Gamma/Hadron Separation with Neural Networks

B. Osborn[1], D. Wyatt[2]
(1) University of Maryland, College Park, MD 20742, USA
(2) University of California, Santa Cruz, CA 95064, USA

1. Introduction

The sensitivity of the Milagro detector depends on the ability to distinguish particle showers caused by γ-rays from the background of hadronic showers. This γ/hadron separation can be quantified by a quality factor, $Q = \epsilon_\gamma / \sqrt{1 - \epsilon_h}$, where ϵ_γ is the efficiency of detecting a γ and ϵ_h is the efficiency of detecting a hadron. The signal-to-noise ratio of the detector is proportional to Q. Monte Carlo simulations of both types of showers indicate several identifying characteristics. γ showers are prevalently electromagnetic while hadron showers generate a larger number of muons. Hadron showers also contain secondary hadrons which create smaller sub-showers. At the ground level, these sub-showers appear as inhomogeneities in the distribution of particles. In a γ shower, the particles are distributed more uniformly around a strong central core.

Inspection of data generated by the Monte Carlo simulation indicates that the inherent differences between showers will be reflected in the data from the detector. Both γ and hadron showers display patterns in the number of tubes hit and the sum of the photo-electron signal on the upper and lower layers. The spatial distribution of the signal over the pond may also be useful for separation. Possible methods of γ/hadron separation include elimination of events which create a large peak signal on the bottom layer of photo-multiplier tubes [1] (indicating a muon) or using a wavelet analysis technique to quantify the inhomogeneity of the showers [2].

These and other indicators can be further analyzed by combining their results into a set of numbers which characterizes each shower and processing these with a neural network program. Being nonlinear extensions of conventional methods, neural networks have several advantages, including their simplicity and (often) superior performance. This memo investigates the effectiveness of the neural network approach for background suppression in Milagro.

2. The Neural Network

This chapter provides a short general description of the neural network method. For a more detailed description, see *e.g.* [3].

From a mathematical perspective, a neural network is simply a mapping from $R^n \to R^m$. In the case of the Milagro data, the net maps an input vector comprised of γ/hadron indicators onto the interval [0,1]. The mapping can be visually represented as layers of processing units, called nodes, which are connected by adaptive weights. Fig. 1 shows a neural network with three layers. The first layer is called the input layer and holds the information which will be processed by the network. The second

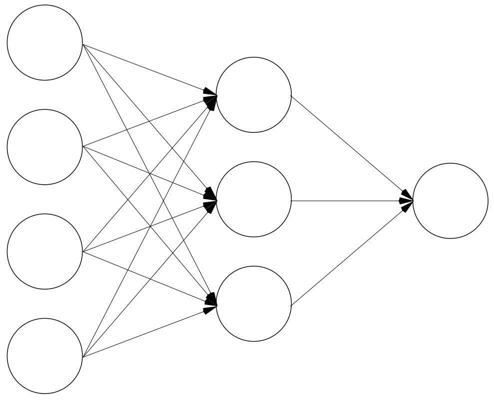

Figure 1. Diagram of a neural network with one hidden layer

layer is generally referred to as a hidden layer since its output is not observed. The third layer is the output layer which returns the vector generated by the mapping.

Each node on the first layer is assigned the value of the corresponding component of the input vector, $\vec{x}$. Calculating the scalar product of the input vector and the vector $\vec{w}$ whose components are the weights between the nodes of the first layer and the j^{th} node of the second layer gives an activation value, a_j, for the j^{th} node of the second layer. This begins the forward propagation of information through the network. The general formula for the activation value of any node in the network is given by the equation

$$a_j = \sum_{i=1}^{n} w_{ij} z_i \tag{1}$$

where z_i is the output value of a node from the previous layer connected by a weight, w_{ij} This value is then acted on by a transfer function, g, and the quantity $z_j = g(a_j)$ is the output value of the node. This process is completed for each node in the layer and each layer in the network to obtain a vector $\vec{o}$ whose components are the output values of the nodes in the last layer. The function represented in Fig. 1 can be written explicitly as

$$o_k = g_o \left(\sum_{j=0}^{3} w_{jk} g_h \left(\sum_{i=0}^{4} w_{ij} x_i \right) \right) \tag{2}$$

where g_o and g_h are the transfer functions for the output and hidden layers. The process can be generalized for network structures with arbitrary numbers of hidden layers and nodes in each layer.

