Performance and training bias of BDT vs NN in ttH(bb) search at ATLAS

Liv Helen Våge, University College London
Supervisors: Dr. Paul Glaysher, Dr. Judith Katzy

September 6, 2018

Abstract

The Higgs-Top Yukawa coupling can be determined directly at the LHC by measuring the cross section of ttH(bb). This would confirm predictions of the Standard Model, and give insight to newer theories like whether the Higgs was responsible for inflation. ttbar plus b-jets has an equivalent signature to ttH(bb), so signal and background have to be separated. Boosted decision trees have been used to separate signal and background in the analysis so far. Different Monte Carlo simulations describe ttbar events differently, resulting in a large uncertainty. It possible that neural nets would perform better than the BDT, but it is hypothesised that they would result in a larger uncertainty. This paper explores this in TMVA and a TMVA function that integrates with Keras. There is no evidence that the neural nets are more biased than the BDT. Some neural nets defined in Keras slightly outperform the BDT, and Keras might be a promising option for further exploration.

Contents

1 Introduction

Six years after the discovery of the Higgs boson, its production and decay modes are still being explored to confirm predictions of the Standard Model. Higgs boson production in association with two top quarks (ttH) is particularly interesting as it involves the two heaviest known particles. Measurements of the ttH process at the Large Hadron Collider (LHC) directly probes the Higgs-Top Yukawa coupling since this is proportional to the square root of the cross-section. Higgs-Top coupling can give insight to whether the Higgs boson was responsible for inflation, or help find new phenomena by considering the stability of the electroweak vacuum [1]. Some people even claim that "at the present moment the only quantity which can help us get an idea about the scale of new physics is the top Yukawa coupling." [2]

ttH only accounts for 1% of Higgs boson production at the LHC, so a promising way to observe it is to look for the Higgs boson's main decay channel to two bottom quarks. Unfortunately, QCD production of ttbar and b-jets has the same signature, and is more frequent by three orders of magnitude [3]. An example of a signal and a background process is shown below. To separate signal from background, Monte Carlo (MC) simulations are used. There are different simulations depending on slightly different assumptions, e.g. how coloured final state partons radiate and combine to become hadrons. The difference between these simulations is taken as a systematic uncertainty, which is almost as large as the signal strength, posing the largest challenge in the ttH(bb) analysis at the time of writing.

Figure 1: Feynman diagram examples for a) ttH(bb) production and b) tt +bb background.

Since ttH(bb) is very rare, simply removing the background signal and considering cuts by manually eliminating events is insufficient. Machine learning (ML) is a powerful tool to classify signal and background. The ML algorithm must also be robust against bias - that is, it must not be very susceptible to picking up on features specific to the Monte Carlo simulation it is trained on, resulting in a large uncertainty. Boosted decision trees (BDT) were used in the results published in 2017 [3]. They are well suited to the problem as they are quick to train, and are fairly robust against bias. Neural nets (NN) are known to perform well on complex tasks, and might be a better option than the BDTs. This paper explores how well neural nets extract the signal as compared to the BDTs, and how biased they are.

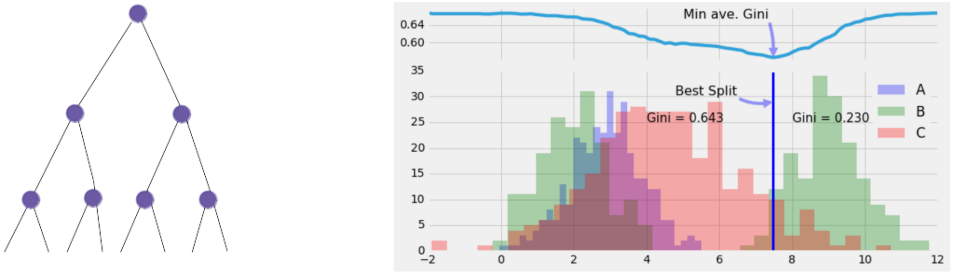
2 Machine learning

Supervised machine learning is machine learning where the categories are known - here, signal and background. Since we know which events are signal and background for the Monte Carlo simulations, we can supervised machine learning. In general, a supervised ML algorithm takes input arrays for m variables, and applies functions to them to sort them into the known categories. It compares the classification to the correct category using a loss function, and updates itself to minimise this function, thereby learning. A vast array of ML algorithms exists, but two of the most popular are BDTs and NNs. BDTs are known to perform well with both Monte Carlo and real data for ttH(bb), and is quite impervious to systematic overtraining or bias. It does not pick up patterns that arise from low statistcs or is specific to the MC simulation it is training on that are not present for all MC data. However, neural nets have been known to perform well on similar tasks [4], and it is possible that neural nets will outperform BDTs if optimised well. The main concern is that the NN would overtrain, so the increase in classification power would be outweighed

by an increase in uncertainty. The neural nets discussed in this paper were all implemented in Root's TMVA's Artificial Neural Net functionality, except those defined in Keras, discussed in section 6.

2.1 Boosted Decision Trees

A decision tree makes a decision at each node which grows into a full tree that ends in a classification. Most decision trees use the Gini Impurity Coefficient to make each decision. To get an intuitive understanding, consider Fig. 2. Each of the features input to the BDT will have a plot like 2b. The Gini Coefficient will be calculated for all x-values along this plot, roughly measuring the overlaps of each classification category. Consider the vertical blue line. To the right and left of this line, a measure of how many of A's events that fall within B's histogram, C within B and so on, will be calculated. This is combined to a weighted average, which is shown as the light blue line on top of the plot. The minimum of this function reflects the best point to make a cut. This is done for all variables, and the one with the minimum Gini coefficient is chosen as the node of the decision tree. This is repeated for each node in the tree.



(a) Decision tree

(b) Gini Impurity Index

Figure 2: A decision tree like a) is built by making decisions at each purple node by the method illustrated in b). (Figure b) adapted from [5]).

