Physics Senior Honors Thesis

University of Notre Dame

CMS Collaboration

Identifying Top Quarks Decaying Hadronically with a Deep Neural Network

Author:
Matthew Drnevich

Advisor:
Kevin Lannon

May 1, 2019

Abstract

The goal of the CMS experiment is to provide evidence for new physics by identifying extremely rare proton collisions which show signs of new physics from among the overwhelming background of uninteresting ones. One particularly interesting class of collisions are those that involve top quarks, the most massive particle currently known. In order to better classify particle collisions involving top quarks, we use machine learning models to label triplets of quarks with a score representing the possibility of resulting from top quark decay. Boosted decision trees, one approach to machine learning, using engineered features are the current models in use for this purpose, but here we show that artificial neural networks with only simple features can achieve a relative improvement in performance of approximately 7%. Furthermore, there are indications that the neural network is learning different characteristics than the decision tree, and further research could perhaps extract this additional information and give us insights into new important features in the particle collisions.

1 Particle Physics

As particle physicists, we are interested in the most fundamental questions about nature. We seek to understand the smallest fibers that weave the fabric of the universe. Through studying the fundamental constituents we can understand how everything else is built from them and potentially alter our perception of the universe. The only known way to effectively study particles roughly a billionth the size of an atom is to hit things together at incredible energies and see what they produce. Currently, the most powerful device for this purpose is the Large Hadron Collider managed by CERN in Geneva, Switzerland. At the Large Hadron Collider, we smash together protons that are each moving at nearly the speed of light to probe the fabric of the universe. After each collision a multitude of particles are produced that subsequently decay repeatedly until they reach a stable state. Once these stable state particles are measured by the surrounding detector, it is up to the data analysis to decipher the experimental data and find any possible new discoveries. These collisions that we are seeking often occur on the order of once in every billion collisions. For the sake of time, running costs, and making scientific conclusions, it is important that every such rare occurrence is identified in the analysis.

1.1 Analysis at the LHC

In particle physics analysis, the goal is often to take the physical information obtained from a collision's final state particles interacting with the detector and deduce the probabilities that certain classes of decay processes produced those final states. In particular, what particles were produced in the collision can provide evidence for the existence of new particles, as was the case of the Higgs boson observed in 2012 [1]. Figuring out this puzzle is incredibly difficult because of the probabilistic nature of physics on such small scales as well as physical limitations in hardware for measuring these final state particles. Any final resultant state of particles is likely the signature of many different decay processes, so taking one collision and labeling its exact process is not as trivial as making some calculations. Rather, it is only possible to classify the collision with a certain probability and confidence level. Therefore, the more information that we collect about a certain collision the better that we can effectively label its classification probabilities. This data often includes physical data collected from the detector, features calculated through known physics equations, and "engineered features" that are (sometimes arbitrary) calculated quantities which seem to have good discrimination power between collision classes. Since there is no closed form analytic expression, doing any calculation of these probabilities by hand would be impossible. Hence it is best for researchers to use machine learning techniques to perform these classifications.

1.2 Top Tagging

Rather than directly trying to classify an entire collision, here we focus on "top tagging". When a top quark is produced in a collision it is unstable with a mean lifetime of about 5×10^{-25} s. When it decays it almost always, about 99.8% of the time, produces a W boson and bottom quark [see 2, Top Quark]. The W boson is also unstable and will either decay into a lepton and its corresponding neutrino or a pair of hadrons. If it decays into hadrons, because of the nature of the strong force, the quarks will produce more pairs of quarks, resulting in many hadrons travelling in roughly a cone together as they hit the detector. When such a cluster of particles is measured in the detector the entire cluster is identified as a "jet". Top tagging is applied to triplets of jets in the final state of collisions to identify triplets arising from the final decay products of a top quark, that is, the bottom quark and two W boson jets. The ability to label a triplet of jets as the children of a top quark can act as a new feature that allows another technique to properly classify the collision better. Tackling these smaller problems within the overall goal tends to be more feasible and simplifies the final analysis. Our research focused on solving this problem of top tagging using two standard techniques in the field of machine learning, boosted decision trees (BDTs) and artificial neural networks (ANNs).