To generate a network which can be used for pattern recognition, we choose a set of input vectors whose target output, $\vec{t}$, is known. In the absence of a set of γ-showers from the experimental data, we have to use Monte Carlo generated proton and γ showers. Their output should be 0 and 1, respectively. The weights of the network begin as random values, and the output of this random network is compared to the desired output to calculate a sum of squares error,

$$E = 1/2 \sum_{i=0}^{m} (o_i - t_i)^2 \tag{3}$$

which is a differentiable function of the network weights. Derivatives of the error function are used to minimize error with respect to the weights.

Our program minimizes E by combining the following methods of back-propagation (calculating the error propagation backwards through the network) and gradient descent. An input vector representing a Monte Carlo event is run through the network. From Eq. 1, we calculate

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial a_j} \frac{\partial a_j}{\partial w_{ij}} \tag{4}$$

Defining the error $\delta_j \equiv \partial E / \partial a_j$ and using Eq. 3 to calculate $\partial a_j / \partial w_{ij}$ gives

$$\frac{\partial E}{\partial w_{ij}} = \delta_j z_i \tag{5}$$

The error on a weight in the output layer, δ_k, is

$$\delta_k = g'(a_k) \frac{\partial E}{\partial o_k} \tag{6}$$

We use the chain rule for partial derivatives to calculate the error δ_j on weights in the hidden layers.

$$\delta_j = \sum_k \frac{\partial E}{\partial a_k} \frac{\partial a_k}{\partial_j} \tag{7}$$

Evaluating the derivatives using Eq. 1 gives

$$\delta_j = g'(a_j) \sum_k w_{jk} \delta_k \tag{8}$$

where the sum runs over all of the connections between the j^{th} node and the k nodes in the next layer. Since the error δ_k on the nodes in the output layer is known, Eq. 8 can be applied to each previous level of the network.

The gradient descent technique uses the results of the back-propagation algorithm to adjust each of the weights in a direction which will decrease the total error, E, of the network. Each weight is changed by an amount

$$\Delta w_{ij} = \eta \delta_j z_i \tag{9}$$

where η, the learning rate, is a user controlled parameter which scales the adjustment. We use the method of on-line learning in which a Δw is applied after every pattern in the data sample.

Our program also uses a momentum term α, $0 \leq \alpha \leq 1$, which adds a fraction of Δw from the previous pattern to prevent the net from settling in a flat spot of the error landscape.

Typically, the learning rate is not a constant, but is multiplied by a scaling factor < 1 after each epoch. Decreasing the learning rate during training allows the net to settle into a stable state.

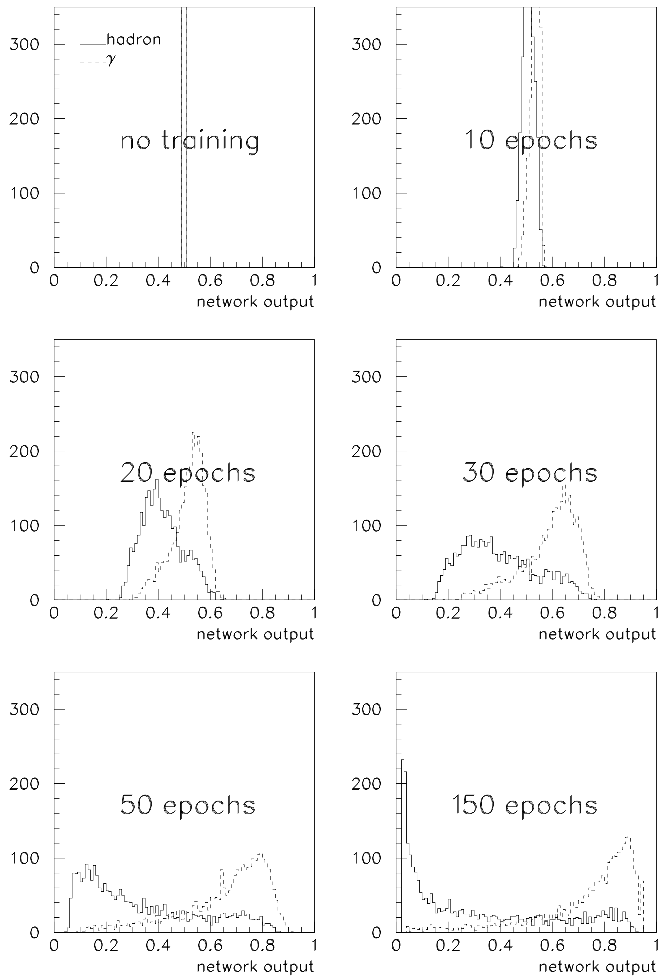

Figure 2. Network Training: This sequence shows the output of a network on a test set of γ and hadron showers at several intervals in the training process. The learning rate for this net is artificially low to visualize the learning: with a "reasonable" learning rate, as few as 10 epochs are sufficient to achieve a good separation.

