1 Experimental Introduction: Fermilab and D0

The Tevatron at Fermi National Accelerator Laboratory is currently the worlds's most powerful particle accelerator. It accelerates protons and antiprotons in stages and then collides them at a center-of-mass energy of 1.96 TeV. The Tevatron accelerates these particles towards each other to collide at two locations, one being inside the D0 Detector, and the other being inside the CDF Detector. Both were built to study high mass states and phenomena with large transverse momenta. These include the search for the top quark (discovered in early 1995 at Fermilab), the production of bottom quarks (which were also discovered at Fermilab in the summer of 1977), a high-precision study of W and Z bosons to test the electroweak theory and determine extremely accurately the masses of the W and top quark, a detailed analysis of QCD phenomena, and also phenomena beyond the standard model. While the Higgs boson remains undiscovered, knowing the mass of the top quark to high precision allows physicists to zero in on the Higgs mass.

The D0 Collaboration includes physicists from over 100 institutions from 19 different countries. The detector is thirty feet tall and fifty feet long. The collision point inside of the D0 detector is surrounded by subdetectors. The first detector that a particle passes through is the tracking system, which is built of silicon detectors. This records tracks that particles leave behind from interacting with the material and gives very high precision energy and momentum measurements. Outside of the silicon detectors, there are scintillating fibers, which will produce photons to be carried along the fiber to a photo-multiplier tube, or PMT, which converts the incoming light into a current. The whole tracking system is in a powerful magnetic field so that the charges and momenta of the particles can be determined.

Following the tracking system outwards from the center, the next layer of the detector is the electromagnetic calorimeter, and it is followed by the hadronic calorimeters. Particles that interact electromagnetically are dectected in the electromagnetic calorimeter. The hadronic calorimeter is built with alternating layers of a dense material followed by a more active material. A particle entering the dense material (depleted uranium) will interact with the material and initiate a shower of particles, if the original particle interacts via the strong force. This shower will be recorded in the active material (liquid argon).

Muons are detected in the third subsystem, known as the muon detection system. It is composed of several toroidal magnets, which again help physicists determine the momentum and charge of the muon. The muon system also includes proportional drift tube chambers, or PDTs. The muon detection system is the outermost layer because muons do not get absorbed in the calorimeters. Finally, the trigger system should be mentioned because it is also a very important part of the detector. There are 2.5 million collisions a second, of which a maximum of twenty a second can be stored. The trigger system decides which events are interesting enough to store and discards the rest of the data.

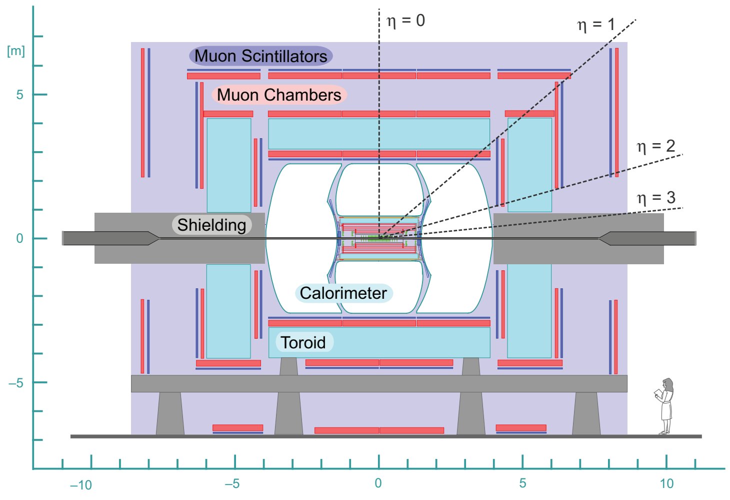

Figure 1: A cutaway side view of the D0 detector, showing the different subdetectors.

2 The Physics of Vector Boson Fusion

A cross section is a way of indicating the probability of interaction between two particles. It uses the simple concept of suggesting a particle has a certain surface area which another particle can collide with, assuming no force interactions. As such, it is expressed in units of area. The barn is the standard unit of area for measuring cross section, and it is equal to 10^-24cm^2. The name for the unit ”barn” comes from two Purdue scientists, Marshall Halloway and Charles Baker, who were working on the Manhattan Project. The barn is approximately the size of a uranium nucleus, which is considerably larger than the neutrons they were firing at uranium nuclei, and hence the uranium nucleus was as ”big as a barn”. The slang name helped to conceal the scientific nature of the subject matter. The term was declassified after the war.

Weak Boson Fusion (WBF), the subject of this analysis, has a low cross section. Therefore, it requires a large amount of data to verify the process is occuring. This study was conducted at a time when a necessary amount of data has been collected from the detector.

As the beams accelerate towards each other, they emit braking radiation, also known as ”Bremstrahlung”. In WBF, the radiation specifically must be a weak force carrier. When both beams emit Weak Bosons (any weak force carrier), they can potentially interact or collide, emitting various other particles. The process of interest in WBF is:

$u + u \Rightarrow d + u + W + Z \Rightarrow d + u + W \Rightarrow u + d + \nu_e + e$

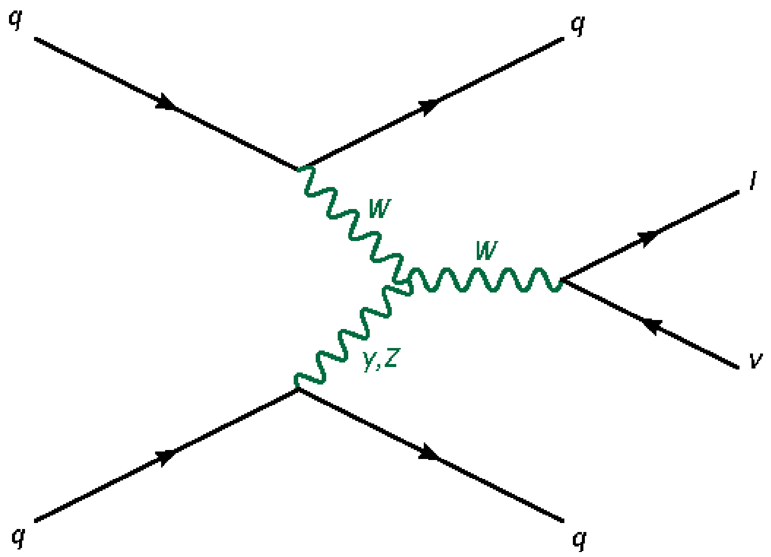

Figure 2: This is the Feynman diagram of the primary signal in Weak Boson Fusion. The jets are found in the forward regions at a high η as the emitted particles cannot throw the momentum of the proton (or anti-proton) far off of their original course.

$u + u \Rightarrow d + u + W + \gamma \Rightarrow d + u + W \Rightarrow u + d + \nu_e + e$

This is the main process of interest, and this process shall be known as the "signal". There are of course other interactions occuring, which may have signatures similar, at least in part, to that of WBF. These include:

$u + u \Rightarrow \gamma \Rightarrow d + u + W \Rightarrow u + d + \nu_e + e$

$u + u \Rightarrow g \Rightarrow d + u + W \Rightarrow u + d + \nu_e + e$

To differentiate between the two, an advanced statistical classification algorithm is required. A popular technique in high energy physics is the use of an artificial neural network, or ANN. In this study, the ANN was applied using a class of the C++ programming language called TMultiLayerPerceptron. Root is used to implement TMultiLayerPerceptron and is an ideal way to use the ANN because root allows practical displaying options and advandced data analysis.