2 Machine Learning

As humans, we are often confronted with observations in nature that follow relationships for which we cannot write down an explicit relationship in an equation or algorithm. Although we may not be able to develop a closed theory, we can still collect data to help us understand the relationship better, and then perhaps discover a descriptive mathematical model. In elementary situations, this could be something as simple as using linear regression or fitting a polynomial to your data. However, as the dimension of the data grows and the relationship gets more complex we cannot resort to fitting simple functions without prior knowledge of the relationship. Therefore an alternative, more general, approach was developed that allows modelling any arbitrary data relationship. The term "machine learning" encompasses all such models, i.e. functions/algorithms, that can be "fit" to data by iteratively improving the model's parameters in a way such that it "better approximates" the relationship in the data. Of course this is rather vague as the user has control over what "better approximates" means, and within this field are both simple techniques like linear regression and complex techniques like boosted decision trees and artificial neural networks. The power in all of these techniques relies on the scientist having (usually) large amounts of data for which each entry in the dataset is labelled with its true value. For example, if we have a set of images then we also need to tell the model what is in the image, e.g. label every image with the value 1 for "cat", 2 for "dog", and 0 for neither. We would then expect the machine learning technique trained on this dataset to learn how to identify cats and dogs. For any given problem there are many possible techniques to choose from, but in particle physics analysis two of the most popular are boosted decision trees and artificial neural networks.

2.1 Boosted Decision Trees

A classic and very powerful machine learning technique is a boosted decision tree. Essentially, what a decision tree does is ask your data a series of yes/no questions until it reaches a final verdict regarding the probability that the data belongs to a certain class, which, in our case, is whether or not the jets are descendants of a top quark. More mathematically, if your dataset resides in an n-dimensional space then a decision tree linearly splits the n-space into a set of n-dimensional cubes, each with its own probability of corresponding to a top quark. Then, to produce a more general technique we "boost" the decision trees. Boosting is when you train one decision tree on the dataset, then you train another decision tree but weight the entries in the dataset by how well the first decision tree performed on that entry. As a result, the second decision tree will care more about the entries that the first had trouble with, but will do worse on the rest. We then repeat this process until a given stopping criterion is met. The final boosted decision tree (BDT) is then the weighted average of each of these individual decision trees. Although decision trees may seem rather simple, through boosting they become universal approximators, meaning that a sufficiently large BDT can model any continuous function restricted to the subspace consisting of the training data. The power in this technique is that it is simple to use and efficient both in training and evaluating as well as easy to understand. However, this technique still lacks the power to be fully expressive in a concise manner, i.e. it has a difficult time modelling complicated non-linear functions. In order to help this problem, it is commonplace to construct customized features, i.e. engineered features, that makes it easier for the BDTs to separate the data. Since designing the actual model is fairly straightforward, most researcher time tends to be spent on choosing and creating new engineered features that help separate the data. This process makes sense while it is motivated by physics knowledge; however, in practice we have seen that even wildly unmotivated features can be constructed that greatly benefits the BDT. This raises the question, how do we know if we've found the best features for the BDT?

2.2 Artificial Neural Networks

Another powerful machine learning technique is artificial neural networks, or simply neural networks, which have taken the machine learning world by storm this millennium. Neural networks are inspired by the neuron architecture of the human brain and are armed with the power of the Universal Approximation Theorem, meaning, equivalently to the BDTs, they can approximate any continuous function. In theory, this theorem makes them incredibly expressive. In practice, it is quite another story, as optimizing the model such that it properly reaches this perfect approximation is very difficult, but when successful tends to be more concise. The major disadvantages of this approach are that there are many different settings to tune through trial and error (called hyperparameters), they are slower to train and evaluate, more difficult to interpret, and difficult to optimize effectively, i.e. less "plug-n-play". The main benefit that we care about here, other than the theoretically more efficient approximation than a BDT, is that a neural network should not need engineered features to reach its best approximation. That is not to say that engineered features won't help it. In fact, it would make things easier, but a neural network can still achieve the same level of performance without them. This means that if we provide the neural network with all simple variables, meaning the ones that are natural measurements and not human designed, it should internally be able to create the necessary engineered features and reach equivalent, if not better, performance to a BDT. Researchers could then spend less time concerned with feature engineering for BDTs and instead focus on optimizing their neural network model by tuning hyperparameters which, in principle, can be done in an algorithmic manner. Consequently, researchers will have more time to focus on solving physics problems.