We refer to the combination of back-propagation and gradient descent as training the network. Running the network through all of the patterns in a data sample is called a training epoch. Fig. 2 shows the evolution of a network over 150 epochs.

After successive training epochs, the net begins to return the proper output for each type of shower. By the last frame, the network has converged on the best weight values for the training sample it processed.

3. Choosing an Input Vector

The performance of a network is improved by finding the best combination of γ/hadron indicators for training. We have experimented with the following parameters which all, either alone or in combination, show some separation power:

- Total number of PMTs hit (upper layer) (1)
- Total number of PMTs hit (lower layer) (2)
- $\sum$ photo-electron signal (upper) (3)
- $\sum$ photo-electron signal (lower) (4)
- Maximum PMT signal on the lower layer (5)
- t_χ of hottest lower layer PMT (after angle fit) (6)
- $\sum$ photo-electron signal of nearest neighbors of hottest upper layer tube (7)
- $\sum$ photo-electron signal of nearest neighbors of hottest lower layer tube (8)
- Maximum t_χ and corresponding photo-electron signal in upper layer (9,10)
- Maximum t_χ and corresponding photo-electron signal in lower layer (11,12)
- Distance between two hottest lower layer tubes (13)

To guarantee a stable network training process, these values have to be preprocessed, as they all cover a rather different range of possible values. The number of PMTs has been normalized to the maximum number of PMTs in the upper and lower layer, respectively, and we use the logarithm to the base 10 of the photo-electron signals rather than the actual value.

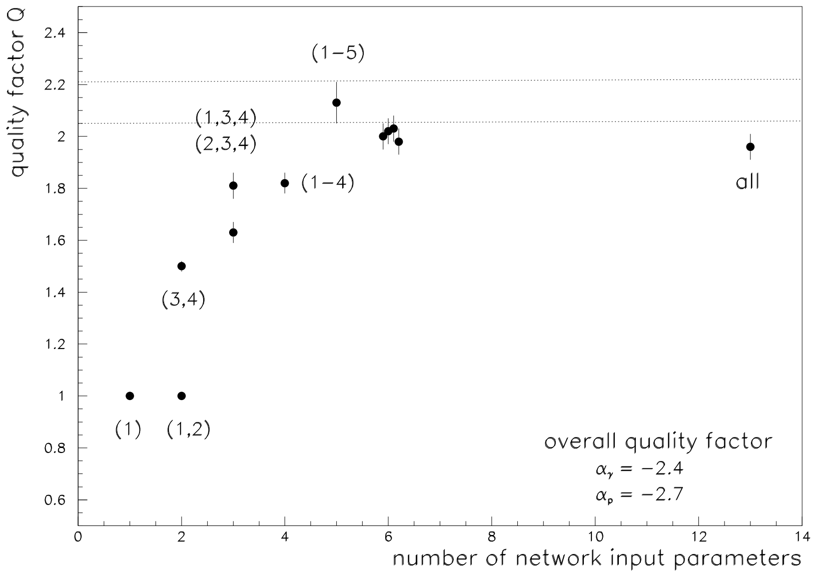

Figure 3. Comparing Networks: Quality factor for networks using different combinations of input parameters. The parameters used are indicated by the numbers, which correspond to the numbers assigned in the text. The networks with 6 input nodes represent combinations of parameters (1-5) with one additional parameter from (6-13).

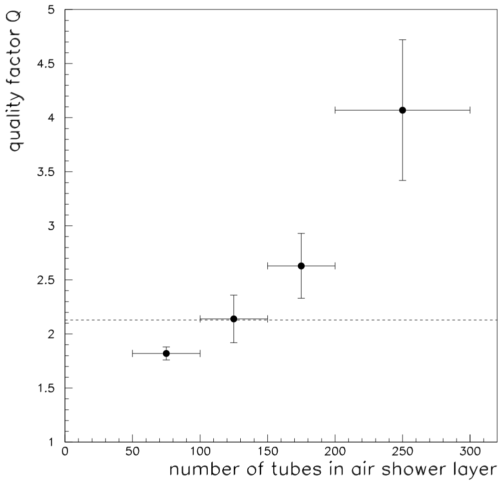

Figure 4. Quality factor versus number of PMTs for the network with the 5 basic parameters. Of the MC γ-showers, 54% have more than 200 tubes, 19% have between 150 and 200 tubes, 9% have between 100 and 150 tubes, and 18% have between 50 and 100 tubes.

