Identification of Mixing of $D^0$ and its Antiparticle

Using Neural Network Software

Austen W. Rau
Dept. of Physics
The Ohio State University
Columbus, OH 43210-1117

Physics REU Summer 2005

Collaborators: Prof. Richard Kass and Dr. Amir Rahimi
Dept. of Physics
Ohio State University
Columbus, OH 43210

Abstract

This summer I used the neural network software MLPfit to distinguish between signal and background in data to study the mixing of $D^0$ and $D^{0*}$-bar. I used Monte Carlo data from the BaBar experiment [1] to train the neural networks, which then determine whether a certain event was a signal or another process (background) by providing a binary output. This study has shown that neural networks are very effective in separating signal and background.
Introduction

Mixing is a process that occurs when a particle spontaneously turns into its antiparticle. Figure 1 shows the normal decay of D$^0$ and the mixing of D$^0$ and its decay products. For mixing to occur, it must obey certain conservation laws. Since the additive quantum numbers (electric charge, baryon number, and lepton number) of a particle and its antiparticle are opposite in sign, they must vanish. Mixing has already been detected for K$^0$ and B$^0$ but not D$^0$. All three of these particles are bound states of a quark and an antiquark (K$^0$=d s-bar, B$^0$=d b-bar, D$^0$=c u-bar) [2]. Since quarks have baryon number +1/3 and antiquarks have baryon number -1/3, the total baryon number for these particles is zero. The lepton number for these particles is also zero, because there are no leptons involved in this process. Total electric charge is also zero for these particles (q$_d$=-1/3, q$_s$-bar=+1/3, q$_b$-bar=+1/3, q$_c$=+2/3, q$_u$-bar=-2/3, in units of proton charge). Because the additive quantum numbers equal zero, they will all be conserved if mixing occurs.

![Figure 1. a) Normal decay of D$^0$. b) Mixing of D$^0$.](image)

The Standard Model predicts that mixing of D$^0$ should occur approximately once out of every 10 billion events that produce a D$^0$. Figure 2 shows the Feynman box
diagram that describes the mixing according to the short-distance effects in the Standard Model. Other long-distance effects predict larger mixing rates in the Standard Model. One of the approaches to estimating the long-distance contributions is the dispersive approach [3, 4], which predicts larger mixing rates than what is predicted by the Feynman box diagram. The BaBar detector is currently capable of detecting mixing rates of only about $10^{-4}$. This analysis will test the Standard Model. If mixing rates are larger than what the Standard Model predicts, this could be evidence of physics beyond the Standard Model.

Detection of mixing, however, is no simple task. $D^0$ has a lifetime of only $4 \times 10^{-13}$ seconds, and its electric charge is zero, which make it very difficult to detect a $D^0$. So we examine the decay products of $D^0$ and $D^0$-bar to detect mixing. Unfortunately, other processes with similar decay products to $D^0$-bar can mimic mixing, so we must use some

**Figure 2. Feynman box diagram for mixing of $D^0$ and $D^0$-bar.**
method of data analysis to determine which processes are true signal and which are background.

Figure 3. Example of the structure of a neural network.

Neural Networks

Neural networks [5] are a complex means of modeling complicated functions. They are designed as an extremely simplified model of the human brain with feedforward structure. They consist of usually three layers of “neurons” (input, hidden, and output layers) connected by “synapses.” Each neuron accepts input, which is then passed through a transfer function to produce the output. The transfer function of the input and output layers is linear, while the transfer function in the hidden layer is the sigmoid function, \( y = \frac{1}{1 + e^{-x}} \). The purpose of the sigmoid function is to take a wide range of inputs and produce output in a much smaller range. The synapses represent weights, such that the value of each neuron is a weighted sum of the outputs of the previous layer. Figure 2 shows an example of a neural network.
This summer I used MLPfit [6], which is computer software written in C++ that was developed at CERN for use in research that creates and trains neural networks. MLPfit trains the neural network by examining a set of data with both inputs and outputs and creating a function with the lowest possible error, 

\[ E = \frac{1}{2} \sum_p \omega_p (o_p - t_p)^2 \]

where \( \omega_p \) is a weight which is usually set to 1, \( o_p \) is the output of the function, and \( t_p \) is the known output. This process is done by iteration. Initially the weights are all random values between -1 and 1. MLPfit then feeds the inputs into the neural network function and compares the output to the known output. After calculating the error, MLPfit attempts to lower the error by changing the values of the weights using various learning methods. The learning method that I used in the following studies is the Broyden, Fletcher, Goldfarb, and Shanno (BFGS) method [6], named for its creators. This process is repeated until the error converges by reaching a minimum. MLPfit actually requires two sets of data, the learning data, which is actually used to train the neural network, and the test data, which is used to test the output function on data that is not used in training. Errors on both sets of data must converge in order to ensure that the neural network will accurately predict the outputs.

