Training a neural net to identify top quark jets

KDSMIL001 — September 2022

Abstract

We train a neural net to identify jets as originating from top quarks as opposed to other quarks; a process known as top tagging. High-level variables describing the jets are used as opposed to constituent data as constituent requires considerably more computing power. The model's accuracy is then determined for different cuts of jet kinematics to investigate where the model has deficiencies.

1 Introduction

In proton-proton collisions at the Large Hadron Collider (LHC) at CERN, top quarks (along with their anti-particle counterpart) are produced about once every few seconds. Due to their proportionally large mass ($\sim 173\,\text{GeV}$) their decay time is too short for them to be observed directly, so the next best way to determine if a top quark was produced is to look at the hadronic jets produced by the interaction and determine if they were a result of a top quark or some other, less interesting quark. This process is called top-tagging.

One way to perform this classification between signal (top quark jet) and background (any other jet) is to use a neural net. Neural nets are well suited to this task as they are able to handle the large volumes of Monte Carlo truth data that are available to us and can easily be constructed to provide an output from 0 to 1, which we can take to be the neural net's classification of signal (closer to 1) and background (closer to 0). We used data from `https://cds.cern.ch/record/2825328` [1] to train and test a deep neural net in top-tagging using "high-level" quantities describing the jets. This report will focus on the creation of that neural net, its performance according to commonly used statistics, and some investigation into where the neural net performs best in relation to the kinematics of the jets being top-tagged.

2 The data

As stated before, the data we used to train and test the neural net comes from the dataset used in [1]. Due to computation power restrictions, we were unable to use their train dataset, so we split the test dataset into $\frac{2}{3}$ for training and $\frac{1}{3}$ for testing.

The data is split up into two types: constituent and high-level. The constituent data is simply the p_T, η, φ and energy E of the particles making up each jet. This is a lot of information and would ultimately be the best data to use as it holds all the available information about a jet, which neural nets are very good at sifting through to find the information relevant to the task at hand. The only issue is that it takes a lot of RAM to load all the data into, as well as requiring a lot of computation power (or time) to really get to a meaningful result.

To try cut down on computation time and resources, we used the high-level data. These are 15 variables calculated for each jet, from the constituent data, which have been identified by [2] and [3] as summarising the data in a way that lends itself to top-tagging. The 15 variables are:

- Energy Correlation Ratios: ECF_1, ECF_2, ECF_3, C_2, D_2
- N-subjettiness: τ_1, τ_2, τ_3, τ_{21}, τ_{32}

- Splitting Measures: $\sqrt{d_{12}}$, $\sqrt{d_{23}}$
- Q_W

The loss function we used, which is discussed further in section 3, was the binary cross-entropy loss function. If we were to use two input variables with wildly different scales, say energy on the GeV scale and distance on the nanometre scale, then changing an energy parameter even slightly will have such a large impact on the output of the neural net, and thus the loss function, that changing a distance parameter will practically have no effect. To avoid this, we did some basic preprocessing. This involved simply subtracting the mean of a specific quantity from the value for all the jets, then dividing by the standard deviation. This centers the data around 0 and gives it a standard deviation of 1. Doing this ensures that our neural net will not be trained to weight a specific input more simply because it's on a larger scale than another input.

Each jet has a label stating whether it is signal or background, as well as weights used in training to weight the loss function according to the true distribution of signal and background. The 15 high-level quantities, the labels, and the weights are all that we gave to the neural net for training.

3 Creation of the neural net

The neural net was created using Python and Keras [4], which is an interface for using TensorFlow. We used an input layer with 15 inputs; one for each of the high-level quantities. We then used 3 hidden layers each with 20 neurons using the Rectified Linear Unit activation function. Finally an output layer of 1 neuron using the sigmoid activation function was used. The model was compiled with the argument `optimizer='adam'`, which has a learning rate of 0.001. We used a batch size of 100, with our training set being made up of around 1.6 million jets. 8 epochs were used. A validation dataset of 5% of the training dataset was used.