For the training and testing of the network, we use MC γ and proton showers with a zenith angle θ between $0°$ and $45°$ and a primary energy between 100 GeV and 100 TeV. The spectral index is 2.4 for γ and 2.7 for proton showers. The core throw area is $300 \times 300\,\mathrm{m}^2$, and a minimum of 50 air shower tubes per event is required. Note that the background is represented by proton showers only. Typically, γ/proton separation is the most difficult task, and heavier nuclei can be rejected with higher efficiency than proton showers. This should be checked once a sufficient amount of showers initiated by heavier nuclei is available.

network topology	5–5–1
(initial) learning rate	0.01
learning rate decay factor	0.999
momentum term	0.1
number of training epochs	500
transfer function	$1/(1+e^{-x})$
number of γ training showers	4000
number of proton training showers	4000
spectral index γ/proton	2.4/2.7

Table 1. Network Parameters for the optimal net.

After adequate training on the events of the training data set, the network is tested by running on an independent test data set. To have a method for comparing different nets, we calculate Q as a function of the number of PMTs hit in an event in the test set, using the hadron cut which maximizes Q. One of the advantages of the neural

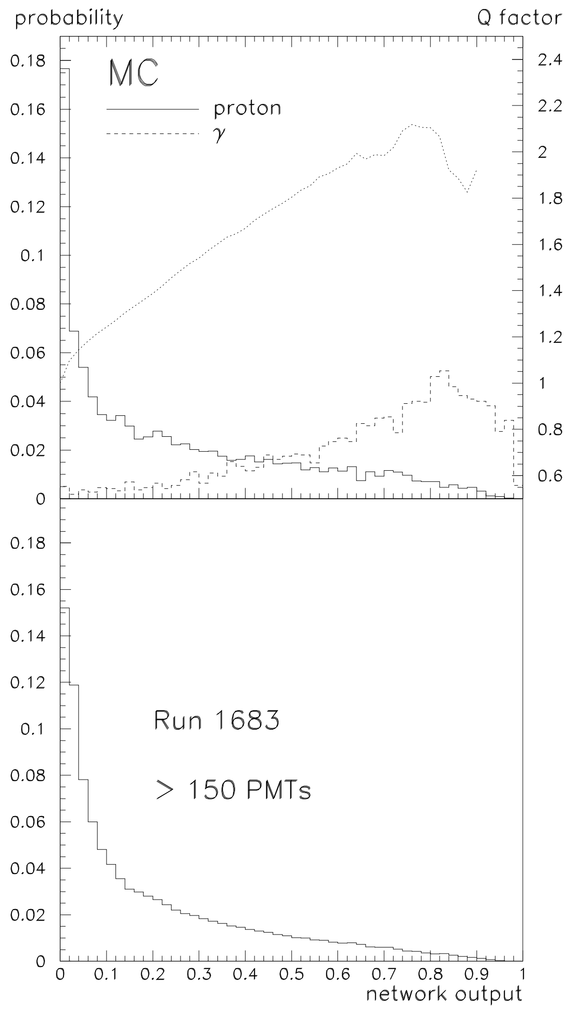

Figure 5. Network output for MC showers (top) and experimental data (bottom). The line graph is of the Q factor vs hadron cutoff.

network is that is allows for a qualitative evaluation of potential separation factors. Fig. 3 is a comparison of different networks which represent combinations of inputs from the list.

According to the network performance, the basic information from an event is sufficient and necessary for separation of most events. A net that only uses 5 parameters, the (normalized) number of PMTs hit in upper and lower layer, the sum of photo-electrons in upper and lower layer, and the number of photo-electrons in the hottest lower layer tube gives the best results. Fig. 4 shows the quality factor as a function of the number of upper layer tubes in the event, and Fig. 5 shows the network output for the MC test data (top) and experimental data (bottom).