This process usually leads to a weak learner - a learner with poor classification power - since such a simple algorithm rarely can reflect the appropriate complexity of the input distributions. One way to remedy this is to combine weak learners, known as a boosted decision tree (BDT). This paper considers the AdaBoost method. Simply put, one decision tree is built. The events misclassified in this tree are then weighted more than the rest, and a new tree is built. This is repeated so that each tree corrects the mistakes of the previous trees. When the set number of trees have been built, they are all weighted by their classification power and combined into one strong learner. As shown in Fig. 2b, how the BDT makes cuts on the variables can be read, giving insight to the learning algorithm that may be useful for further analysis [6].

2.2 Neural Nets

Neural nets were made with the intention of emulating a biological brain. The same input as was fed to the BDT is given to the neural net. Each neuron will calculate a weighted sum of its inputs and add a constant. The neuron will apply an activation function to this value, essentially scaling the input. If the final value is above a certain threshold, the neuron will fire - much like a biological neuron firing if the electrical current is above a certain value. If the neuron does not fire, it will be ignored for that cycle (see section 4.2 for more info on cycles). The neurons connect to each other, and result in a classification, in this case as either signal or background. Like the BDT, this is compared to the actual value using a loss function. The weights and connections of the neural net are updated in cycles to minimise this loss.

This method has proven to be extremely effective on a wide range of tasks, and extensive research has gone into the justification of this. Neural nets are also known to be the most difficult ML

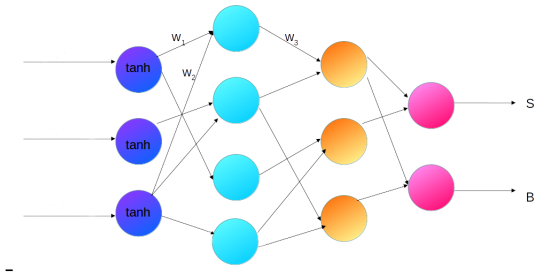


Figure 3: A simple neural net structure showing hyperbolic tangent as the activation function. Each arrow has an associated weight, as illustrated. The final layer consists of two nodes so that the output will be either signal or background.

algorithm to fine tune - just choosing different different loss functions can have a large impact on the classification power. It is also considered a notorious "black-box", the final model is difficult to interpret physically. [7]

2.3 Variable and data selection

The data from the Monte Carlo simulations are fully simulated MC events at reconstruction level, and must be pre-processed. Only events with 5 or more jets where 3 or more are b-tagged are considered here. The analysis is also restricted to a 70% b-tagging working point - corresponding to the fraction of b-jets kept in the analysis. Three techniques were used to prepare the variables, described in detail in [3]. 28 variables were then used as inputs to the ML algorithms, which can be found in 7. Optimisation of these were explored in [8]. Five different Monte Carlo simulations were considered for the background; a mixed sample, Sherpa, Powheg-Pythia8 (PP8), Powheg7 (PoH7) and aMC@NLO (aMC). All MLs discussed until section 5.2 were trained on a sample of arbitrarily mixed Sherpa and PP8. All MLs from section 5.2 onwards were trained on PP8.

3 Performance and bias metrics

In order to compare BDTs and NNs, some figures of merit have to be used. The ones considered here are calculated automatically when doing ML in TMVA.

3.1 Measuring bias

In this paper, the bias is measured qualitatively by considering overtraining plots, like Fig. 4. The ML algortihm classifies each event as signal or background with a certain probability. The number of events with this probability is counted for both signal and background and normalised. Positive numbers on the x-axis denote the probability of an event being a signal-like event, and negative denote the probability of being a background-like event. Note that for neural nets, this ranges from 0 to 1 rather than -1 to 1. Events that are truly signal events are shown in blue, and true background events are shown in red. A good classifier would therefore have most of the blue to the right of the plot, correctly classifying the signal with a high certainty, and the red to the left, correctly classifying the background with a high certainty.

Bias comes into play when there is a large difference between the data the ML is trained on and tested on. The test set is shown by the solid colour, and the training by the markers. If the model is overtrained, the training will differ from the test set. That is, the model trained on one MC sample is not good at classifying another MC sample, reflecting bias. The MC simulations for the signal are fairly uniform, so bias is only considered for the background. As will be shown in section 4, sudden spikes in the distribution may signify that the algorithm found a local optimum rather than a global. Hence, these plots can give information about classifying power, bias and learner convergence.

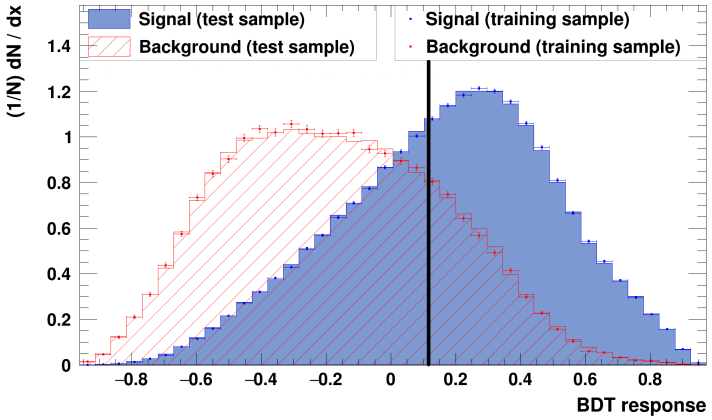


Figure 4: BDT response for BDT with depth 5, 400 branches with an arbitrary superposed line.

3.2 Integral under the ROC curve

To measure performance quantitatively, the integral under the receiver operating characteristics (ROC) curve can be used. Any classifier will correctly identify some signal events, known as a a true positive, and fail to classify others - a false negative. It will also correctly classify some background events - a true negative, and incorrectly classify others as signal - a false positive. The ROC curve is a way to measure how many events fall into these categories. Consider the arbitrary vertical line superposed on Fig. 4. The number of signal events to the right is divided by the total number of actual signal events- i.e. the true positives by the signal events. This is the signal efficiency. The same is done for the background - the false positives against all actual background events, corresponding to the background rejection. This will be one point on the ROC curve. This is repeated for all possible positions of the vertical line. The better the classifier, the closer the ROC curve will get to the top right corner. A random classifier is shown in red. Thus, classification power is largely reflected by the integral under the ROC curve [9].