3 Artificial Neural Networks

It is easy to locate a point on a line to split a one-dimensional vector into two classes, one above a certain threshold value, and one below. It becomes more difficult to draw a line to seperate a two-dimensional vector into classes above or below a certain threshold value, especially when

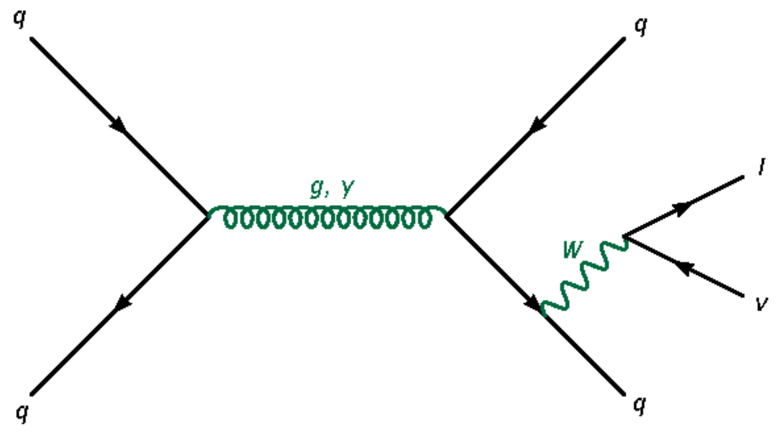

Figure 3: This is the Feynman diagram of the primary background in Weak Boson Fusion. It has a similar signature but, among other differences, the jets are not found in as far forward a region.

there is uncertainty in both components of the vector. The creation of the artificial neural network allows an n-dimensional hyperplane to divide the classes, in an analagous fashion as the point or line mentioned above. The two classes in our study are, as mentioned earlier, events that are signal-like and events that are background-like.

3.1 Neurons (Nodes)

Artificial neural networks, or ANNs, seek to mimic the success of the human brain's neural network in pattern recognition. A neuron, or node, adds up a set of input quantities, compares that value to some threshold, and then indicates whether the threshold is exceeded, and depending on the degree of complexity of the node, the node can indicate to what degree the threshold is exceeded. As an example, consider how humans decide what a random object is. Suppose the object is a vegetable (and the output node is a measure of how much like a vegetable the object is). On a scale from 0 to 1, things must show certain properties of a vegetable to justify firing the vegetable node, such as "crunchiness", "leafiness", "greenness", etc. Something that might score high on all of those parameters (which can be considered input variables) is a head of lettuce (assume the sum of all of its properties add up to 3 and the vegetable node passes a "heck yes"). Something that might not score quite as high is a clove of garlic, although it is still a vegetable (assume the sum of all of its properties only add up to 1.5 and the vegetable node passes a "I suppose"). Finally, the properties of a cellular phone do poorly at the vegetable node (assume the sum of all of its properties only add up to .1 and the node simply doesn't fire). However, lettuce doesn't score nearly as well on other parameters such as "magnetic" or "hardness", so those input nodes are kept quiet. The more vegetable properties something has (and the more vegetable property nodes that are turned on), the better an example it is of a vegetable (and the higher the activation level of the vegetable node). Similarly, as shall be seen later, the more signal properties an event has in the ANN of this study, the better an example it is of a signal-like process.

The simplest ANN is the single layer perceptron, developed in 1958 by Frank Rosenblatt, a computer scientist at Cornell University. In his own words, he "hoped that the fundamental laws of organization which are common to all information handling systems, machines and men included, may eventually be understood." The single layer perceptron is simply an input node (or multiple input nodes), and one output node (like the vegetable property input variables and the vegetable node). It is capable of classifying linearly seperable patterns, and it generally gives as its output a large value (signal, success, true, etc.) or a small value (background, failure, false, etc.). After receiving an input, each node will assign weighting to a path which will be multiplied by the input value. (For example, the vegetable node automatically looks more rigorously at properties like "greenness" than it does at properties like "magnetic".) It will then classify the inputs into two classes, and there will be some "line down the middle" which all of Class 1 is above and all of Class 2 is below. Specifically, this "line" is actually the hyperplane, as mentioned above, defined by where the dot product of the weight matrix and input matrix is equal to 0. By lying on seperate sides of this hyperplane, the two classes can be defined as linearly seperable. Since they are linearly seperable the single layer perceptron can be trained, via weighting as mentioned above, to find this hyperplane. Multi-layer perceptrons are used in the case of things that aren't so tidy (specifically, patterns without linear seperability). The single layer perceptron cannot classify non-linearly seperable patterns. In fact, ANNs declined in popularity when it was pointed out by Rosenblatt's bitter rival, Marvin Minsky of MIT, that single-layer perceptrons were incapable of learning the "exclusive-or" function (known as "xor", and meaning "Class 1 or Class 2, but not both"), which, among other fatal flaws, he incorrectly postulated could not even be learned by multi-layer perceptrons. Rosenblatt died in 1969, the same year Minsky proved the failure to learn the "exclusive or" function, and the combination of his death and the proof became the downfall of research and funding into neural networks, until Minsky himself began to work with neural networks.

The failure of the perceptron to interpret the exclusive-or function is a simple yet important concept to understand. Suppose, as in Figure 4, that two paths are weighted as shown (these values are set and not random). In the following figures, green represents a node firing, red represents a node as silent, and yellow represents a uncomputed decision. The xor node sums the inputs and fires if the sum is greater than 0.5. If node A is on, then the path is multiplied by A's weight (for simplicity assume it is 1), and likewise for B (also 1). The xor node should not fire, but since the sum is far above its threshold (0.5) the node is shouting to the next layer when it should be silent. Much smaller weights will also fail at the xor node because their sum will be greater than 0.5 and again the node will see it as a success and fire to the next layer, but both paths individually will fail and the node should be silent. For reasons such as these, Minsky wrote off the perceptron as a waste of research money and time. Of course there is a solution, which is to have the xor node compute "(A or B) and not (A and B)" by breaking up the decision, as in Figure 5. In that figure, the first multi-layer perceptron of this discussion, the decision was broken up into an “or” and an “and” decision. The “or” decision node handles the “or” case as it logically should, but the “and” node does not compute “and”. The node is only capable of comparing a value to a threshold. If that value is not reached, it will not fire. It will always sum the inputs. That being said, the reader can walk his or her way through the four examples which show how the multi-layer perceptron is capable of handling the “xor” function. The node only fires a 1 if the threshold is exceeded. That value is multiplied by the weight and the product of the two is the input to the layer which follows.

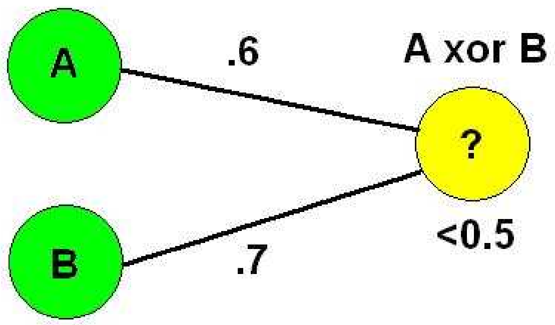

Figure 4: This figure shows how a perceptron fails the "exclusive-or" logic operation.

3.2 The Multi-Layer Perceptron

The multi-layer perceptron (MLP) uses similar concepts to attempt to classify non-linearly seperable patterns. The MLP, unlike the single layer perceptron, requires a desired output

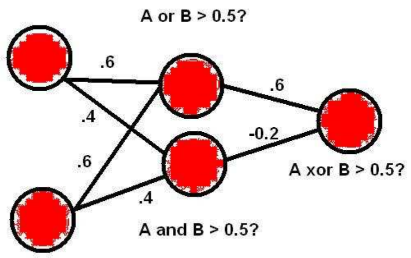

Figure 5: This figure shows how an additional layer of nodes allows the perceptron to handle the "exclusive-or" logic operation for the (0,0) case, which should fail.