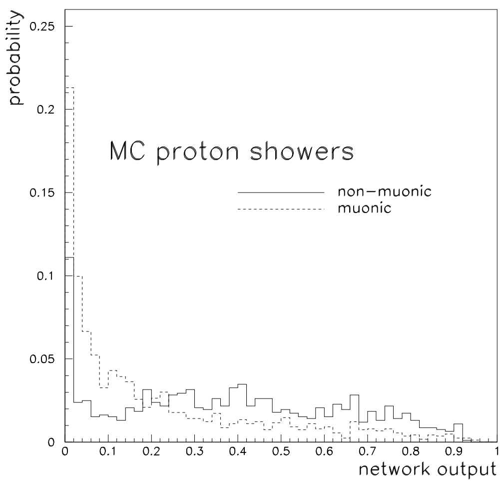

Figure 6. Network output for MC proton showers with and without a muonic component.

The performance of the network is relatively independent of the choice of parameters like the network learning rate and the network topology (*i.e.* number of hidden nodes). However, a sensible choice of these parameters usually stabilizes the learning process and minimizes the difference between the network performance on training and test data. Table 1 shows the parameters used for the network.

4. Non-muonic showers

The five basic parameters may not, however, characterize all showers. Further analysis of the network shows that hadron events which have a network output value closer to zero are those events that contain muons (Fig. 6). For example, the best network above has $\epsilon_p = 0.95$ for proton showers with a muonic component. However, $\epsilon_p = 0.82$ when the network is tested on a data sample of proton showers without muons.

These results indicate that the basic γ/hadron indicators tend to favor muonic showers. To significantly improve Q beyond ~ 2, we would like to develop a network which is trained to distinguish between γ showers and proton showers without muons. Visual inspection of these events indicates that they may be separated by using information from the upper layer, but development of an appropriate network is currently limited by a relatively small data set of non-muonic showers.

5. Effects of Core Location

Another notable feature of the network is its facility in separating events whose core falls off the pond. Fig. 7 shows histograms of network output for events with their core on and off the pond.

For events with their core off the pond, the network achieves a significantly better Q. As the second histogram shows, the γ showers are no longer nicely stacked close to 1. This effect may be caused by the misidentification of the core as a muon.

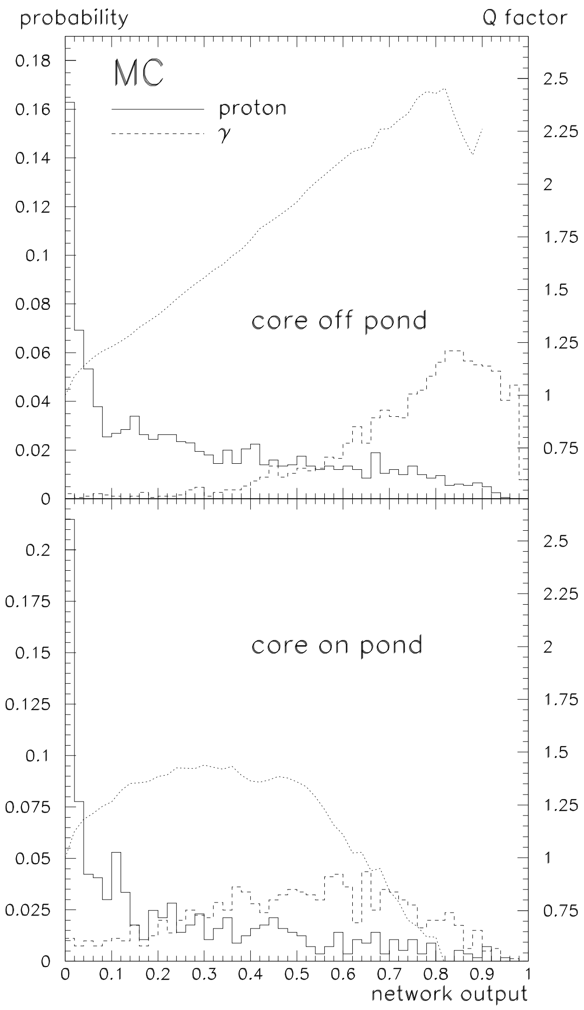

Figure 7. Showers with core on and off pond: The top graph shows a histogram of the network output for events with their core off the pond. The bottom graph shows those with the core on the pond. The line graphs show the Q factor vs hadron cutoff.

This feature of the network suggests that locating the core may be an integral part of γ/hadron separation. Adding the location of the core to the network input vector would help distinguish between ambiguous muon/core showers.

6. Using "Raw" Data

An alternate approach to choosing separation indicators is to use unprocessed information from the pond to characterize an event. Given a large enough training set, the photo-electron signal of each tube would be a component of the input vector. Within the limitations of our available data, we approximated this by dividing each layer of the pond with a grid and using the sum of the signal in each section as an input.