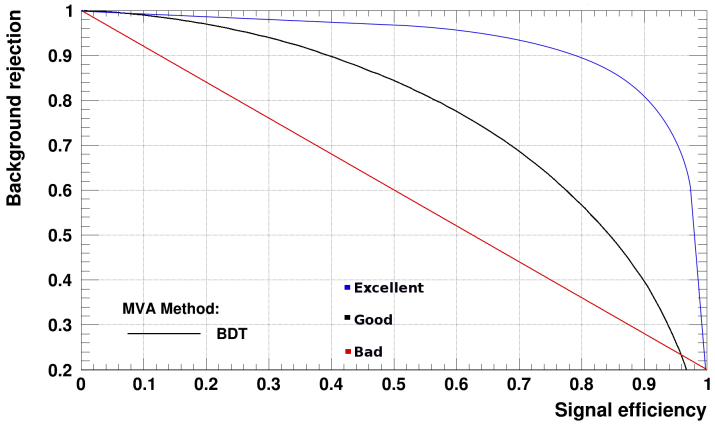


Figure 5: ROC curve for a BDT of depth 5, 400 branches shown in black. Red and blue represent supposed bad and excellent classifiers, respectively.

4 Achieving NN performance comparable with the BDT

The first step to compare BDT and NN bias is to achieve a similar performance. About 100 different NN architectures were tested. To reduce training time, 20 of the NNs tested only used the 14 best variables as determined by the largest variable separation. This was disregarded as the integral under the ROC curve was consistently lower and the error bars in the overtraining plots were larger. There were 5 main parameters to explore in TMVA; sampling, number of cycles, number of neurons and layers, activation functions, and the use of a regulator. The hyperparameter

that decides how to update model parameters is called the loss function, and was set so BFGS, as recommended by TMVA [10]. The learning rate - the hyperparameter deciding how much weights are changed with respect to the loss function - was set to 0.007. This was only determined by a some quick tests, and could very likely be better optimised. The NNs did not reach the same ROC value as the BDT, and they generally took longer to train.

4.1 Sampling

In TMVA, a sample value can be set so that the NN only trains on a fraction of the data. If e.g. 0.6 is set, TMVA will take a random sample at each cycle so that the total fraction of data used through all cycles is 0.6. The main advantage is to reduce training time, as sampling is roughly proportional to training time. A sampling importance can also be set between 0 and 1. If it is less than one, the probability that an event will be selected again depends on the error function. If an event reduces the loss function, the probability that the event will be selected again is multiplied with the sampling importance, i.e. the NN is well trained for that type of events, and should focus on other events. If the event increases the error function, the NN should consider that event more closely, and the probability that it will be selected again is divided by the sampling importance [10]. The number of training events was around 360 000 for the signal and 160 000 for the background. The NN performed well with low sampling fractions such as 0.01, but the uncertainties on the response plots decreased as sampling increased. The sampling also affected the shapes of the response distributions. The sampling importance did not seem to have any effect.

The overall effect of sampling on performance is illustrated below. Even with a sampling below 0.1, a ROC integral value above 0.74 can be reached, provided the NN architecture is good. The vertical lines demonstrate that the other factors explored in this section also affect performance.

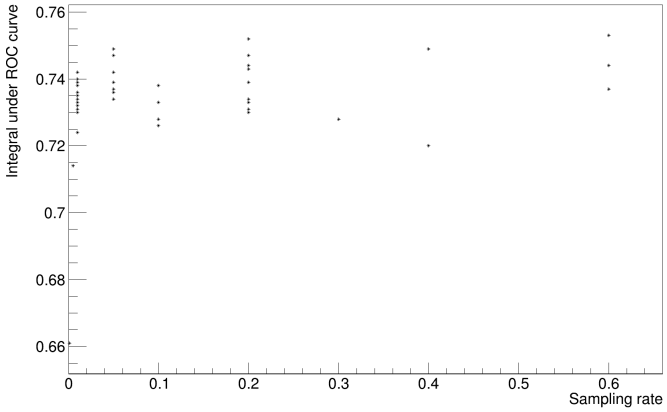


Figure 6: Integral under the ROC curve vs. sampling fractions for various NN architectures - each point represents a different NN architecture

Guided solely by the above plot, one might think a low sampling is sufficient to achieve a good model. The overtraining plots give a deeper insight; let us consider a very simple NN as shown below.

An increased sampling reduces the error bars of the response drastically, but also changes the shape of the distributions. However, the effect is a bit different when considering a more complex network, as shown below.

From the above plots, one can see that the ROC integral is very similar, as is the shape of the distributions. The only difference seems to be that the error bars are smaller on the plot with no sampling. The NN with sampling took 1 hour 40 min to train, the other took 4 hours. There was extensive evidence that once the NNs grew complex enough to run for more than an hour, most architectures had a similar overtraining plot and ROC integral, as is also shown in Fig. 12. If the plots had any spikes or a shape otherwise dissimilar to Fig. 10, these trends disappeared as the training time grew due to either an increased number of sampling, cycles, or layers/neurons. This

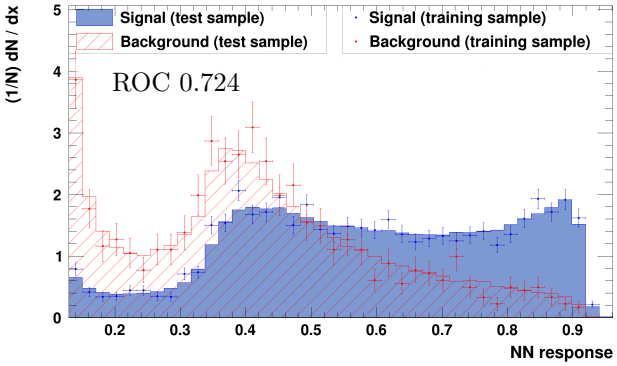


Figure 7: Response of NN with 3 neurons, 50 cycles, sigmoid activation function and sampling of 0.01