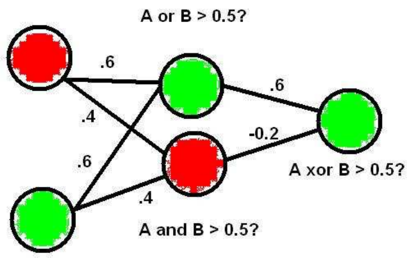

Figure 6: This figure shows how an additional layer of nodes allows the perceptron to handle the "exclusive-or" logic operation for the (0,1) case, which should pass.

value or values which it aims to meet. The MLP then uses a process called "error back-propagation" to train itself. The process of the back-propagation in the MLP begins with the initial weighting of the paths connecting each node, which needn't be assigned and can be random. Recall that an individual node's weight is the sum of all of the inter-node connectors' weights that feed into it. As the initial data is fed through and weighted, the MLP's output is compared to the desired output. After this comparison, one inter-node connection weight is modified. This new weight is then passed through the entire network and the ANN compares the value to the previous value and determines whether the output has become more signal-like or more background-like. This is repeated thousands of times, and each time the MLP tries to lower its error and create the desired output. If a relatively high error still occurs at a node after several iterations, then "blame" begins to be assigned to the paths behind the node in question.

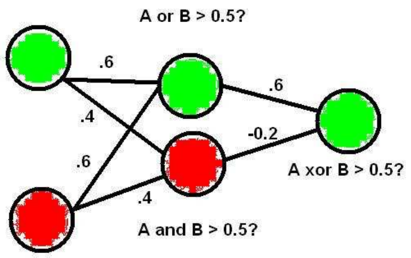

Figure 7: This figure shows how an additional layer of nodes allows the perceptron to handle the "exclusive-or" logic operation for the (1,0) case, which should pass.

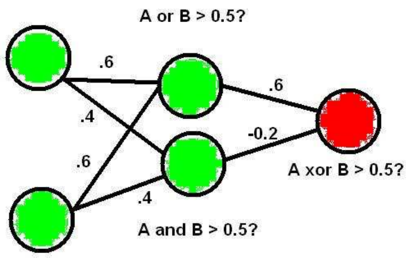

Figure 8: This figure shows how an additional layer of nodes allows the perceptron to handle the "exclusive-or" logic operation for the (1,1) case, which should fail.

This raises the threshold of the node, and it becomes harder for the paths entering the node to sum higher than the node's new threshold, and thus harder to justify a signal-like event. This effectively begins to reroute the responsibility of the node, which is passed on to other nodes which produce less error. Additionally, with less input into these "blamed" (high error) nodes, the sum of the weighted inputs again does not as easily climb above the new threshold value, and thus the node is "quieter" when providing output, and vice versa with the nodes producing less error. In the Neural Network Results image in Figure 10, the more "responsible" paths are represented with thicker lines. The MLP attempts to get closer and closer to recognizing the hyperplane's values. Due to the multiple layers and nodes, the MLP can't know the effect of changing a single weight on the whole network without back-propagation. It uses the derivative of a global error function (usually the sigmoid function as in Figure 9) with respect to that individual weight. By taking the partial derivative of the error of the entire network with respect to each weight the inherent minimum will eventually be found.

To get an idea of how back-propagation works, consider how one finds the error of the network due to a change in the weight of an input in the single layer perceptron:

$\Delta w_i = x_i \delta$

where Δw_i = a change in weight, x_i = input at the $i^t h$ node, and δ = (desired output - actual output), or $\delta = d_i - y_i$

For the MLP, the equation is similar:

$\Delta w_i = \eta x_i \delta$

where η = learning rate (a quantity set by the user)

The sigmoid function is implemented in the following way:

$y = \frac{1}{1+e^{-x}}$, and $y' = x\left(1-x\right)$

The δ function for the output neurons, however, is now:

$\delta = \left(d_i - y_i\right) * y_i \left(1 - y_i\right)$

and for hidden layer neurons:

$\delta_p\left(q\right) = x_p\left[1 - x_p\left(q\right)\right] \sum w_{p+1}\left(q,i\right)\delta_{p+1}\left(i\right)$

As can be seen in the subscript of the weight and δ, the error is based on the weights and errors of the layer following the layer in question. This is how error based on a change in weight is propagated throughout the network.

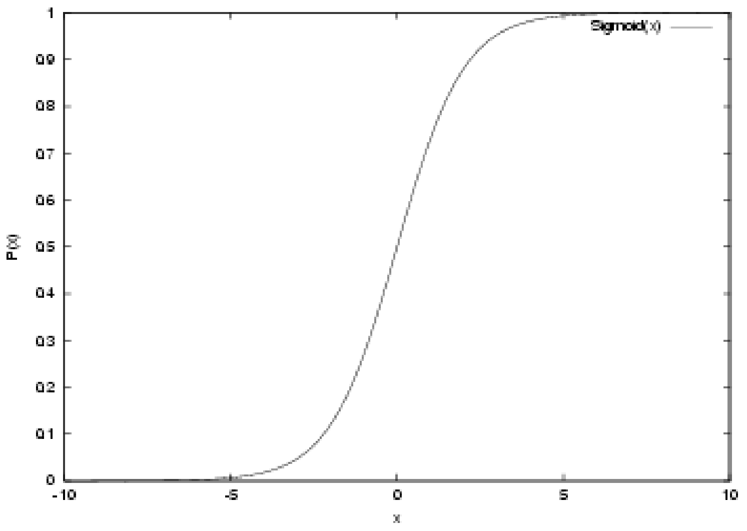

Figure 9: The sigmoid function is often used in error back-propagation calculations.

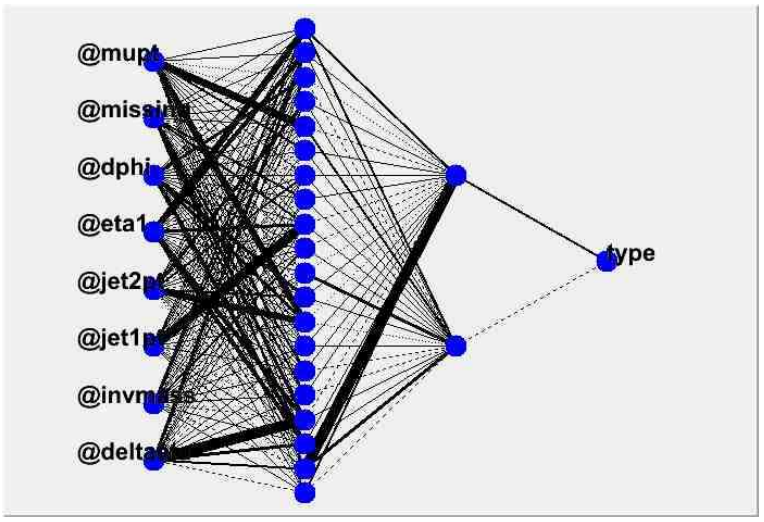

Figure 10: This figure shows how some inter-nodal paths are weighted more heavily than others. This neural network uses eight input variables, twenty first layer hidden layer nodes, and two second hidden layer nodes.

This day and age, the MLP is a powerful tool when facing a problem with multiple variables in which any one variable alone cannot be relied upon to distinguish. Each input variable can be considered one dimension of a vector, and an n-dimensional distribution of points based on the n different variables can be visualized. Training (via desired output and back-propagation) is necessary to find what exactly makes a part of the sample desirable, and then the non-linear hyperplane that seperates the sample into signal and background is closed in upon. If the ANN begins to magnify small differences between samples, which may be due to statistical fluctuations, or the sample is too small, the hyperplane will be wrongfully crafted. This is an effect known as overtraining.

Up until this point only the training of the ANN has been mentioned. The second step is the testing of data, which should be a set of data entirely seperate from the training data. The new set of n-dimensional data are plotted in the fashion of the training data, and the hyperplane discovered in the training is used to distinguish, in the case of WBF, the signal events from the background events.

