies by the cosine of the angle of the cell are used in summed transverse energy triggers; thus weighted sums of energies are also common. A discriminator cutting on, say, summed  $E_t$ , is identical to a single layer perceptron, the simplest neural network, without hidden units. High energy physicists have been using neural networks for years without knowing it!

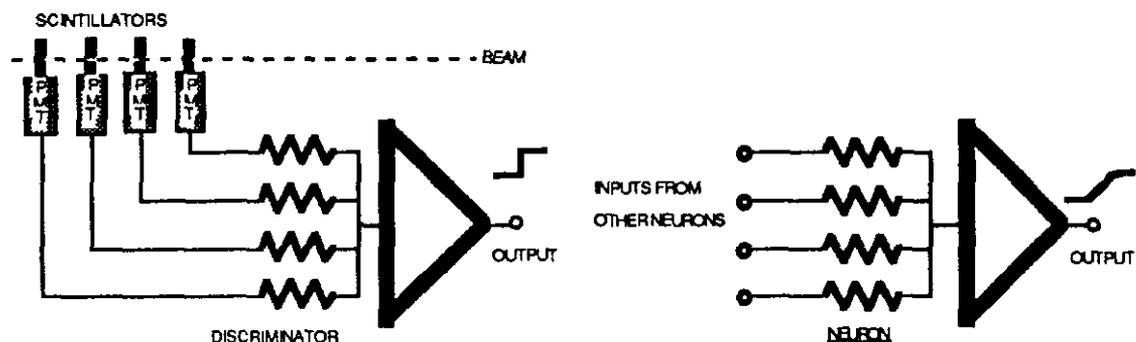


Figure 37.

What is new is the idea of using very dense neural networks to implement algorithms which are more complex than had been previously thought possible at the trigger level. Since a three layer neural network can implement any function, it can implement any trigger. Special

purpose hardware for triggers is not a new idea, but what is attractive about the neural networks is that they are reprogrammable without modifying the hardware.

## ***4.2 Current Hardware Status***

We limit ourselves here to commercially available products and a few products from private labs. We exclude 'crazy' things like acoustic charge transport in GaAs, which is very new, and optical implementations which are too slow at present.

### ***4.2.1 True Analog Approaches***

In these approaches, the neuron is built out of op-amps, synapses are 'resistors' or something which acts like them, the sum of products is done with Ohm's law and Thevenin's theorem. The synapses can be true resistors, multiplying DAC's, or floating gates. An example is the Intel ETANN chip<sup>54</sup> which has 64 neurons and 10240 floating gate synapses. The analog implementations are the fastest. Processing time for the ETANN is of order 1 microsecond per layer, independent of the number of inputs or hidden units. The analog chips are less precise than digital ones. The ETANN has the equivalent of only about 6 bits of precision.

Another analog chip which is even faster is the ANNA chip from Bell Labs<sup>55</sup>. It has 4096 synapses which can be configured as 16 neurons with 256 weights each, 256 neurons with 16 weights each, or anything in between. The configuration can be changed on each instruction. The I/O to the chip is completely digital but onboard processing is analog. The accuracy on the weights is 6 bits but the accuracy for the inputs and outputs is only 3 bits. Weights are stored on capacitors which are refreshed every 100 microseconds. Processing takes 200 nanoseconds. This chip is optimised for local interconnects and has been used in online optical character recognition applications.

### ***4.2.2 Digital Approaches***

The Adaptive Solutions company<sup>56</sup> offers a chip with the 'processor per neuron' approach. Each hidden unit is a small processor with local storage of all weights connecting to that unit (see fig 38). The inputs are presented sequentially and each hidden unit does a multiply and add to its locally stored sum of products. This type of chip does totally digital processing. The speed of the network depends linearly upon the number of inputs. The clock cycle of 40 nanoseconds is fast, but for a network of 100 inputs, the processing time will be 4 microseconds, slower than the ETANN.

Neural Semiconductor company<sup>57</sup> has commercialized the stochastic pulse train encoded synaptic weight multiply technique (figure 39). All weights and activations are stored digitally but the actual multiply is done with the rate multiplier as shown in figure 39. Basically the activation and the weight are input as pulse streams into an AND gate. The frequency of the output pulse stream is the product of the frequencies of the input pulse streams. In this case, processing is parallel but the accuracy depends on the desired precision in the weights. For 6 bit weights, 50 microseconds is required with a 40 ns clock cycle; thus again a digital technique is considerably slower than the analog one.