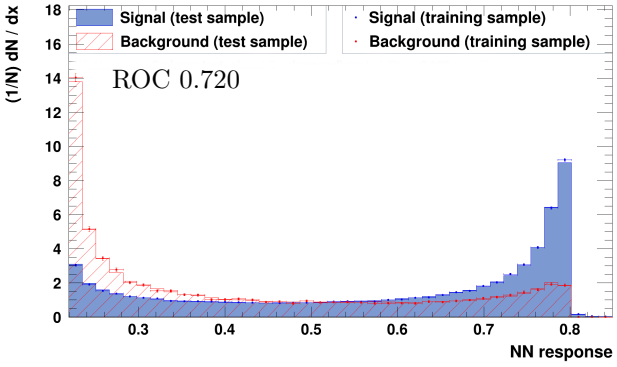


Figure 8: Response of NN with 3 neurons, 50 cycles, sigmoid activation function and sampling of 0.4

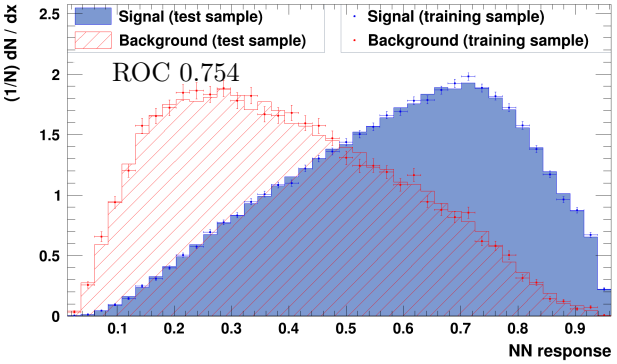


Figure 9: Overtraining plot for NN with structure 5-5-3-2, 800 cycles, sigmoid activation function and sampling of 0.2

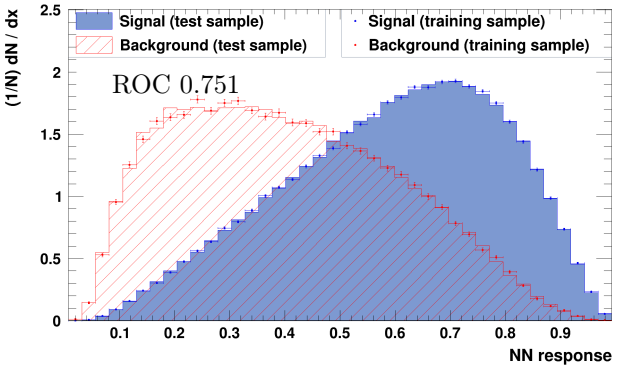


Figure 10: Overtraining plot for NN with structure 5-5-3-2, 800 cycles, sigmoid activation function, no sampling

was also true for Fig. 8. Using sampling might therefore be a good way to reduce training time, provided the NN has enough cycles or layers/neurons to compensate.

4.2 Number of cycles

Increasing the number of cycles did not seem to lower the error bars, as sampling did. In fact, the number seemed to have little effect on the distribution and ROC integral value for both complex and simple networks, provided the number of cycles was above around 200. Above this, the ROC integral would typically increase by 0.02 for a doubling of cycles, which may be within the uncertainty of the ROC integral value.

4.3 Number of neurons and layers

There was no clear correlation between number of neurons and/or layers and performance, as illustrated in Fig. 11. This also held true for a higher sampling than shown in this figure.

Fig. 11 shows that increasing the number of layers or number of neurons in each layer, does not necessitate a higher performance or lower errors on the NN score. The top right plot suggests that deeper neural nets need more statistic to achieve a good performance, i.e. either a higher sampling or more cycles. This is quite intuitive, since a deeper NN has more parameters to learn. The same NN was tried with a sampling of 0.6, which gave a similar shape to the other three plots, and a ROC integral of 0.757. There is also extensive literature on how a single layered NN can perform

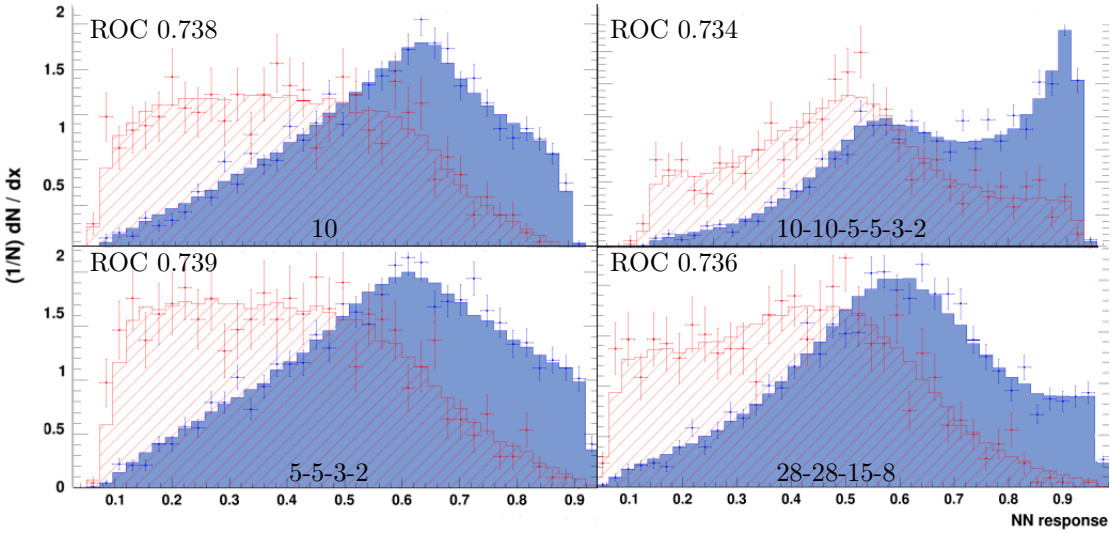


Figure 11: Different NN architectures for sampling 0.01, 400 cycles, sigmoid activation function. The hyphenated numbers denote the number of neurons in each layer.

as well as a multilayered NN e.g. as presented in [11] and [12]. However, as [12] states "Empirical work, however, shows that it is difficult to train shallow nets to be as accurate as deep nets.". A well optimised shallow learner can often do as well as a deeper learner, but may need more optimisation. Since a deep learner has more parameters than a shallow learner, one furthermore expects the bias to be higher for a deep learner. Therefore, both deep and shallow NNs were chosen to compare bias, as outlined in section 5.2.