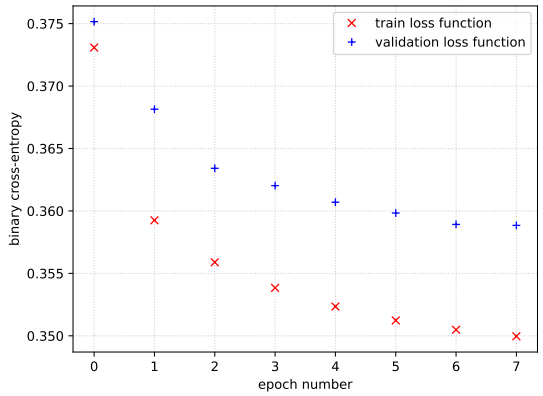
This configuration was chosen fairly arbitrarily, aside from the input and output layers, with a focus on simplicity and short run times (around 10 minutes of training). We chose the binary cross-entropy for our loss function as it is both simple and well-suited to a (surprise) binary classification task such as this one.

We can look at the evolution of the model with two parameters—the loss function and the accuracy—evaluated at each epoch in figure 3.1 for both the training and validation datasets. In this case, the accuracy is defined as the proportion of correct predictions when calling everything with prediction greater than 0.5 a signal, and background otherwise. This is a fairly rudimentary statistic and will be improved upon in later sections.

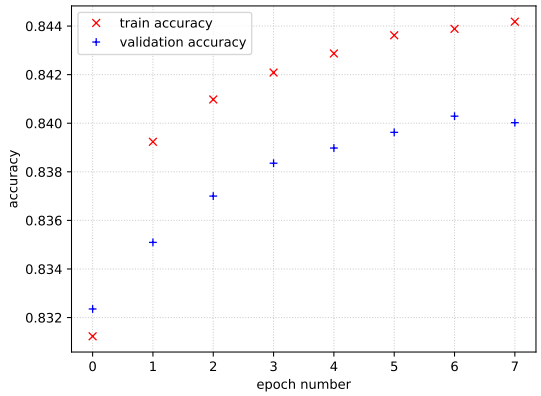
The neural net finished training with a loss function value of 0.3500 and an accuracy of 0.8442.

4 Predictions

With the neural net trained, we were then able to use it to predict the classification of jets in our testing set. This dataset contained around 8 million jets. A first step towards improving the accuracy metric calculated earlier is to look at the confusion matrix, which tells us how many true positives, false negatives, false positives, and true negatives were predicted. In our case we are considering signal to be "positive" and background "negative".



(a) Progression of the loss function over the training of the neural net, shown for both the training dataset as well as the validation dataset. The loss function used was binary cross-entropy.



(b) Progression of the accuracy of the neural net over its training, evaluated on both the training dataset and the validation dataset. Accuracy here is defined simply as the proportion of correct predictions.

Figure 3.1

		Predicted Classification	
		Signal	Background
Actual Classification	Signal	TP	FN
	Background	FP	TN

Table 4.1: Confusion matrix for

References

[1] *Constituent-Based Top-Quark Tagging with the ATLAS Detector*. Tech. rep. All figures including auxiliary figures are available at https://atlas.web.cern.ch/Atlas/GROUPS/PHYSICS/PUBNOTES/ATL-PHYS-PUB-2022-039. Geneva: CERN, 2022. URL: https://cds.cern.ch/record/2825328.

[2] *Identification of hadronically-decaying top quarks using UFO jets with ATLAS in Run 2*. Tech. rep. All figures including auxiliary figures are available at https://atlas.web.cern.ch/Atlas/GROUPS/PHYSICS PHYS-PUB-2021-028. Geneva: CERN, 2021. URL: https://cds.cern.ch/record/2776782.

[3] *Identification of Hadronically-Decaying W Bosons and Top Quarks Using High-Level Features as Input to Boosted Decision Trees and Deep Neural Networks in ATLAS at $\sqrt{s}$ = 13 TeV*. Tech. rep. All figures including auxiliary figures are available at https://atlas.web.cern.ch/Atlas/GROUPS/PH PHYS-PUB-2017-004. Geneva: CERN, 2017. URL: https://cds.cern.ch/record/2259646.

[4] *Keras: the Python deep learning API*. URL: https://keras.io/ (visited on 09/29/2022).