3 Experimental Design

3.1 Data Selection

For training the neural network we focused on using simulated data of collisions where a pair of top quarks (top & anti-top) were produced. Specifically, we focused on such collisions that also produced a Higgs boson ($t\bar{t}H$), a W boson ($t\bar{t}W$), a Z boson ($t\bar{t}Z$), or just the top pair ($t\bar{t}$+jets). Equal parts of each of these datasets were used to train the neural network. In all, roughly 8 million collisions were used for training. For training the neural network discussed here, only the "basic" information in the dataset was used. This means only information specific to individual particles, not any higher level information that is computed from multiple particles, e.g. invariant mass. For simplicity, we will refer to these as "high-level" variables. Specifically, the basic variables used per particle for the neural network were: transverse momentum p_T, azimuthal angle in the plane transverse to the collision beam, pseuodorapidity, mass, charge, DeepCSVprobb D_b, DeepCSVprobbb D_{bb}, DeepCSVprobc D_c, DeepCSVprobudsg D_{udsg}, and qgid. The DeepCSV variables represent discrimants from neural networks that score the jet as possibly arising from a bottom quark, pair of bottom quarks, charm quark, or a lighter quark/gluon, respectively. Qgid similarly represents a quark/gluon discriminant for that particle and is produced by a neural network. When creating datasets for training the neural network each triplet of jets was sorted by their overall b-score, which is the sum $D_b + D_{bb}$. The datasets were also filtered to only include collisions with 96 GeV $\leq$ mass(3-jets) $\leq$ 266 GeV. Other experiments were conducted where a neural network was trained on only the BDT variables (described next) or on the combination of basic features and BDT features using this same dataset.

Loukas Gouskos and Huilin Qu developed and provided us with the BDT considered in this paper. They designed the data selection as follows [3][4]. The BDT training dataset consisted of both basic and high-level variables and had each triplet organized such that the first jet (labelled the b-jet) had the highest b-score, the second (labelled the W_1-jet) had the higher p_T among the two remaining, and the third (labelled the W_2-jet) had the lower p_T. The basic variables for the BDT were the transverse momentum, mass, b-score, cTagCvsL, cTagCvsB, ΔR, ptD, axis1, and multiplicity of each jet. The cTag variables are defined as:

$$cTagCvsL = D_c/(D_c + D_{udsg})$$
$$cTagCvsB = D_c/(D_c + D_b + D_{bb})$$

The ptD, axis1, and multiplicity are all engineered variables that form a quark/gluon discriminant similar to the qgid used for the neural network dataset. ΔR is an engineered variable that measures the spread of the particles constituting the jet. The high-level variables are the invariant mass m_{ij} for each pairwise combination as well as for the total triplet. Only $t\bar{t}$+jets collisions with $|m(\text{3-jets}) - 173| < 80$ GeV, where 173 GeV is the mass of the top quark, were used for training the BDT.

3.2 Model Selection

After trying many different configurations and neural network approaches, we settled on the following relatively simple feed-forward neural network architecture and training configuration. The neural network consists of six hidden layers, each of constant width. Each layer consisted of a linear component, one parameterized rectified linear unit (PReLU), one batch normalization component, and one dropout component. The linear component only consists of the matrix of weights that is multiplied by the inputs to the layer. The output of the linear layer is the input to the PReLU activation function which is consistent across all nodes in the layer. We chose PReLU as the activation function since it has been shown that they perform especially well on classification tasks [5][6]. Then the activations are batch normalized with learnable affine scale parameters. Batch normalization was originally introduced as a way to reduce internal covariance shift, but has also been shown to improve training convergence to a better local optima and reduce variance [6][7]. Although the original paper suggests implementing batch normalization before the activation unit, there is recent empirical evidence that suggests implementing it after the activations improves performance [6][8]. Finally, there is a dropout component with a setting of 0.3, meaning there is a 30% chance that any individual input to the dropout will be output as zero and a 70% chance that it will be unchanged. This helps prevent overfitting while using a large neural network. For the final layer, since we are doing binary classification, we use a sigmoid activation function to ensure the targets represent probabilities. This structure is represented more visually in Table 1. For more details regarding neural network architectures, see [9].

The hyperparameters used for training the neural network are presented in Table 2. Since we are doing binary classification we used the standard loss function for this task, binary cross entropy [9]. Adam was chosen as the optimizer since it is particularly robust to the choice of learning rate and has been shown to perform very well for most tasks [10][11]. The optimal learning rate was determined through a small grid search while the other parameters for Adam were chosen to be consistent with best practices. The learning rate annealer decreases the learning rate over time to help the neural network converge to a local optima. Given our settings, every ten rounds of training the learning rate will decrease by a factor of 10 but with a lower bound of 10^{-4}. All of the weights were initialized from a uniform distribution. Xavier and Kaiming initializations were tried, but all produced roughly the same results. The dropout setting, number of training examples per batch, and number of total rounds of training (epochs) were all optimized through manually grid searching. A GPU was also used in order to speed up training times.

Layer	Output Dimension	Number of Learnable Parameters
Input	30	
Hidden 1	975	
Linear		$30 \cdot 975 = 29,250$
PReLU		1
Batch Norm		2
Dropout		0
Hidden 2-6	975	
Linear		$975^2 = 950,625$
PReLU		1
Batch Norm		2
Dropout		0
Output	1	
Linear		975
Sigmoid		0
Overall	$30 \mapsto 1$	4,783,368

Table 1: Neural network architecture.