4 A Weak Boson Fusion Search

To use the discriminating power of the ANN to identify evidence of weak boson fusion in the D0 detector, the ANN must be tailored. It must be properly trained with signal event and background event data. This data comes from an event generator, which is a computer program that replicates what happens at a detector, and helps physicists see results, which may or may not be accurate. The data can be verified by experiment to prove the accuracy of the generator. The ANN in this study was originally given signal event data generated using Madgraph. However, at the time of this study, another event generator, named VBFNLO, which was built specifically for vector boson fusion, was published. The signal files were generated using VBFNLO. The background files were generated with Alpgen and Pythia Monte Carlos.

4.1 Variable Selection

The ultimate goal of the ANN created in this study is to distinguish signal-like events from background-like events. For the neural network to be able to strongly discern between signal-like events and background-like events, it must be trained with certain inputs which themselves differ significantly between signal and background. Comparing Figure 28 and Figure 24 shows the difference between an effective input variable and a poor input variable. The variables for the neural network were chosen based on this requirement.

These variables were (as used in vbfNNTrain.C):

jet1pt - The transverse momentum of the leading jet. See Figure 21.

$p_{t1} = \sqrt{p_{x1}^2 + p_{y1}^2}$

jet2pt - The transverse momentum of the second-leading jet. See Figure 22.

$p_{t2} = \sqrt{p_{x2}^2 + p_{y2}^2}$

eta1 - The pseudo-rapidity of the leading jet. See Figure 23.

$\eta_1 = -\ln\left(\tan\left(\frac{\theta_1}{2}\right)\right)$

where $\theta_1 = \arctan\left(\frac{p_{t1}}{p_{z1}}\right)$

deltaeta - The difference in rapidity between the leading two jets. See Figure 24.

$\delta\eta = |\eta_2 - \eta_1|$

dphi - The difference between azimuthal angles of the leading and second-leading jets. See Figure 25.

$\delta\phi = |\phi_2 - \phi_1|$

where $\phi = \arctan\left(\frac{p_y}{p_x}\right)$

missing - The missing transverse energy. See Figure 26.

wtrmass - The transverse mass of the W Boson. See Figure 27.

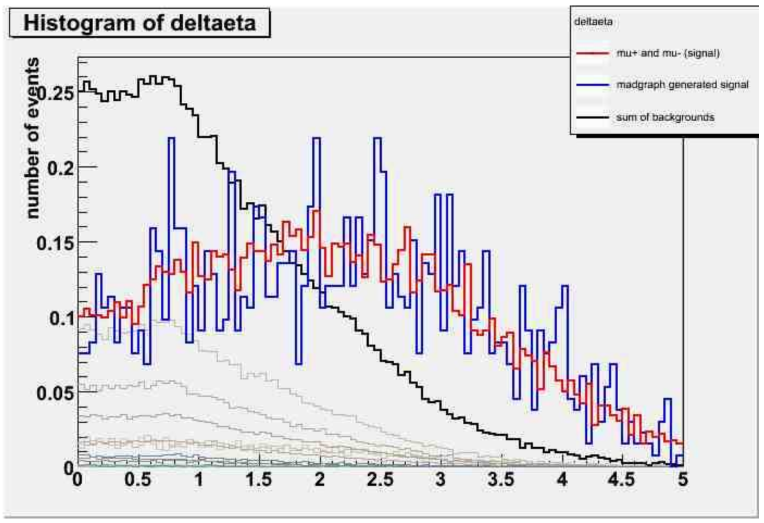

Figure 11: This image shows how the distribution of the signal events (Madgraph in blue, VBFNLO in red) differs significantly in shape from the distribution of the background events (in black), meaning that it is a good variable to train the ANN.

mupt - The leading muon transverse momentum. See Figure 28.

invmass - The invariant mass of the two leading jets. See Figure 29.

$\sqrt{(E_1 + E_2)^2 - \left((p_{1x} + p_{2x})^2 + (p_{1y} + p_{2y})^2 + (p_{1z} + p_{2z})^2\right)}$

As can be seen, these can all be considered good variables because the signal differs in shape from the background, with the possible exception of mupt, wtrmass, and missing.

With the variables chosen, the next step is to instantiate and optimize the ANN. Optimization is a way of controlling the training to produce the most effective possible ANN with the lowest possible error. Optimization requires the output and end result of the ANN's training and testing, so that the user can see when the error was minimized. Upon completion of training and testing, TMultiLayerPerceptron displays several "Neural Network Results" plots. These will be introduced here when referred to in the next section.

To optimize the ANN, there are four main "knobs to turn": the number of hidden nodes, the number of hidden layers, the number of training epochs, and the training type.

4.2 Number of Hidden Nodes, Hidden Layers, and Training Epochs

The number of hidden nodes is precisely that: the number of nodes in the hidden layer. More nodes imply a more complex network, but may reduce overall error by providing new paths

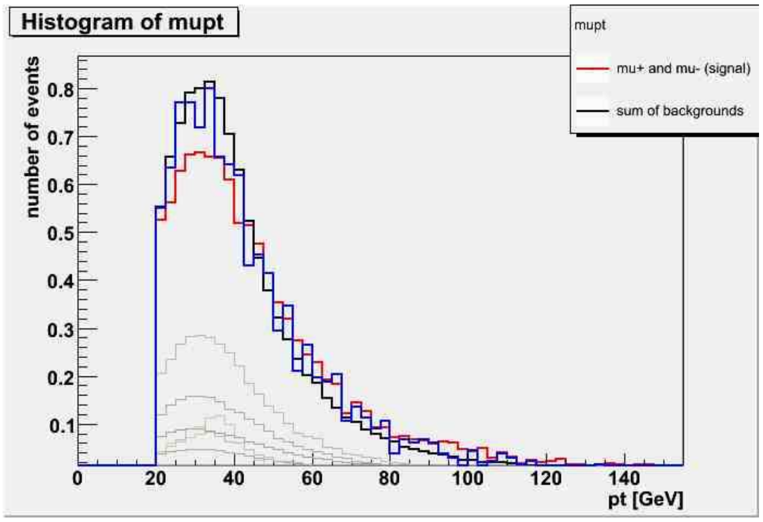

Figure 12: This image shows how the distribution of the signal events (Madgraph in blue, VBFNLO in red) does not differ significantly in shape from the distribution of the background events (in black), meaning that it is not a good variable to train the ANN.

for weighted inputs to flow from "blamed" nodes. Too few nodes may hinder the ability of the ANN to realize certain correlations between the input variables. Figure **??** shows a diagram of the ANN, with lines connecting the nodes, upon completion of several training epochs. If a particular node is more discriminating based on a certain input variable, the line connecting that variable to that node will be thicker. This is a good way to make sure that all of your input variables are good ones to be using. An input node with only thin lines reaching to the hidden layer nodes indicates that the input parameter isn't a powerful discriminator. Figure 13 shows a plot which displays the impact of the variable on the entire ANN with respect to a change in the input. If a certain variable was discovered by the neural network to be more discriminating, it reaches higher on the plot. If only a small deviation in that variable has a large effect on the neural network, it would be shown by a high peak near the left side of the graph, such as $\delta\phi$.

More hidden layers are a similar case: they also imply a more complex network but again can reduce error and accentuate correlations.

The number of training epochs is perhaps the most straightforward to optimize- the number is optimized when additional epochs no longer decrease the overall error of the ANN. 14 is an output image of TMultiLayerPerceptron. The error is minimized at point A and any further training may only hurt the discerning capability of the ANN.

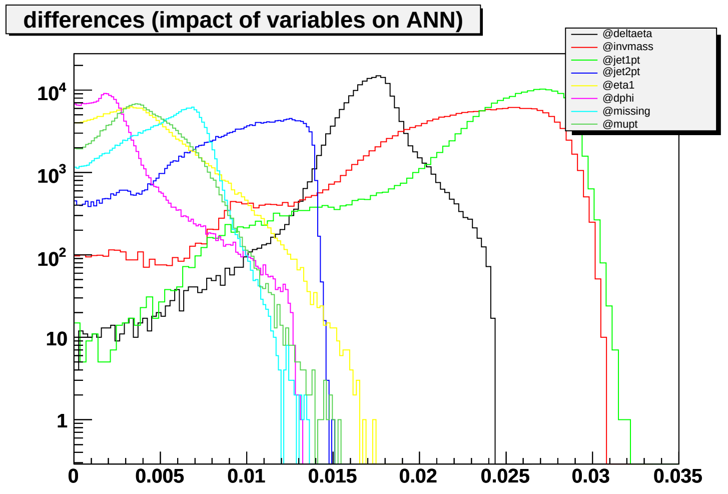

Figure 13: This figure shows the impact of different variables on the entire ANN as a function of how much the weighting was changed.