Oxford Computer Company<sup>58</sup> produces a digital neural network chip which is essentially a fast matrix multiply chip. It is basically a memory chip with 256 1 bit processors imbedded in it. The processors do the weight multiplication. The processing time depends upon the number of hidden units. Twenty microseconds is the estimate for a typical problem in high energy physics.

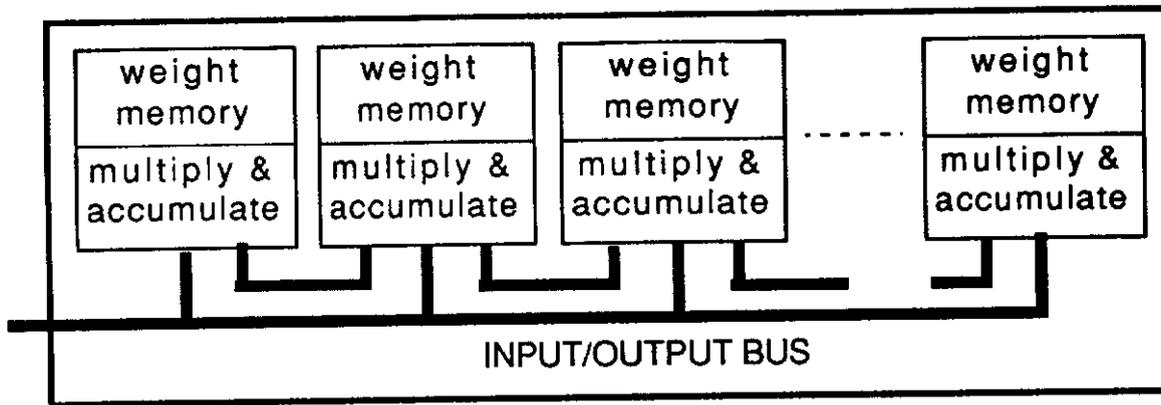


Figure 38. Processor per neuron approach

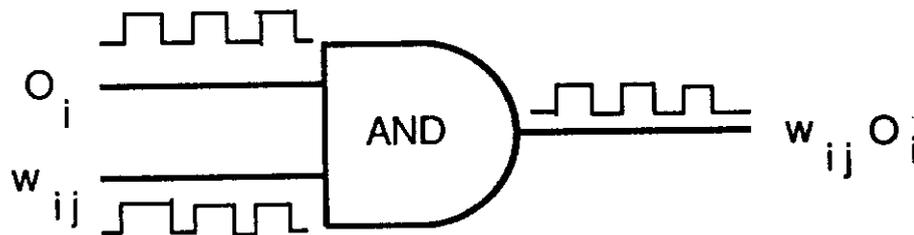


Figure 39. Stochastic pulse train encoded synaptic weight multiply

#### 4.3 Prognosis for Triggers

At Fermilab recently, a VLSI neural network has been used online in a test beam experiment to test a muon trigger which calculates slope and intercept of tracks in muon chambers in a few microseconds<sup>29</sup>. A neural network built of discrete components has been used to calculate the z position of muon tracks in the same chambers<sup>6</sup>. These are the first applications of hardware neural networks in high energy physics.

The D0 experiment at Fermilab hopes to follow up on the above test beam work and build a neural net muon trigger for the D0 muon upgrade in 1993/94<sup>59</sup>.

The CDF experiment at Fermilab is currently installing hardware for three neural net triggers, and results from these should be coming out in the upcoming months<sup>15,16</sup>.

A group at the H1 experiment at Hera is building a neural net trigger which will use 43 calorimeter energy sums to distinguish physics signals from beam gas, etc.<sup>60</sup> This experiment is just coming online.

#### 5. Acknowledgements

The author wishes to acknowledge the work of all those from whom he has borrowed in order to prepare this tutorial. Many of the figures have been taken directly from the work of others for demonstrative purposes. Thanks to F. Fogelman-Soulie, C.S. Lindsey, and to my CDF colleagues for helpful comments and suggestions.