Since the Q achieved with this method was significantly lower than that of other networks, we decided that the method was not worth pursuing as a primary separation technique. However, it may prove to be useful for finding non-muonic proton showers if a data sample of these type of events is generated which is large enough to create a generalized network. Currently, a network with high enough resolution to find spatial patterns would also have enough input nodes to allow a perfect fitting of the network mapping to each training shower input vector. The result is a net which makes a complete separation of a training set, but no separation of a test set.

7. Future Work

This initial investigation into a network separation indicates that the method is worth pursuing. The best network to date approximately doubles the signal to noise ratio, and we expect this performance to improve with some continued effort.

The network separation is based on the study of simulated data. The ultimate test is the effect it has on a signal from a well-understood source, *i.e.* the Crab. This study is under way. It is very likely that severe adjustments have to be made to the simulation to better reflect our data, and the network training has to be redone with the improved simulation. Once a separation has been established, the network could be refined to better separate the showers whose current net output is ambiguous. It is apparent that the network is very good at separating showers containing muons from γ showers whose core falls off the pond. Inputs which characterize non-muonic proton showers are needed to move beyond Q of 2. Some possibilities include quantifying the spatial distribution of either particles or wavelet clusters, and geometrically correlating information from the upper and lower layers. This information, as well as the location of the core should improve the muon/core separation.

8. Using the Program

The programs used for this study are available. They are located in the directory `/usr/vhep/milagro/network/` on milagro.ucsc.edu (accessible on the web as http://scipp.ucsc.edu/milagro/network/). To create the executable `neural`, use the `makefile` for Linux PCs. The relevant routines are `neural.c`, `network.c`, `diag.c`, and `io.c`, some structure definitions are in `structures.inc`. To use the program for any separation problem with two classes of events A and B, where the network is to be trained to give an output of "1" for class A events and "0" for class B events, create the following files:

- a training data file with events of class A
- a training data file with events of class B
- a test data file with events of class A
- a test data file with events of class B (these test files should contain independent events that are not also used for training).

The training and test data files should be simple text files with the input data arranged in rows i and columns j, with a new row for each event and j_{max} columns for the j_{max} input parameters of the network.

Before running `neural`, adjust the input file `param` containing user-defined network parameters:

- the number of layers and the topology (number of nodes per layer, starting with the input layer
- the columns of the training and test files that should be used as network input (this is useful if not all parameters in the data files are to be used in the separation or if the user wants to test which parameters out of a variety of available parameters actually allow separation)
- the file names for the training and test data
  - training data set class A (network output 1)
  - training data set class B (network output 0)
  - test data set class A
  - test data set class B
- the learning rate η
- the learning decay factor (the learning rate is multiplied by this factor after each learning epoch)
- the range r for the initial weights before training (weights are taken from the interval $[-r, r]$)
- the momentum term α, $0 \leq \alpha \leq 1$
- the number of epochs to run
- the transfer functions connecting the layers of the network.

The range r for the initial set of the weights is a crucial parameter. Before the network training starts, the net has to be initialized, which is done by assigning random values from the interval $[-r, r]$ to the weights. If a sigmoid transfer function is used, the learning process should start in the linear part of the transfer function. If these initial values are too large, the network will already be saturated. The start interval therefore depends on the number of nodes and should be $\mathcal{O}(1/N_{nodes})$ or less, where N_{nodes} is the number of nodes.

For an analysis of the resulting network and its performance, three output files are generated. Two data files, `class_a_out.dat` and `class_b_out.dat`, contain all events of the corresponding test data sets, with an *additional column (column 1)* which provides the network output for the corresponding event. These files can then be analyzed, *e.g.* in PAW (`netout.kumac` provides an example).

The program also generates a file `net_weights.dat` with the final weights of the neural network. `example.c` shows how to initialize and read in an already existing network configuration `net_weights.dat` and run it on data.

`diag.c` contains a couple of subroutines that are specific to the problem of γ/hadron separation in Milagro. As an example, the subroutine `qPlot` called at the end of `neural.c` generates a file `q_plot.dat` which gives γ and proton efficiency (column 2 & 3) and quality factor (column 4) as a function of the cut in the network output (column 1).

References

[1] G.B. Yodh, R. Atkins, *A Simple Algorithm for Hadron Rejection in Milagro*, Memos 39, 42, &46 (1999).
[2] R.S. Miller, *An Image Analysis Approach to Analyzing Milagro Data*, Memo 45 (1999).
[3] C. M. Bishop, *Neural Networks for Pattern Recognition*, Oxford University Press (1995).