Hyperparameter	Value
Loss Function	Binary Cross Entropy
Optimizer	Adam
Learning Rate	10^{-4}
β_1	0.9
β_2	0.999
Learning Rate Annealer	
Patience	10 epochs
Decay Factor	0.1
Threshold	10^{-4}
Weight Init	Uniform
Dropout	0.3
Batch Size	8192
Epochs	60
GPU	Yes

Table 2: Settings for training the neural network.

4 Analysis

4.1 Model Training

We implement the neural networks in the PyTorch [12] framework which is a very flexible library that shines at balancing low level control with higher level generalization and clean usage. The Python code implementations of all of the results shown in this paper are publicly available on Github[1]. The dataset split was chosen to be 80/10/10 for training, validation, and testing, respectively. This means that 80% of the data was used to optimize the model, 10% was used to make heuristic decisions about the model such as when to stop training, and 10% was used to evaluate model performance. In Figure 1 is the Binary Cross Entropy loss of the "Basic NN" over the course

[1]github.com/mdkdrnevich/DeepHadTopTagger

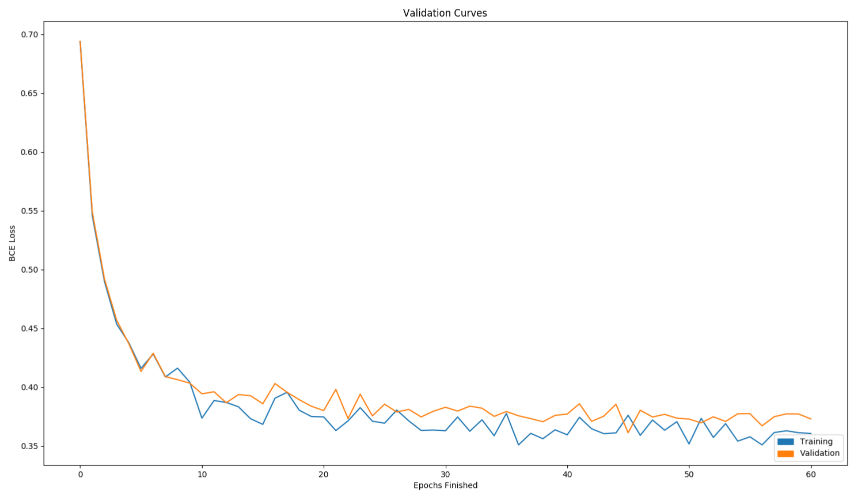

Figure 1: Loss curves for the Basic NN on training and validation sets.

of the training. The BDT was trained by L. Gouskos and H. Qu using the XGBoost framework [3].

4.2 Comparisons

For evaluating these two models we set aside roughly 120,000 $t\bar{t}H$ collisions that neither model has seen in any form. First, we evaluated the neural network and BDT on each possible triplet for the 120,000 collisions, without any selection criteria. Then, we generated a receiver-operator characteristic (ROC) curve for each and plotted them in Figure 2. The ROC curve plots the true positive rate versus the false positive rate across various cutoff choices for the model output. For any given cutoff, an output above that cutoff is labelled as signal and an output below it is labelled background, then the true and false positive rates are measured. The area under an ROC curve (AUC) is a standard metric of discriminant performance. Note that there are three versions of neural networks plotted here, demonstrating the performance of neural networks trained on exclusively basic features, the BDT variables (listed as "Eng"), or all of these together. There is a very slight improvement in the performance of the neural network when training