4.4 Activation functions

Unless one has set up multicore CPU or GPU training in TMVA, a single activation function is defined for the whole NN, and the only two commonly used functions available are sigmoid and hyberbolic tangent. This is a limitation, as e.g. the very popular function ReLu, which only considers input above 0, has been shown to converge six times faster than tanh for certain tasks [13]. Of the two available functions, sigmoid generally created a uniform peak for signal and background, such as shown in Fig. 10. Tanh tended to have sharp peaks, similar to the top right of Fig. 11. However, a tanh NN that ran for 4 hours converged on the same shape as sigmoid. Is is therefore hypothesised that sigmoid converges faster than tanh for this task. One should generally be careful when using sigmoid, as this is known to saturate, which may impair learning. As an output function, sigmoid is also known to be inferior to softmax, which not only squashed the input to between 0 and 1, but also normalises the sum of all events to 1. TMVA does not have softmax, which is yet another limitation [14]. Therefore, sigmoid is considered the best option in this case, but is very likely not the best option if using any other setup, including TMVAs deep neural net functions. This was later confirmed using Keras, see section 6.1

4.5 Regulator

The regulator in TMVA adds a term to the error function, effectively penalising large NN weights, and thereby reducing model complexity. For simple NNs, the regulator seemed to smooth the overtraining distributions slightly. For neural nets running for an hour or more, it had no noticeable effect, and was therefore not used for further analysis.

4.6 Overall performance

None of the neural nets tested in TMVA's Artificial Neural Net functionality performed as well as the BDT. To achieve similar distributions and error bars in the BDT overtraining plot, several hours of training were needed for the neural nets. Some architectures seemed to converge on this shape faster than others - a structure of 5-5-3-2 seemed to do particularly well, even with low sampling. However, increasing the sampling decreased the error bars, as discussed in section 4.1. Three different architectures that all performed similarly were chosen to proceed with the comparison, which is discussed further in section 5.2.

5 Bias and performance comparison for NN and BDT

Based on the exploration outlined in section 4, three neural nets with different architectures were chosen to compare with the BDT. The term performance here refers mainly to the integral under the ROC curve, but other parameters like training time and response uncertainty is also considered.

5.1 Performance

Three of the highest performing NNs are shown next to the BDT below. The different architectures were chosen to test whether bias increased with NN depth.

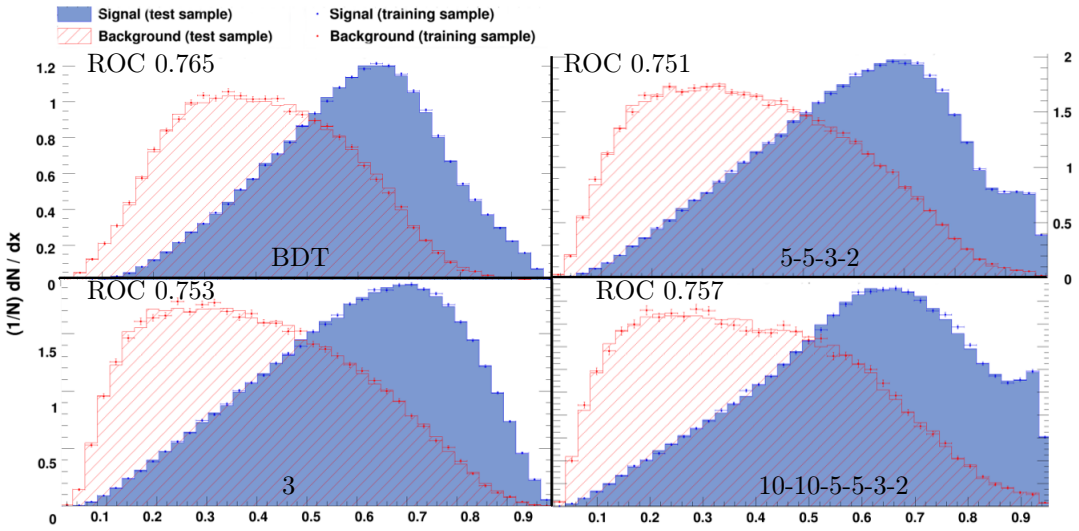


Figure 12: The BDT (upper left) and the three neural nets chosen for further analysis. All had sampling 1 except the deepest, which had sampling 0.6. Number of cycles was 400 except the top right, for which it was 800.

Even though the NNs have very different architectures, they all have similar response distributions. Even the NN to the bottom right, which has a sampling of 0.6, achieves a similar shape as the others. It is possible that once the NNs reach a certain complexity - reflected either in depth, neuron numbers, number of training events or otherwise - they converge on a similar optimum.

5.2 Bias

All MLs discussed up to this point were trained on a background sample consisting of an arbitrary mix of Sherpa and PP8. From now on, the discussion will involve MLs trained on PP8, and tested on various Monte Carlo samples. 4 different testing samples were used for the background; PP8, Sherpa, POH7, and aMC. There was no evidence that the neural nets were more biased than the

BDT for any of these. The generator with lowest statistics that is also known to be the most different from the other MC samples is aMC. The overtraining plot for this generator is shown for the BDT and the deepest NN in Fig. 13