4.3 Training Method

The training type uses different algorithms or formulas to minimize the error. The different training methods that can be implemented by TMultiLayerPerceptron include Stochastic Minimization, Batch Learning, the Steepest Descent Algorithm, Conjugate Gradients (Fletcher-Reeves), Conjugate Gradients (Polak-Ribiere), and the Broyden, Fletcher, Goldfarb, and Shanno Method.

4.4 Optimization Procedure and Results

In addition to the error plots generated by TMultiLayerPerceptron, the signal significance was used as an additional indicator in choosing the optimized ANN. Signal significance is defined as: $\int_n^{100} \frac{S}{\sqrt{B}}$ where n is an arbitrary point, S is the NN weight of the signal sample and B is the ANN weight of the background sample. The larger the signal significance, the greater the discrimination between signal-like events and background-like events, and the better the ANN. The signal significance is best visualized in Figure 15.

After attempting several trials with the different training methods, it was deemed that BFGS was the best training method. Figure 16 shows how the different methods compared against each other, with a range of nodes, hidden layers, and epochs.

Keeping the best training methods (FR and BFGS) and removing wtrmass, missing, and

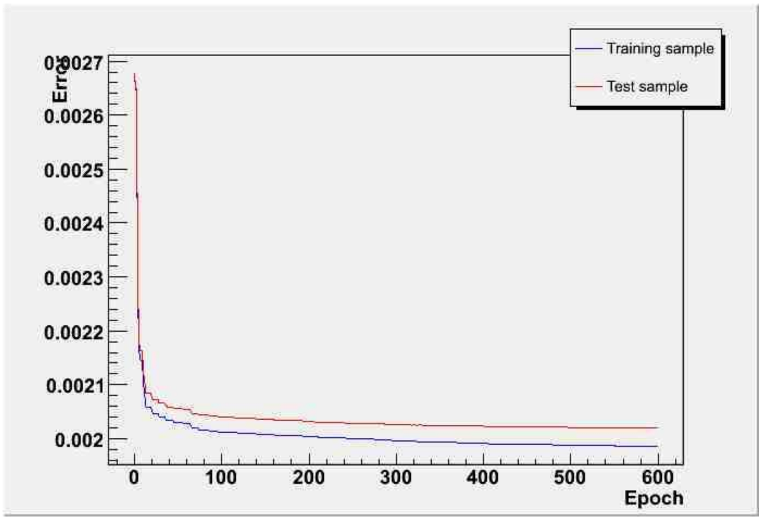

Figure 14: This figure shows how the error of the entire ANN diminishes with each passing epoch. Within the first 100 epochs, the error levels out and does not diminish much more. If the error were to start to rise, the optimal number of epochs would occur before this point.

mupt from the variable set, the ANN was tested again. The motivation behind this step was to see if the variables that did not show a significant difference in the spreads of signal vs. background were hindering the ANN. From inspecting Figure 17, it can be seen that these variables do not hinder the ANN and removing them may hurt the ANN.

In Figure 18, the six different training types are plotted against the ANN error. Again, BFGS was chosen due to its consistently low values and was reinforced as the best training method due to its consistently high signal significance as mentioned above.

After BFGS was chosen as the dominant training method, the number of nodes and hidden layers was chosen. In Figure 19, the number of hidden layers is plotted against the signal significance. It appears that two hidden layers is the best value. However, the number of nodes and the number of epochs was not fixed when recording this data. Further trials should be completed to enhance confidence that two hidden layers is indeed the best value.

Although training epochs seem to be the easiest value to optimize, the ANN seemed to always reduce its error, even with values as high as 750. The optimal number of training epochs need not be optimized unless all other quantities have been optimized. Since there is a lack of confidence in how optimal the other values truly are, the number of training epochs was not officially optimized.

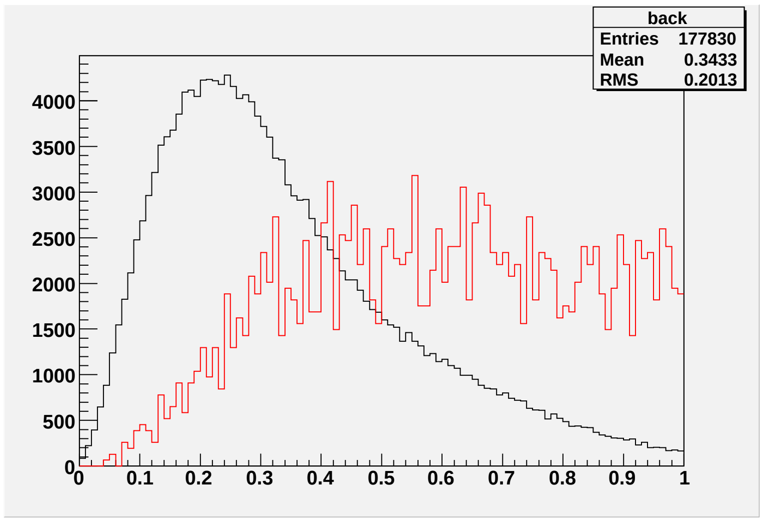

Figure 15: This figure shows the weighted signal-like events as closer to one and the weighted background-like events as closer to zero.

Finally, the number of nodes was chosen. In Figure 20, the number of nodes is plotted against the signal significance. It appears that 14 nodes is the best value. However, the number of hidden layers and the number of epochs was not fixed when recording this data. Further trials should be completed to enhance confidence that 14 nodes is indeed the best value.

5 Discussion

The optimization of this particular ANN is in its infancy. It seems fairly clear that the optimal training method is BFGS and the variables were chosen well; the optimal number of nodes, epochs, and layers, however, is more convoluted. At this time, the optimal number of nodes seems to be 14, the optimal number of epochs is unknown, and the optimal number of layers seems to be two.

6 Appendix 1: An Example of Back-Propagation

An example of back-propagation helps to show the seperating power it produces. ANNs often use the sigmoid function,

$y = \frac{1}{1+e^{-x}}$

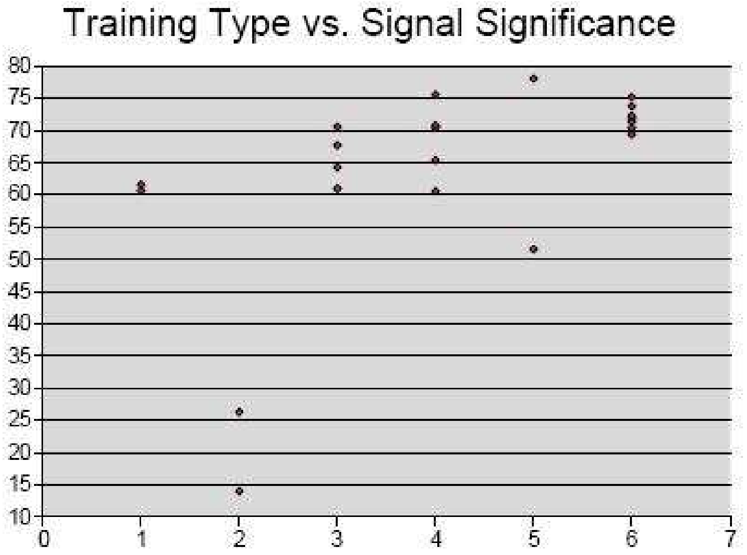

Figure 16: This figure shows how the different training types compare to each other, where the x-axis is the training type (1-6 as listed above) and the y-axis is the signal significance. Method 6, BFGS, was chosen due to its consistently high values.

as the error function, whose upper limit approaches 1 and lower limit approaches 0, as seen in Figure 9.