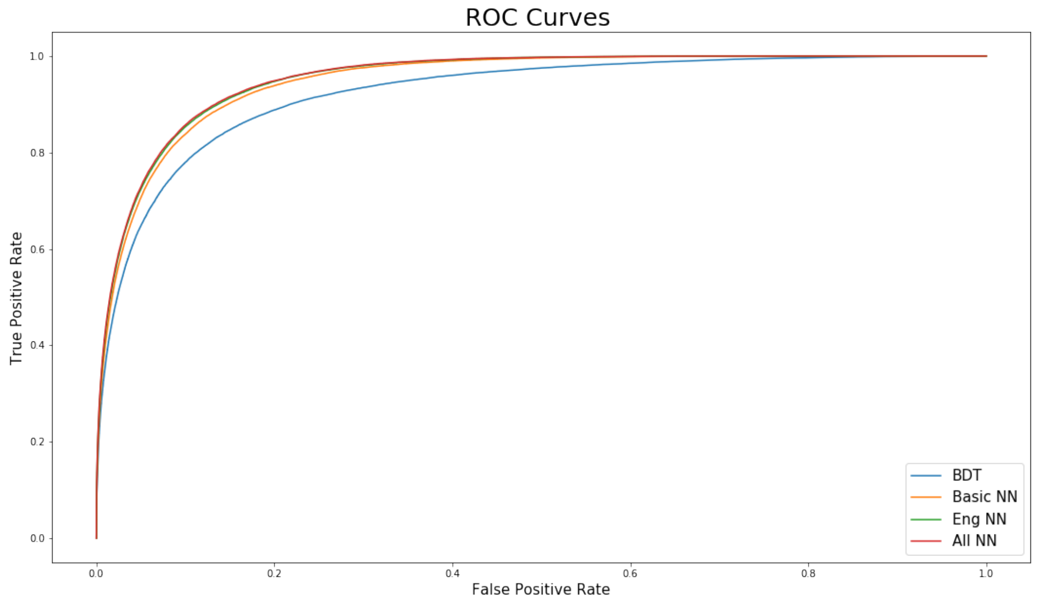

Figure 2: ROC curves for the BDT and ANN.

on just the engineered features or all of the features, but this is small enough that it could just be a statistical fluctuation in the training. However, all of the neural networks strongly outperformed the BDT. The area under the ROC curve was 0.923 for the BDT and 0.948 for the Basic NN, which is a significant relative improvement for this type of analysis.

To compare the effectiveness of neural networks versus boosted decision trees for the task of tagging top quark hadronic decay particles we have to define a custom accuracy metric. The standard metric used previously for models that involve "tagging" is defined as follows. Given a collision, we take each possible combination of three jets and evaluate the model on each combination. Then we choose the combination with the highest discriminant value and label that as the top quark triplet. This guess is then validated against the ground truth information to determine whether it is correct. The fraction of correct assignments over the dataset is the accuracy. Mathematically, let N be the number of collisions, S_i be the true set of triplets of jets that came from a top quark (obtained using simulator information) for the i^{th} collision, $M : X \to \mathbb{R}$ be the model (BDT or ANN), and $Comb(i, 3)$ be the set of all triplets of jets corresponding to the i^{th} collision. Then we can write our accuracy metric as

$$L(M) := \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}_{S_i}\left(\underset{X\in Comb(i,3)}{\arg\max}\, M(X)\right)$$

where $\mathbb{1}_{S_i}(X)$ is the indicator function on the set S_i, i.e.

$$\mathbb{1}_{S_i}(X) = \begin{cases} 1 & \text{if } X \in S_i \\ 0 & \text{if } X \notin S_i \end{cases}$$

We then filtered the 120,000 collisions by only allowing collisions with two same-sign leptons and a tau lepton to match the preselection standard used in previous experiments for evaluating top-tagging models, reducing the evaluation set to 8000 collisions. On these collisions, the BDT achieved an accuracy of 60.49% while the ANN achieved an accuracy of 64.99%. This is a relative improvement by the neural network of 7.44% and agrees with the conclusions from analyzing the area under the curve metric.

Model	Area Under the Curve	Accuracy
BDT	0.9234	60.49%
NN - Basic Variables	0.9485	64.99%
NN - BDT Variables	0.9526	N/A
NN - All Variables	0.9537	N/A

Table 3: Accuracy and AUC results for each model. Accuracy was only computed for the Basic NN and the BDT to validate the AUC metric.

4.3 Performance Validation

Although the comparisons between the machine learning models looks successful, there is always the possibility that the neural network learned how to model nuances in the simulated data instead of real relationships inherent to physics. In an effort to reveal whether this had happened to our model, we constructed a method of directly comparing the Monte Carlo data to real detector data. We took a dataset of real detector data, where one was known