## 6. References

1. D.E. Rumelhart, G.E. Hinton, and J.L. McClelland, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 1, MIT Press, Cambridge, Ma., 1986.
2. see, for example, R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*, John Wiley and Sons, New York, 1973; also,  
  
J.B. Hampshire and B. Pearlmutter, 'Equivalence Proofs for Mult-Layer Perceptron Classifiers and the Bayesian Discriminant Function', *Proceedings of the 1990 Connectionist Models Summer School*, Touretzky, Elman, Sejnowski, Hinton, Eds., San Mateo, California: Morgan Kaufman, 1990.
3. L. Garrido and V. Gaitan, *Use of Neural Nets to Measure the Tau Polarization and its Bayesian Interpretation*, UAB-LFAE-91-04, April, 1991, Universitat Autònoma de Barcelona preprint, submitted to *International Journal of Neural Systems*.
4. D. Conner, "Data Transformation Explains the Basics of Neural Networks", *EDN Magazine*, May 12, 1988.
5. L. Gupta, A. Upadhye, B. Denby, and S.R. Amendolia, "Neural Network Trigger Algorithms for Heavy Quark Event Selection in a Fixed Target High Energy Physics Experiment", Fermilab-Pub 91/117, submitted to *Pattern Recognition*.
6. Herman Haggerty, private communication.
7. B. Denby, *Computer Physics Communications*, 49 (1988) 429. Also, B. Denby "Neural Network and Cellular Automata Algorithms", Florida State University preprint FSU-SCRI-88-141, June, 1988. Tallahassee, Florida.
8. C. Peterson, *Nucl. Inst. & Meth.*, A279 (1989) 537.
9. G. Stimpfl-Abele and L. Garrido "Fast Track Finding with Neural Nets", UAB-LFAE 90-66, submitted to *Computer Physics Communications*, 1990.
10. B. Denby and S. Linn, *Computer Physics Communications* 56 (1990) 293.
11. C. Peterson, "Neural Networks and High Energy Physics", *Proc. of the International Workshop on Software Engineering, Artificial Intelligence and Expert Systems for High Energy and Nuclear Physics*, Lyon Villeurbanne, France, March, 1990, eds. D. Perret-Gallix and W. Wojcik, Editions du CNRS (Paris 1990).
12. C. Peterson and B. Soderberg, "A New Method of Mapping Optimization Problems onto Neural Networks", *International Journal of Neural Systems* 1 (1989) 3.
13. M. Gyulassy and M. Harlander, *Computer Physics Communications* 66 (1991) 31.
14. M. Ohlsson, C. Peterson, and A. Yuille, "Track Finding with Deformable Templates - The Elastic Arms Approach", Lund University Preprint LU TP 91-27, November 1991, Lund, Sweden, submitted to *Computer Physics Communications*.
15. B. Denby et al., *IEEE Trans. Nucl. Sci.* 37 No. 2 (1990) 248.

16. B. Badgett et al., CDF Internal Note 1310, "A Pattern Recognition Level-2 B Trigger at CDF in 1991", CDF Collaboration, Fermi National Accelerator Laboratory, Batavia, Illinois.
17. A. Cherubini and R. Odorico, "Identification by Neural Networks and Statistical Discrimination of New Physics Events at High Energy Colliders", in *Proc. of the workshop Neural Networks: From Biology to High Energy Physics*, Elba International Physics Center, Isola d'Elba, Italy, June 5-14, 1991, ETS Editrice, Pisa.
18. Mimetics, S.A., 5 Centrale Parc, Avenue Sully Prudhomme, 92298 Chatenay Malabry, France.
19. NeuralWare Inc., Sewickley, Pennsylvania.
20. California Scientific Software, Grass Valley, California.
21. NeuroDynamX Inc., Boulder, Colorado.
22. Intel Corporation, Santa Clara, California.
23. NT\_ROUTINES is the Fermilab simulator. Send mail to denby@fnal.
24. JETNET. Send mail to thepdr@seldc52 (University of Lund, Sweden).
25. B. Denby et al., CDF Internal Note 1538, "Proposal for a Level-2 Isolated Plug Electron Trigger for the 1991/1992 Run", CDF Collaboration, Fermi National Accelerator Laboratory, Batavia, Illinois.
26. B. Denby, E. Lessner, and C.S. Lindsey, *Proc. 1990 Conf. on Computing in High Energy Physics*, Santa Fe, NM, (1990) AIP Conf. Proc. 209 211.
27. C.S. Lindsey and B. Denby, *Nucl. Inst. & Meth.* A302 (1991) 217.
28. C. S. Lindsey, "Tracking and Vertex Finding in Drift Chambers with Neural Networks", in *Proc. of the workshop Neural Networks: From Biology to High Energy Physics*, Elba International Physics Center, Isola d'Elba, Italy, June 5-14, 1991, ETS Editrice, Pisa.
29. C.S. Lindsey, B. Denby, H. Haggerty, K. Johns, "Real Time Track Finding in a Drift Chamber with a VLSI Neural Network", Fermilab-Pub-92/55, accepted for publication in *Nucl. Inst. & Meth. A*.
30. B. Denby "Status of Neural Net Triggers at Fermilab Tevatron", to appear in the *Proceedings of the Second International Workshop on Software Engineering, Artificial Intelligence, and Expert Systems for Nuclear and High Energy Physics*, La Londe les Maures, France, January 1992, World Scientific.
31. L. Lonnblad, C. Peterson, and T. Rognvaldsson, *Phys. Rev. Letters* 65 (1990) 1321.
32. L. Lonnblad, C. Peterson, and T. Rognvaldsson, *Nucl. Physics* B349 (1991) 675.