Nodes are defined by $x_{l-1}(n)$ where l is the layer, and n is the node.

Consider an ANN with two input nodes, three nodes in the hidden layer, and one output node. First, random weights are assigned for each node. Weights are defined by: $w_l(f, n)$ where l is the layer, f is the neuron from the previous layer that its connection came from, and n is the number of the neuron. Random weighting gives:

Hidden node 1:
$w_2(0,1) = 0.341232$
$w_2(1,1) = 0.129952$
$w_2(2,1) = -0.923123$
Hidden node 2:
$w_2(0,2) = -0.115223$
$w_2(1,2) = 0.570345$
$w_2(2,2) = -0.328932$
Output node:

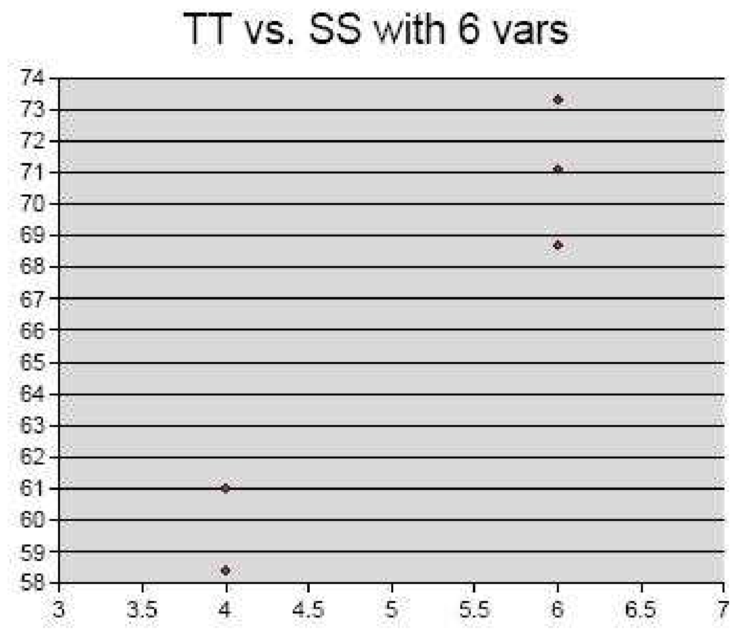

Figure 17: This figure shows how the best two different training types, based on the nine variable data set, compare to each other when there were only six input variables. The x-axis is the training type (4=FR, 6=BFGS) and the y-axis is the signal significance. Method 6, BFGS, was chosen due to its consistently high values.

$w_3(0,1) = -0.993423$
$w_3(1,1) = 0.164732$
$w_3(2,1) = 0.752621$

The bias for any neuron is given by: $w_l(0,n)$

Therefore for node 1, the bias is 0.341232, and for node 2, the bias is -0.115223. When these values are plugged into the sigmoid function, they return:

$x_2(1) = 0.584490$
$x_2(2) = 0.471226$

which are then passed onto the output layer, in addition to:

$x_3(1) = 1$ (bias)

The ANN would now output a sum of weights multiplied by node values:

$(1 * -0.993423) + (0.584490 * 0.164732) + (0.471226 * 0.752621) = -0.542484$

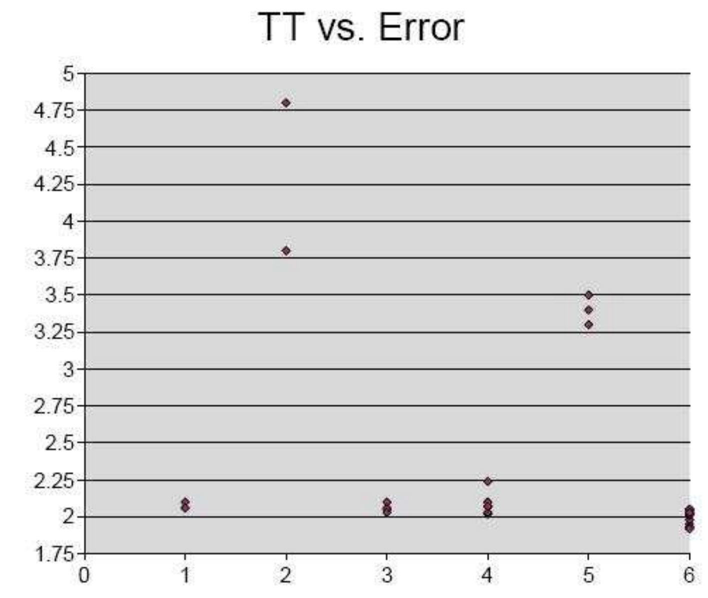

Figure 18: This figure shows how the different training types compare to each other, where the x-axis is the training type (1-6 as listed above) and the y-axis is the ANN error. Method 6, BFGS, was chosen due to its consistently low values.

Plugging this value into the sigmoid function gives 0.367610.

This is merely the first step. Now the weights must be adjusted to get closer to the result desired. Using the following formula, a value is produced that is propagated backwards.

$\delta_3(1) = y_p(1 - y_p)(d_p - y_p)$

$\delta_3(1) = 0.367610 * (1 - 0.367610)(0 - 0.367610) = -0.085459$

Propagated back:

$\delta_2(1) = x_2(1) * (1 - x_2(1)) * w_3(1,1) * \delta_3(1)$

$\delta_2(1) = 0.584490 * (1 - 0.584490) * (0.164732) * (-0.85459) = -0.0034190$

$\delta_2(2) = 0.471226 * (1 - 0.471226) * (0.752621) * (-0.85459) = -0.0160263$

These are the δ values for the layers. Now, the weights are altered. The multiplier in this example will be h=0.5.

$\delta w_2(0,1) = h * x_1(0) * \delta_2(1) = 0.5 * 1 * -.0034190 = -0.017095$

$\delta w_2(1,1) = 0$

$\delta w_2(2,1) = 0$

$\delta w_2(0,2) = h * x_1(0) * \delta_2(1) = 0.5 * 1 * -.0160263 = -0.0080132$

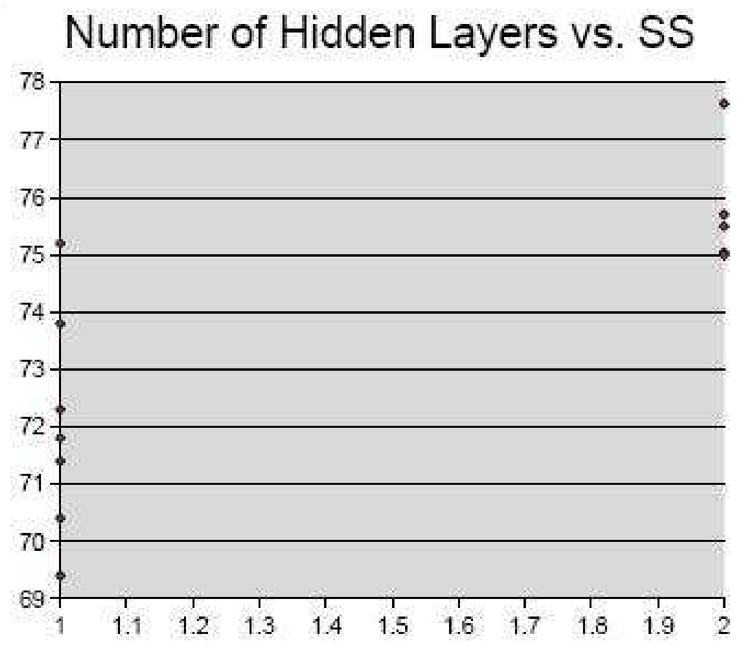

Figure 19: This figure shows how the number of hidden layers compare to each other, where the x-axis is the number of hidden layers and the y-axis is the signal significance. It appears that two hidden layers may be better at discriminating the signal vs. the background.