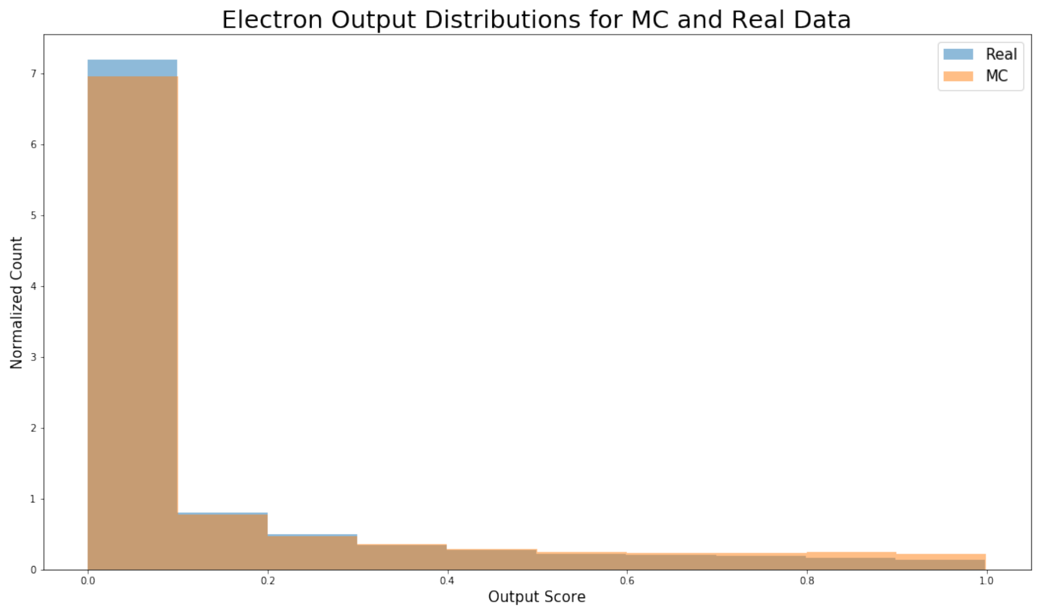

Figure 3: Histogram of the ANN output scores for the electron datasets.

to contain electrons in the decays and the other was known to contain muons, and then applied a filter that would ensure the remaining data would have a relatively high fraction of $t\bar{t}$+jets collisions. The filter was as follows: at least four jets with $\eta < 2.5$ and momentum $p_T > 25$ MeV where at least one must have a high chance of being a bottom quark, i.e. b-score > 0.2219, and exactly one lepton of the given type (electron or muon) with $\eta < 2.4$ and momentum $p_T > 35$ MeV. This same collision selection criterion was then applied to the simulated detector $t\bar{t}$+jets dataset and all possible triplets of jets for every collision in each dataset were computed. The expectation is that these two filtered datasets would be directly comparable and any notable differences should result from mismodelling in the simulation or small amounts of non-$t\bar{t}H$ background. If the neural network focused on poorly modeled aspects of the simulation then we would expect it to have a different output distribution. When we applied the neural network to these datasets we got the histograms in Figure 3 and Figure 4. These results provide a clear qualitative agreement between real and simulated data, thus allowing us to conclude that the neural network learned to model the actual physical relationship and not the simulator.

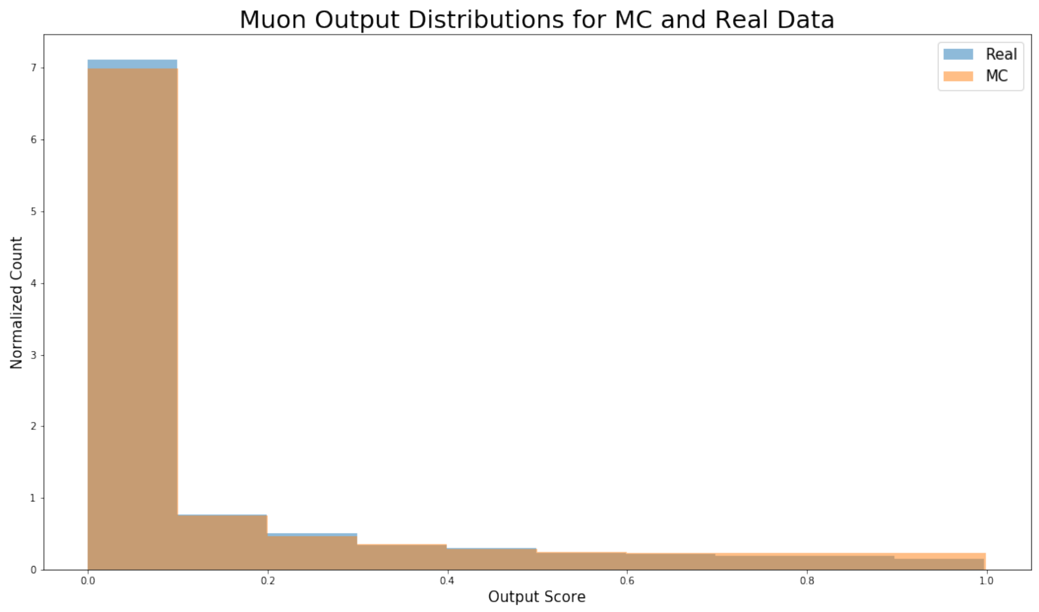

Figure 4: Histogram of the ANN output scores for the muon datasets.