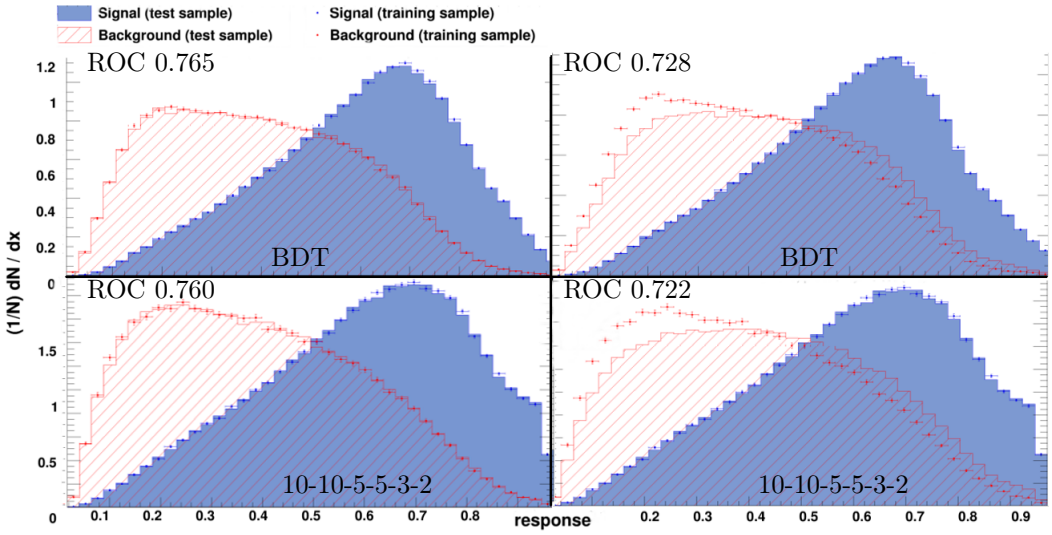


Figure 13: The BDT (upper row) and the deepest neural net (lower row) trained and tested on PP8 (left) and trained on PP8 and tested on aMC (right).

The drop in the ROC integral value as the algorithm is tested on aMC instead of PP8 is almost identical between the NN and the BDT, as is the difference in test and training points by visual inspection. The drop in ROC integral value occurs because the events are classified as more signal-like, reducing the ability to separate signal and background. The two other MC samples did not display this trend as strongly, and there was no evidence of higher bias of the NN than the BDT for any of the NNs showed here.

6 Keras

Keras is a high level neural net package written in Python that can run on top of e.g. Tensorflow or Theano. As discussed in section 4, TMVA has several limitations. Keras, however, is a state of the art package. TMVA is generally preferred since data for these types of analysis are usually handled in Root due to their mere size. It can be a tedious task to convert these to a Python-friendly format. Even then, after a NN is trained, it can be challenging to integrate it to C++ code. Luckily, TMVA integrates with Keras, so that root files can be imported to a Python script using TMVA commands. The NN is then defined in Keras, and the prediction works very similarly to TMVA, giving identical output analysis options. Many advanced Keras options are not available in TMVA using this method, but the method can still provide the best of root's data handling and Keras' freedom of choice in hyperparameters.

About 50 different neural nets were tried in Keras by varying similar parameters to those presented in section 4. Most of these converged on a very similar response distribution, often with a ROC value higher than the BDT. Keras allows batch training (see section 6.2), which TMVA does not, unless one has set up multiple CPU or GPU training. Therefore, the Keras NNs only took a few minutes to train, even with hundreds of neurons. A slight drawback is that applying the model after training and testing is a bit more complicated than TMVA. A prediction function has to be applied to each event, and the result is appended to an array. As a reference point, this took about an hour for about one million events. Considering that the model would mostly be applied on real data and not Monte Carlo data, this is not a major drawback. The plots in this section were made using matplotlib.

6.1 Activation functions

As argued in section 4.4, it is likely that sigmoid is not the best activation function. Using a layered nerual net of 28-50-40-15-2 neurons, this hypothesis was tested. The first layer of this architecture equals the number of input variables, and expands into a pyramid shape, which is commonly done [7]. This was not tested in TMVA as it would have taken very long to train without minibatch training. One neural net was then made using only sigmoid neurons, another with ReLu neurons and softmax as the output layer. Both took just as long to train, but the sigmoid NN achieved a ROC integral of 0.742 and the other achieved 0.779. It is therefore possible that the nerual nets in TMVA could have achieved a performance similar to the BDTs if appropriate activation functions were available. This poses an argument against using TMVA's Artificial Neural Net package for this and similar tasks. Relu is available in TMVA's Deep Neural Net package, but softmax is not. The NN described earlier was trained with only ReLu neurons, which resulted in a ROC integral of 0.757, showing that using softmax as the output neuron makes a noticeable difference. It is possible that the results would have been different in TMVA since the training methods are quite different from Keras. However, the shape of the sigmoid NN's overtraining plot was very similar to the well performing NNs defined in 4, suggesting that the activation functions were indeed a big limitation.

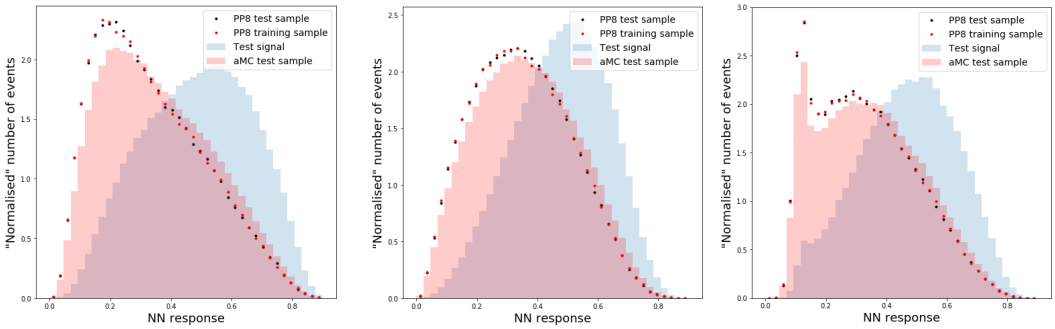
6.2 Number of epochs and minibatches

The number of epochs is the number of times the algorithm passes through the whole NN. Mini batch training is not available in TMVA's Artificial Neural Net package, but is available in the Deep Neural Net package. Minibatch training takes a random sample from the training set and updates the loss function based on the average loss of this sample. This avoids the very computationally expensive method of evaluating each training event separately. This is often referred to as batch training, but should not be confused with full batch training, where all samples are considered as the loss function is updated. An attempt was made to replicate the results from section 5.1 in Keras. Training took more than 14 hours and crashed, suggesting that setting the batch size to 1 and assuming an equivalence between TMVA's cycles and Keras' epochs might not be valid. The batch training, however, is very quick and offers more exploration of the effect of NN depth and neuron number than could be explored in TMVA. A common method is to start with a minibatch size of 32 and tweak it [15]. The number of epochs is very easy to tweak as the loss and validation accuracy can be output as the NN is training. In Keras, TMVA also allows early stopping - a number of epochs n can be set so that the NN stops training if the loss has not decreased in that last n epochs. A batch number of around 32 seemed to perform well for this task, but this was not finely optimised. NNs with a batch of 500 converged to a similar distribution as smaller batches, but generally with a lower performance. Most NNs converged in under 40 epochs.