$\delta w_2\,(1,2) = 0$
$\delta w_2\,(2,2) = 0$
$\delta w_3\,(0,1) = 0.5 * 1 * -0.085459 = -0.042730$
$\delta w_3\,(1,1) = 0.5 * 0.584490 * -0.085459 = -0.042730$
$\delta w_3\,(2,1) = 0.5 * 0.471226 * -0.085459 = -0.042730$
After retraining several thousand times, the network gives values such as these:
Hidden node 1:
$w_2\,(0,1) = -6.062263$
$w_2\,(1,1) = -6.072185$
$w_2\,(2,1) = 2.454509$
Hidden node 2:
$w_2\,(0,2) = -4.893081$
$w_2\,(1,2) = -4.894898$

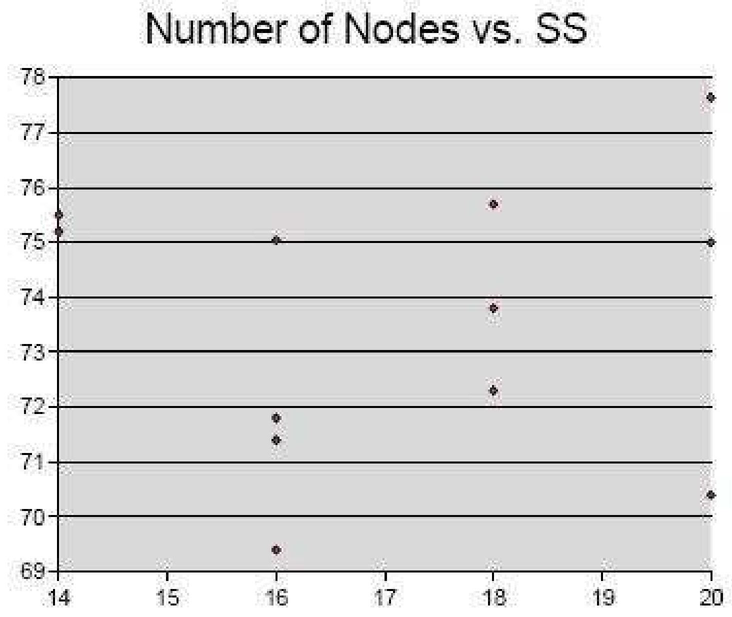

Figure 20: This figure shows how the number of nodes compare to each other, where the x-axis is the number of nodes and the y-axis is the signal significance. It appears that two hidden layers may be better at discriminating the signal vs. the background.

$w_2\,(2,2) = 7.293063$

Output node:

$w_3\,(0,1) = -9.792470$

$w_3\,(1,1) = 9.484580$

$w_3\,(2,1) = -4.473972$

With these outputs, you would get the following results for the "xor" logic function mentioned earlier.

0 XOR 0 = 0.017622 (fail)

0 XOR 1 = 0.981504 (pass)

1 XOR 0 = 0.981491 (pass)

1 XOR 1 = 0.022782 (fail)

7 Appendix 2: Input Variables

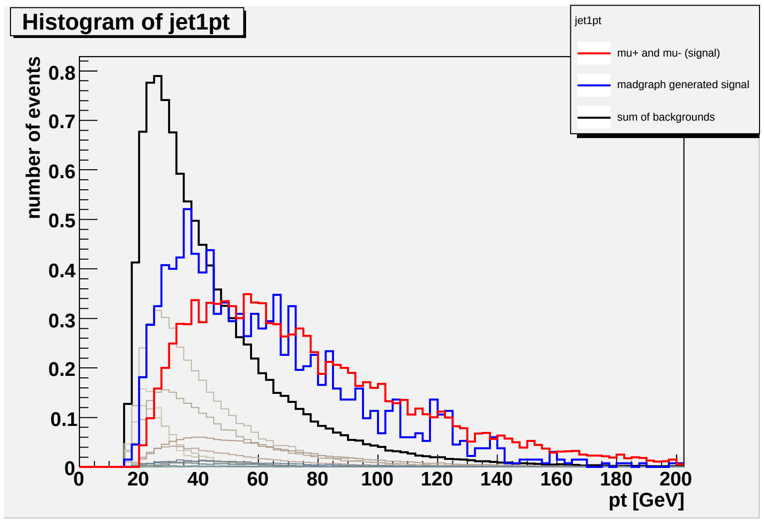

Figure 21: This figure shows the transverse momentum of the leading jet.

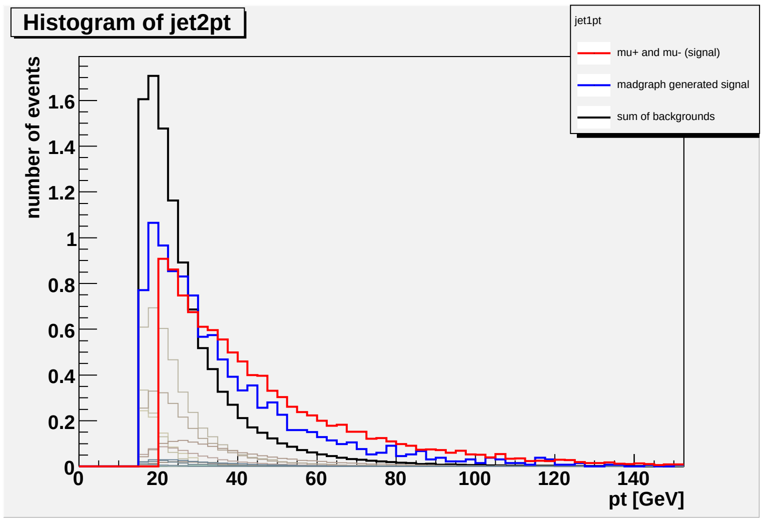

Figure 22: This figure shows the transverse momentum of the second-leading jet.

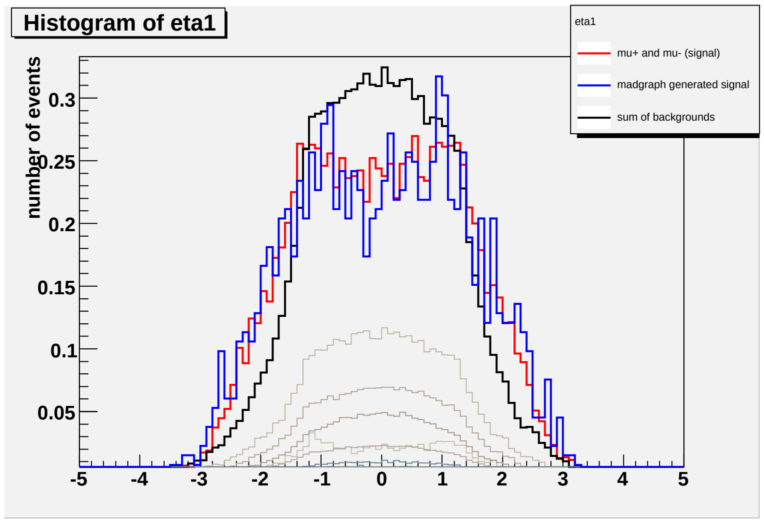

Figure 23: This figure shows the rapidity of the leading jet.