33. P. Bhat, L. Lonnblad, K. Meier, K. Sugano, "Using Neural Networks to Identify Jets in Hadron Hadron Collisions", *Proc. of the 1990 Summer Study on High Energy Physics - Research Directions for the Decade*, Snowmass, Colorado, June 25 - July 13, 1990.
34. I. Csabai, F. Czako, Z. Fodor, "Combined Neural Network-QCD Classifier for Quark and Gluon Jet Separation", CERN Preprint CERN-TH.6038/91 and Eotvos University (Budapest) Institute for Theoretical Physics preprint ITP-Rep. Budapest 483, March, 1991.
35. S. Bianchin, M. Denardi, B. Denby, M. Dickson, G. Pauletta, L. Santi, and N. Wainer, "Classification of Jets from PPbar Collisions at Tevatron Energies", to appear in the *Proceedings of the Second International Workshop on Software Engineering, Artificial Intelligence, and Expert Systems for Nuclear and High Energy Physics*, La Londe les Maures, France, January 1992, World Scientific., and CDF Internal Note 1706.
36. G. Stimpfl-Abele and Lluís Garrido, "Recognition of Decays of Charged Tracks with Neural Network Techniques", Université Blaise Pascal preprint, Clermont-Ferrand, France, submitted to *Computer Physics Communications*, May 1991.
37. J. Proriot et al., "Tagging B Quark Events in Aleph with Neural Networks", in *Proc. of the workshop Neural Networks: From Biology to High Energy Physics*, Elba International Physics Center, Isola d'Elba, Italy, June 5-14, 1991, ETS Editrice, Pisa.
38. C. Bortolotto et al., "A Measurement of the Partial Hadronic Widths of the  $Z^0$  Using Neural Networks", in *Proc. of the workshop Neural Networks: From Biology to High Energy Physics*, Elba International Physics Center, Isola d'Elba, Italy, June 5-14, 1991, ETS Editrice, Pisa.
39. N. De Groot and M. Los, "B-Tagging in Delphi with a Feed-Forward Neural Network", in *Proc. of the workshop Neural Networks: From Biology to High Energy Physics*, Elba International Physics Center, Isola d'Elba, Italy, June 5-14, 1991, ETS Editrice, Pisa.
40. T. D. Gottschalk and R. Nolty, "Identification of Physics Processes Using Neural Network Classifiers", Caltech Report CALT-68-1680, 1991.
41. L. Bellantoni, J.S. Conway, J.E. Jacobsen, Y.B. Pan, Sau Lan Wu, "Using Neural Networks with Jet Shapes to Identify b Jets in  $e^+e^-$  Interactions", CERN-PPE/91-80, 24 May 1991, submitted to *Nucl. Inst. & Meth.*
42. F. Seidel et al., "Extensive Studies on a Neural Networks for b Tagging and comparisons with a Classical Method", to appear in the *Proceedings of the Second International Workshop on Software Engineering, Artificial Intelligence, and Expert Systems for Nuclear and High Energy Physics*, La Londe les Maures, France, January 1992, World Scientific.
43. P. Branchini, M. Ciuchini, and P. Del Giudice, "B Tagging with Neural Networks: An Alternative use of Single Particle Information for Discriminating Jet Events", to appear in the *Proceedings of the Second International Workshop on Software Engineering, Artificial Intelligence, and Expert Systems for Nuclear and High Energy Physics*, La Londe les Maures, France, January 1992, World Scientific, and INFN-ISS 92/1.
44. B. Brandl et al., "Tagging of Z Decays into Heavy Quarks in the Aleph Detector using Multivariate Analysis Methods: Neural Networks, Discriminant Analysis, Clustering", to