6.3 Number of neurons and layers

Since Keras allows batch training, the depth of the NN and the number of neurons in each layer can be increased drastically from what was attempted in section 4. It is of particular interest to see whether a dramatic increase in depth or number of neurons increases bias, as hypothesised. The computer scientist Prof. Bengio suggests that an excessive amount of neurons usually has no negative impact on prediction power, so a few potentially excessive NNs were tested [15]. He also claims that an overcomplete first hidden layer (number of neurons exceeds number of input variables) usually performs better than an undercomplete NN. These hypotheses are explored in Fig. 14.

The deep neural net in Fig. 14c performs better than the other two, and perhaps surprisingly, is also less biased. It is possible that this NN had enough parameters to find a good minimum of the loss function that the others could not. The NN with an excessive amount of neurons seems to possibly have overtrained, resulting in a large bias. Considering that the award winning GoogLeNet only had 22 layers to classify more than 1000 categories, there is perhaps no need to go any deeper than 15 layers. This task is very different, but it is clear that a depth similar to



(a) NN with 28-50-40-15-2 neurons

(b) NN with 14 layers with 50 neurons each

(c) NN with 100-100-200-200-300-300-200-100-2 neurons

Figure 14: Various complexities for NNs with ReLU activation functions and output layer of two neurons and softmax activation function. All were trained with minibatch of 500 over 40 epochs. The integral under the ROC curve was 0.758, 0.765 and 0.760 respectively.

that of Fig. 14b is sufficient for very complex tasks. Hence a depth around 15 and a considerably lower neuron number that in Fig. 14c is a good starting point for further exploration.

6.4 Dropout

One of the very appealing points of using Keras is that it enables dropout - a mechanism designed specifically to address overtraining. This can be done in TMVA as well, but only when multiple CPU or GPU training has been setup. Dropout disregards a set fraction of the neurons in each batch, creating an ensemble of learners that contain all possible combinations of dropout neurons. It is common to define the dropout as the first hidden layer and add another dropout layer before the output layer. This was done for the same NN as in Fig. 14a, where the first dropout layer had a fraction of 0.2, and the second had 0.5.

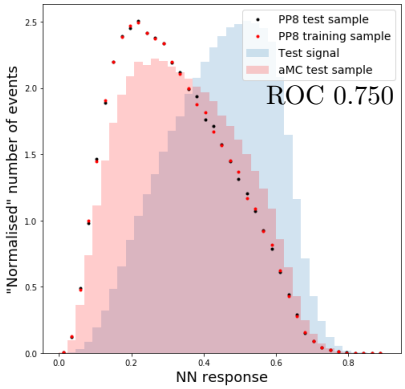


Figure 15: NN with a structure of 28-Dropout0.2-50-40-15-Dropout0.5-2, minibatch 500, 40 epochs

Comparing this to 14a, there is very little difference between the two, except that dropout has reduced the performance. It is possible that dropout would have been better with a deeper neural net and lower fractions.

6.5 Learning with momentum

Momentum can be set when doing a stochastic gradient descent, which was the optimization method used for the plots earlier in this chapter. The momentum adds a short term memory to the loss function, so that if the loss is great, the descent is steeper. This can cause a quicker convergence and may prevent the learning algorithm from getting stuck in local minima [7]. A momentum of 0.2 was set for the plot shown below.

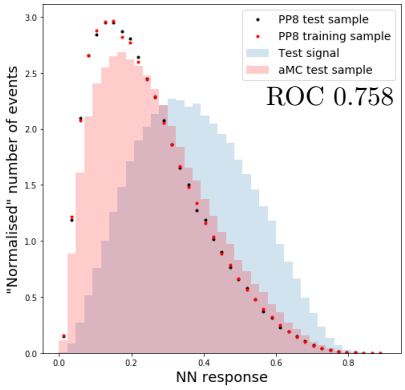


Figure 16: NN with 28-50-40-30-15-2 neurons and gradient descent momentum of 0.2

There was no change in prediction power, and based on a visual inspection, the bias seems comparable to that of 14a. However, it decreased the training time from 54 minutes to 3 minutes. This can therefore be a good solution if stochastic gradient descent is used. Another way to reduce training time is using another optimizer, e.g. Adam, as explored in the next section.

6.6 Optimization functions

In TMVA, the optimization function was set to the Broyden-Fletcher-Goldfarb-Shannon (BFGS) method, which is a slight alteration of standard back propagation. The TMVA guide suggests this as the best optimiser [10]. In Keras, other alternatives such as Adam and Adagrad are available. With Adam, each weight has its own learning rate, which is updated as the training progresses. This was introduced in 2014 and has proven quite successful on a range of tasks [16].

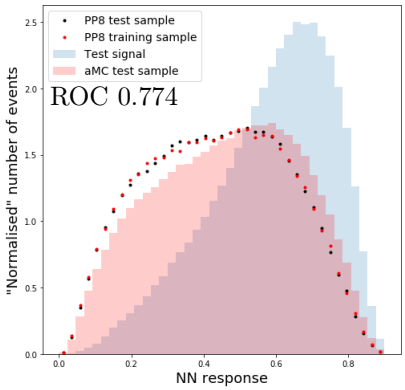


Figure 17: NN with 28-50-40-30-15-2 and the optimizer Adam

This NN outperforms any other NN tested, and also halves the training time as compared to stochastic gradient descent. The bias is also lower, as compared to Fig. 14. Adam might indeed be a good option, but this also suggests that the learning rate, which was not optimised either for TMVA or Keras, has a large impact on the NN. This should be the first step of a further exploration.