5 Discussion

Our research demonstrated that a neural network given only particle-level information could outperform a BDT that used higher level information in addition to particle-level information. Rather than using many graduate students over many years as monkeys on keyboards trying to produce the right engineered variables, we demonstrated that time may be better spent simply training a neural network. One challenging problem with using a neural network for this purpose is that we are never quite sure what features the neural network is internally constructing in order to beat our hand-constructed variables. One might think that it has simply learned how to construct the same variables that the BDT used but to a higher accuracy, yet in this case that does not appear to be so. In Figures 5 and 6 we plot the correlation of the outputs of the neural network and the BDT for each type of dataset used in the training. There is clearly a wide spread and the correlation values lie in the range 0.5 to 0.6 for each plot, showing low commonality among their predictions. A natural conclusion is that the neural network has realized a more valuable subset of information than the BDT which allows it to achieve better performance. Future research could focus on how we might develop a more powerful algorithm, such as a larger neural network, which is able to use all of the information available to it and further improve its performance. One such algorithm could be a neural network that has the outputs of the BDT added to its inputs.

Through training a neural network on individual-jet level information we showed that we could outperform the current most accurate boosted decision tree for identifying top quark decay that was trained on both jet level and higher level features. Although there was some gain in adding the higher level features and other engineered variables to the neural network training, it was small in comparison to the neural network's performance gained relative to the BDT. This result suggests that not only should we use this neural network in particle physics analysis, but we should also focus future particle physics analysis research on training neural networks rather than training BDTs or feature engineering.

References

[1]S. Chatrchyan et al., "Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC", Phys. Lett. **B716**, 30–61 (2012).

[2]M. Tanabashi et al., "Review of particle physics", Phys. Rev. D **98**, 030001 (2018).

[3]L. Gouskos and H. Qu, *Update on resolved top tagger*, 2018.

[4]H. Qu, Private Communication, 2018.

[5]K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: surpassing human-level performance on imagenet classification", CoRR **abs/1502.01852** (2015).

[6]D. Mishkin, N. Sergievskiy, and J. Matas, "Systematic evaluation of convolution neural network advances on the imagenet", Computer Vision and Image Understanding (2017) `https://doi.org/10.1016/j.cviu.2017.05.007`.

[7]S. Ioffe and C. Szegedy, "Batch normalization: accelerating deep network training by reducing internal covariate shift", CoRR **abs/1502.03167** (2015).

[8]Reddit, *Batch normalization before or after relu?*, (2017) `https://www.reddit.com/r/MachineLearning/comments/67gonq/d_batch_normalization_before_or_after_relu/`.