**Early Studies—Two Gaussian Distributions**

In order to understand MLPfit, I spent some time training the neural network with the trivial case of two Gaussian distributions that do not overlap. I generated these Gaussians by adding twelve random numbers from 0 to 9 and either adding or subtracting 50 to change their means to 4 and 104. I also assigned each Gaussian with an output, such that if an input fell into the range of the Gaussian of mean 0, the output would be 0,
and if an input fell into the range of the Gaussian of mean 100, the output would be 1. This binary output is commonly used when solving classification problems using neural networks. I then used sets of data consisting of equal numbers of points from each Gaussian to train the neural network.

![Output using 600 Data Points](image1)

![Output using 2000 Data Points](image2)

**Figure 4.** MLPfit output functions for two non-overlapping Gaussian distributions with different number of data points.

I used these sets of data to examine the different parameters that must be specified when training a neural network using MLPfit. These parameters include number of data points, number of epochs (iterations used to minimize the error), learning methods, and various parameters corresponding to each learning method. The study shown in Figure 4 shows the output functions for data sets with different number of data points corresponding to two Gaussians. This particular study shows the importance of having as large a data set as possible, because as the number of data points increases, the output function approaches the expected output function, a step function. I also conducted studies with two additional data sets, one with two Gaussians that barely overlapped, and the other with two Gaussians that overlapped significantly. These studies were used to
examine the capabilities of neural networks and to determine the appropriate values for each of the parameters.

**Identification of Signal**

The BaBar detector provides us with data that we can use to train the neural network. There are fourteen different variables that are useful in this analysis that the detector measures, but some of these variables will prove to be more useful than others when training the neural network. The distributions of the signal and background for these variables must have some separation to be useful in training a neural network. The more separation a variable has, the more that variable will contribute to an accurate approximation of a function.

With the Monte Carlo data from the BaBar experiment, seven variables have good separation. One of these variables, the invariant mass of the $D^*-D^0$ system, will be used to fit the final data, so we have six variables that will be useful for this study. I began with training the neural networks starting with one input to see which variables would give us the most information about the function. I then tried different combinations of two, three, and four inputs to try to find the best possible combination of variables.

Figure 5 shows the distributions of the signal and background for three of the inputs that had the best separation. For this study the signal mode is represented by $D^*$ decaying into $D^0$-bar and $\pi$ (and charge conjugate mode). The $D^0$-bar then decays into $K^+$, electron, and antineutrino. One of the background samples is the wrong sign background, which is represented by $D^*$ decaying into $D^0$-bar and $\pi^*$; but also detecting a random $\pi^+$, causing the final decay products to look similar or identical to the
signal mode. There are also other backgrounds present in this study, including the peaking and combinatoric backgrounds, which consists of random track combinations of kaon, pion, and electron candidates.

Using these inputs to train the neural network, MLPfit provided a function that separated the signal from background very well. The function is trained so the background peaks near 0 and the signal peaks near 1. One figure of merit that we use to determine how well the signal and background are separated is signal over the square root of background. The larger this value, the better the separation. In order to avoid dividing by zero, I programmed the output to return -1 if the background was zero. In Figure 6 I used the test data to check the output of the MLPfit function. This data was sorted depending on the known output and used as inputs in the function to check the output of the function. The signal over square root of background peaks at approximately 0.8, so if we selected events with output greater than 0.8, we would have excellent signal to background separation.
Figure 6—Converging error and output of MLPfit function on test data.

**Conclusion**

The study shown is just one of many studies conducted in the search for the best separation of signal from background. Neural networks are used in this case because they are expected to perform much better than traditional methods. These functions, however, have only been used with Monte Carlo data and not with real data from the detector. Further studies must be conducted in order to find the actual mixing rate of the process of mixing of $D^0$ and its antiparticle.
References:

  http://www.slac.stanford.edu/BFROOT/


  http://www.statsoftinc.com/textbook/stneunet.html

  http://schwind.home.cern.ch/schwind/MLPfit.html