7 Conclusion

None of the neural nets defined in TMVA achieved the same ROC integral as the BDT, the closest obtaining 0.757 as compared to the BDT's 0.765. There was no evidence that the NNs in TMVA were more biased than the BDTs. Some of the NNs defined in Keras exceeded the BDT's ROC integral, the highest achieving 0.774. Several of the NNs in Keras were more biased than the

BDT, but this did not seem to increase with the depth of the NNs. The hyperparameters of the NNs affected performance and bias extensively for both TMVA and Keras. In TMVA, most NNs converged on a similar distribution of signal and background when trained for more than an hour. In Keras, no NNs were trained for that long. With only a few minutes of training, Keras NNs obtained a better performance than TMVA did with hours of training. Several of Keras' options that are not available in TMVA impacted performance, including advanced activation functions and optimizers. Neural nets might be a better option than BDTs when continuing the ttH(bb) analysis, as the performance was slightly exceeded through this brief analysis, and there is no evidence of increased bias. Keras is a promising option, but needs fine-tuning. Adding dropout layers in particular did not seem to reduce the bias, as hypothesised. There is especially evidence that the learning rate should be optimised further, both in Keras and TMVA. Considering how Keras learns quickly using batch training, and TMVA learns slowly by considering all events separately, it could be interesting to combine several learners from each of these to an ensemble, mimicking the BDT mechanism.

Acknowledgements

Many warm thanks to Dr. Paul Glaysher and Dr. Judith Katzy for all the time and effort devoted to this project - their help has been invaluable. A special thanks to the ATLAS group for being extremely welcoming to all the summer students, and a big thanks to the lecturers and organisers of the summer student programme - everything has been smoothly organised, highly educational and above all very fun.

Appendices

Variables used for machine learning

A Variable List for BDT Training

Variable Name	Variable Description
nHiggs30_70	Number of b-jet pairs with invariant mass within 30 GeV of the Higgs boson mass
nJets_Pt40	Number of jets with $p_{\mathrm{T}} \geq 40$ GeV
pT_jet5	p_{T} of fifth leading jet
HT_jets	Scalar sum of jet p_{T}
HT_all	Scalar sum of all p_{T}
H1_all	Second Fox-Wolfram moment computed using all jets and the lepton
dEtajj_MaxdEta	Maximum $\Delta\eta$ between any two jets
Centrality_all	Scalar sum of the p_{T} divided by the sum of E for all jets and the lepton
dRbb_avg_70	Average ΔR for all b-tagged jets
Mbb_MindR_70	Invariant mass of the combination of any two b-jets with the smallest ΔR
Mbj_MaxPt_70	Invariant mass of the combination of jet and b-jet with the largest vector sum p_{T}
dRbb_MaxPt_70	ΔR between the two b-jets with the largest vector sum p_{T}
dRbb_HiggsMass_70	ΔR between b-jets from the Higgs candidate
dRlepbb_MindR_70	ΔR between the lepton and the combination of the two b-tagged jets with the smallest ΔR
Aplanarity_jets	$1.5\lambda_2$, where λ_2 is the second eigenvalue of the momentum tensor built with all jets
Mjj_MindR	Invariant mass of the combination of any two jets with the smallest ΔR
dRbj_Wmass_70	ΔR between a b-jet and any other jet from the W boson candidate
Mbj_Wmass_70	Invariant mass of a b-jet and any other jet from the W boson candidate
Mbj_MindR_70	Mass of the combination of any jet and b-jet with the smallest ΔR
dRlj_MindR	ΔR between any jet and a b-jet with the smallest ΔR
pT_jet3	Transverse momentum of the jet with the third largest transverse momentum
dRbb_MaxM_70	ΔR between the two b-jets with the largest invariant mass
AH4_all	5th Fox-Wolfram moment computed using all jets and charged leptons
Aplanarity_bjets_70	As Aplanarity, for b-jets
Mjjj_MaxPt	Invariant mass of three jets with largest vector sum p_{T}
Mbb_MaxM_70	Largest invariant mass of the combination of any two b-jets
Mjj_MinM	Smallest invariant mass of the combination of any two jets
dEtabb_Avg_70	Average $\Delta\eta$ for all b-jets

Table 1: List of variables used in the BDT. All b-tagged variables correspond to a b-tagging at 70% efficiency.

References

[1] V. Branchina, E. Messina, and A. Platania, "Top mass determination, higgs inflation, and vacuum stability," *Journal of High Energy Physics*, vol. 2014, sep 2014.

[2] F. Bezrukov and M. Shaposhnikov, "Why should we care about the top quark yukawa coupling?," *Journal of Experimental and Theoretical Physics*, vol. 120, no. 3, pp. 335–343, 2015.

[3] M. Aaboud *et al.*, "Search for the standard model Higgs boson produced in association with top quarks and decaying into a $b\bar{b}$ pair in pp collisions at $\sqrt{s} = 13$ TeV with the ATLAS detector," *Phys. Rev.*, vol. D97, no. 7, p. 072016, 2018.

[4] P. Baldi, P. Sadowski, and D. Whiteson, "Searching for exotic particles in high-energy physics with deep learning," *Nature Communications*, vol. 5, jul 2014.

[5] R. Duane, 2018. [Online; accessed August 31, 2018].

[6] K. P. Murphy, *Machine Learning : A Probabilistic Perspective*.

[7] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[8] S. An, "Optimising variable selection for machine learning analysis in atlas tth search." [Online] Available: https://www.dropbox.com/s/jg539lai09a3xim/VariableSelection.pdf?dl=0 [Accessed Aug 15, 2018], September 2017.

[9] J. A. Hanley and B. J. McNeil, "The meaning and use of the area under a receiver operating characteristic (ROC) curve.," *Radiology*, vol. 143, pp. 29–36, apr 1982.

[10] P. S. A. Hoecker *et al.*, "Tmva - toolkit for multivariate data analysis," 2007.

[11] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals and Systems*, vol. 2, pp. 303–314, Dec 1989.

[12] J. Ba and R. Caruana, "Do deep nets really need to be deep?," in *Advances in Neural Information Processing Systems 27* (Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, eds.), pp. 2654–2662, Curran Associates, Inc., 2014.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, pp. 84–90, May 2017.

[14] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, 2010.

[15] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," 2012.

[16] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.