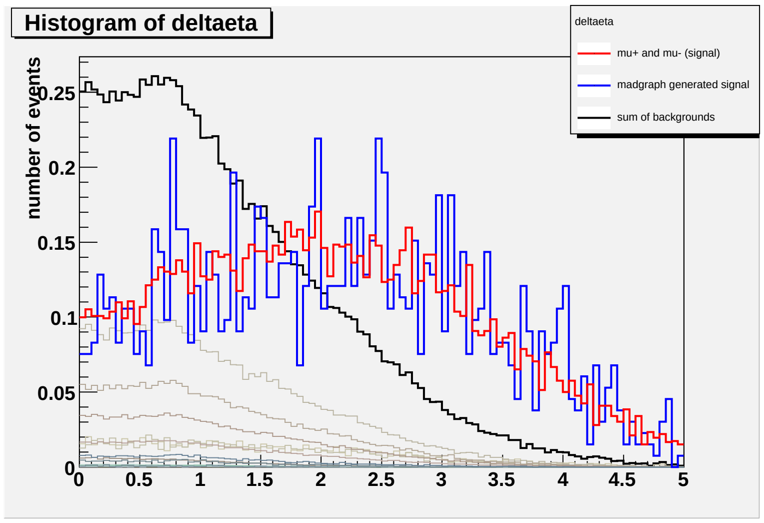

Figure 24: This figure shows the difference in rapidity between the leading two jets.

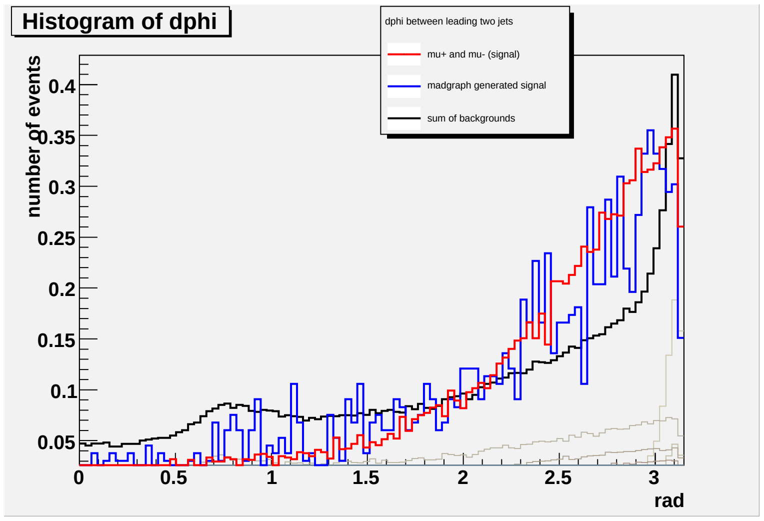

Figure 25: This figure shows the difference between azimuthal angles of the leading and second-leading jets.

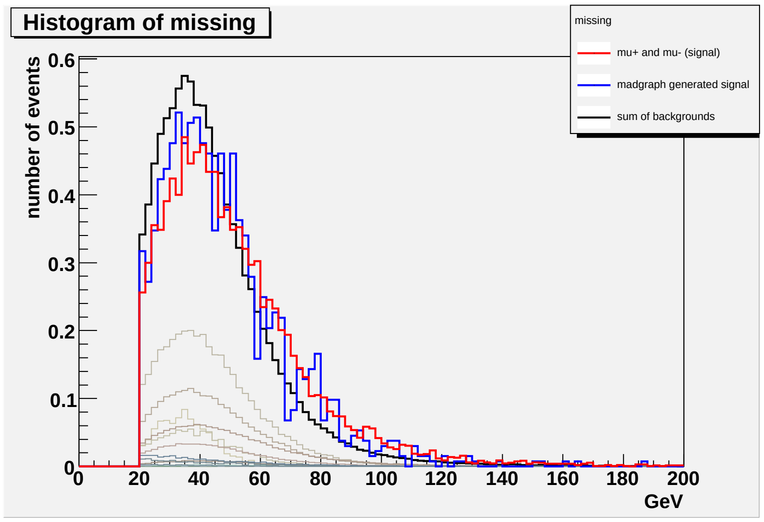

Figure 26: This figure shows the missing transverse energy.

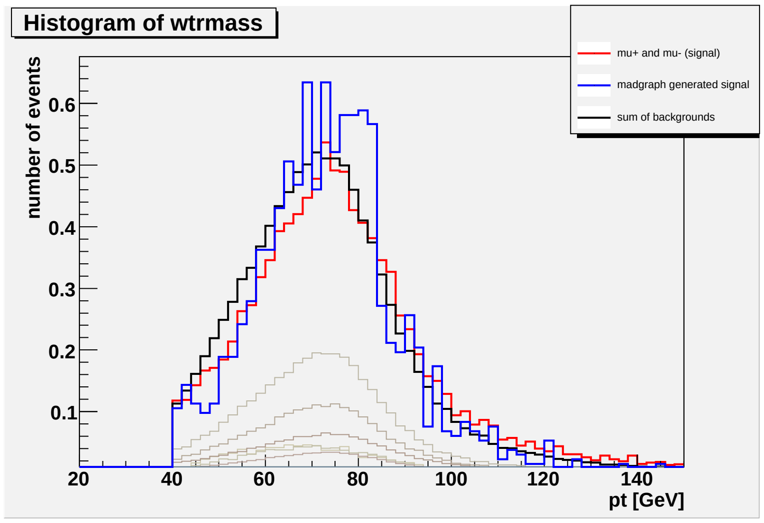

Figure 27: This figure shows the transverse mass of the W Boson.

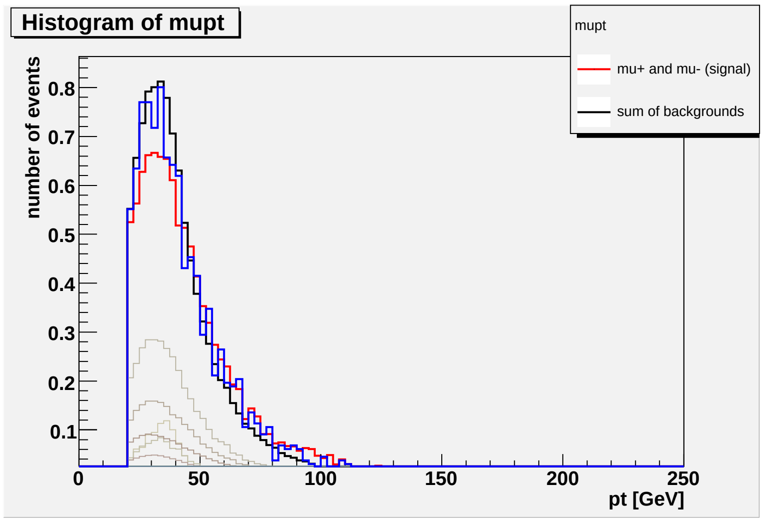

Figure 28: This figure shows the transverse momentum of the leading muon.

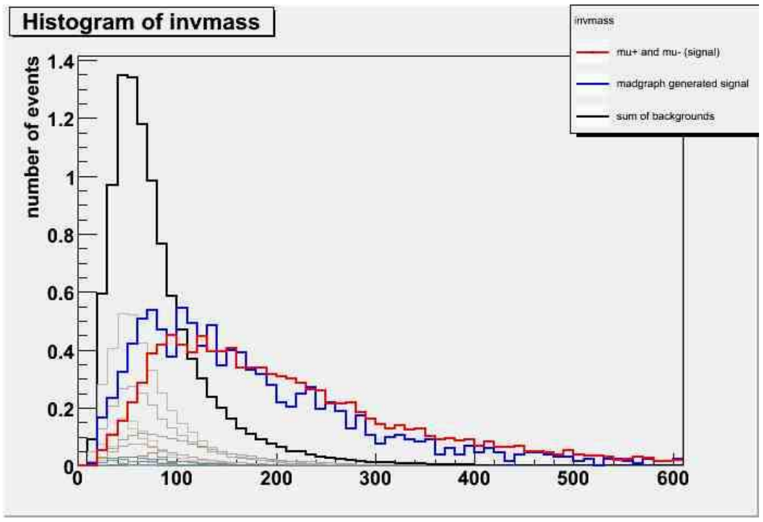

Figure 29: This figure shows the invariant mass of the leading jets.