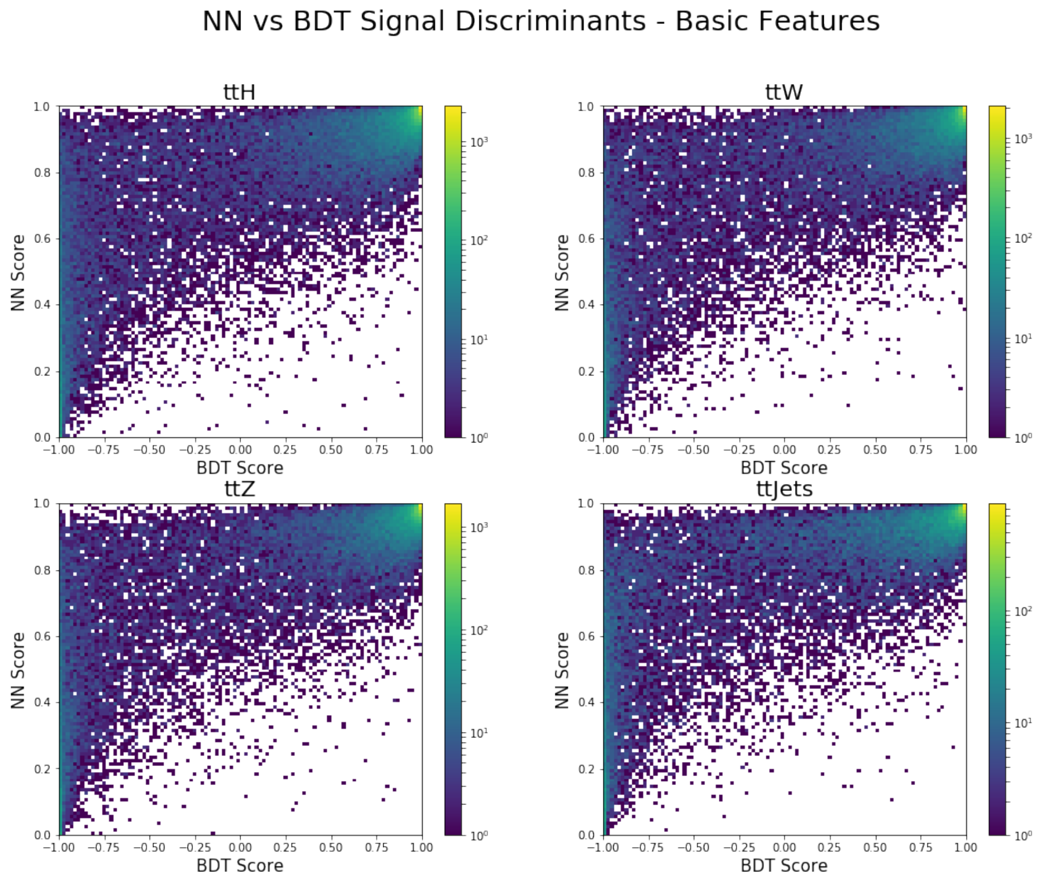

Figure 5: Output comparisons for the signal dataset.

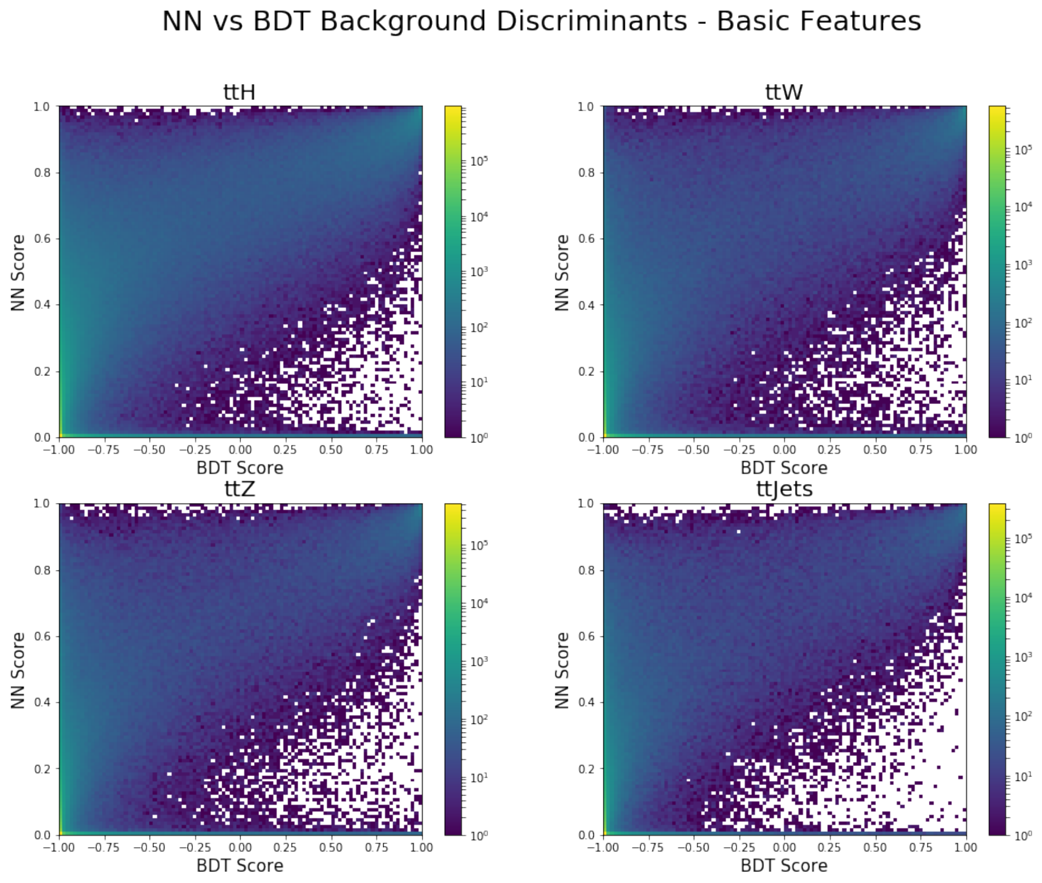

Figure 6: Output comparisons for the background dataset.

[9]I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*, `http://www.deeplearningbook.org` (MIT Press, 2016).

[10]D. P. Kingma and J. L. Ba, "Adam: a method for stochastic optimization", International Conference on Learning Representations, 1–13 (2015).

[11]V. Bushaev, *Adam - latest trends in deep learning optimization*, (Oct. 2018) `https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c`.

[12]A. Paszke et al., "Automatic differentiation in pytorch", (2017).