appear in the *Proceedings of the Second International Workshop on Software Engineering, Artificial Intelligence, and Expert Systems for Nuclear and High Energy Physics*, La Londe les Maures, France, January 1992, World Scientific.

45. T. Altherr et al., "Cerenkov Ring Recognition using Adaptable and non-Adaptable Networks", to appear in the *Proceedings of the Second International Workshop on Software Engineering, Artificial Intelligence, and Expert Systems for Nuclear and High Energy Physics*, La Londe les Maures, France, January 1992, World Scientific.

46. N. de Groot et al., "Pion/Kaon Separation with Neural Networks", to appear in the *Proceedings of the Second International Workshop on Software Engineering, Artificial Intelligence, and Expert Systems for Nuclear and High Energy Physics*, La Londe les Maures, France, January 1992, World Scientific.

47. P. Silva and J. Varela, "Identification of the Quark Jet Charge Using Neural Networks", in *Proc. of the workshop Neural Networks: From Biology to High Energy Physics*, Elba International Physics Center, Isola d'Elba, Italy, June 5-14, 1991, ETS Editrice, Pisa.

48. C. Bortolotto et al., "A Measurement of the Partial Hadronic Widths of the  $Z^0$  using Neural Networks", in *Proc. of the workshop Neural Networks: From Biology to High Energy Physics*, Elba International Physics Center, Isola d'Elba, Italy, June 5-14, 1991, ETS Editrice, Pisa.

49. T. Alexopoulos, "Resonance Searches using a Neural Network Technique", talk at DPF 91, Vancouver, Canada, August 1991, submitted to proceedings, also T. Alexopoulos, Ph.D. Thesis, University of Wisconsin, unpublished.

50. L. Lonnblad, C. Peterson, and T. Rognvaldsson, "Mass Reconstruction with a Neural Network", Lund University preprint LU TP 91-25, October 1991, submitted to *Physics Letters B*.

51. F. Fogelman Soulie, "Neural Networks for Pattern Recognition", to appear in the *Proceedings of the Second International Workshop on Software Engineering, Artificial Intelligence, and Expert Systems for Nuclear and High Energy Physics*, La Londe les Maures, France, January 1992, World Scientific.

52. L. Lonnblad, C. Peterson, H. Pi, and T. Rognvaldsson, "Self Organizing Networks for Extracting Jet Features", Lund University preprint LU TP 91-4, March 1991, submitted to *Computer Physics Communications*.

53. B. Denby, "Neural Network Tutorial for High Energy Physicists", *Proc. of the International Workshop on Software Engineering, Artificial Intelligence and Expert Systems for High Energy and Nuclear Physics*, Lyon Villeurbanne, France, March, 1990, eds. D. Perret-Gallix and W. Wojcik, Editions du CNRS (Paris 1990).

54. Intel 80170NX Electrically Trainable Analog Neural Network, Intel Corporation, Santa Clara, California.

55. B. Boser, E. Sackinger, ATT Holmdel, "An Analog Neural Network Processor with Programmable Network Topology", International Solid State Circuits Conference, February 14, 1991, paper TPM 11.3.

56. Adaptive Solutions Inc., Beaverton, Oregon.

57. Neural Semiconductor Inc., Carlsbad, California.
58. Oxford Computer Company, Oxford, Connecticut.
59. M. Fortner, "Analog Neural Networks in an Upgraded Muon Trigger for the D0 Detector", to appear in the *Proceedings of the Second International Workshop on Software Engineering, Artificial Intelligence, and Expert Systems for Nuclear and High Energy Physics*, La Londe les Maures, France, January 1992, World Scientific.
60. P. Ribarics et al., "Neural Network Trigger in the H1 Experiment", in *Proc. of the workshop Neural Networks: From Biology to High Energy Physics*, Elba International Physics Center, Isola d'Elba, Italy, June 5-14, 1991, ETS Editrice, Pisa.