LIP-STUDENTS-20-00

Deep Neural Network uncertainties in VLQ search at LHC

Gilberto Cunha[1,a]

[1] *Universidade do Minho, Braga, Portugal*

Project supervisors: Nuno Castro, Miguel Romão

October 21, 2020

Abstract. A study of a developed Deep Learning model's uncertainties in Vector Like Quarks signal vs background classifications using Monte Carlo Dropout. The model was developed and trained using generated data in accordance with LHC data and theoretical predictions and tuned using Bayesian inference. The developed model showed good results in background to signal ratio reduction, which is promising for Machine Learning applications in HEP. The Monte Carlo Dropout distributions hint to the model having learned more from background data than signal, but further analysis is required

Keywords: Vector Like Quarks, Deep Neural Networks, Monte Carlo Dropout

1 Introduction

The Standard Model (**SM**) is a Particle Physics model that has proven to be a very good description of the behaviour of fundamental particles and the Electromagnetic, Weak and Strong Nuclear forces. Despite its undeniable success, it's now understood that the SM is still incomplete. Experiments have been and are still being conducted to explore the depths and limitations of the model, one famous case - and the focus of this report - being the LHC particle collider.

In this collider, particles are accelerated to extremely high speeds, and sent to particle detectors where these particles collide and interact with each other, originating particle showers that are measured. The measurements of these interactions - which are also denoted as **events** - are later analysed to verify their agreement with SM theoretical predictions.

Most of the events agree with the SM - these are called the **background** - while other rarer and not yet experimentally observed ones don't - the **signal** or **BSM** (Beyond the Standard Model) events. Signal events are, therefore, events that describe new Physics.

Being able to distinguish the two using measurement data is key to finding what the SM can and cannot describe and in turn to finding new Physics, an example being the discovery of the Higgs Boson back in 2012, confirming theoretical predictions.

A big difficulty in distinguishing signal from background is that signal events are extremely rare - otherwise they would have already been detected - meaning that the vast majority of the measurements are populated by background events, which are not of much interest. The hard task is then to find these very few signal events amidst an overwhelming majority of background.

For this effect, a Neural Network was developed using Keras in order to learn both background and **VLQ** [3] - see Fig. 1 - signal distributions and classify events accordingly from their simulated data.

Even Neural Networks, however, have errors associated with their classifications. This project aims to study these uncertainties in Neural Network predictions by applying Monte Carlo Dropout.

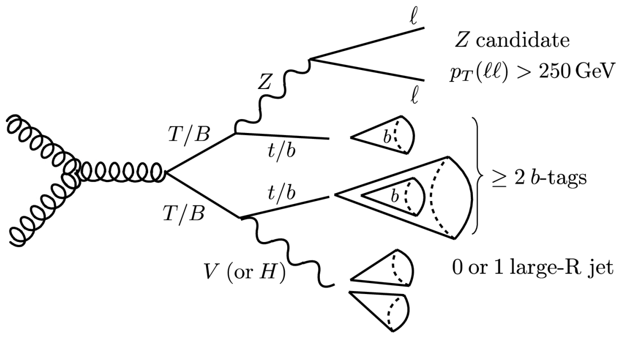


Figure 1. Vector Like Quarks Feynmann diagram. [3]

2 Deep neural network concepts

2.1 The benefits of Machine Learning for this classification

Disregarding Machine Learning, the most common way to classify signal from background is using cuts. This is how the Higgs Boson was experimentally discovered.

However, in a scenario where it isn't known what signal to expect, or that theoretical predictions don't hold altogether, cuts are most likely to be insufficient. Instead, a Neural Network could be trained to learn the background distribution and classify anything that doesn't follow it as signal. This isn't what this project is about - the signal data is generated from theoretical predictions after all - but serves as a motivation for the possible applications of Deep Learning in this task.

2.2 How DNNs work

Deep Neural Networks are basically complex linear regressors. They are composed of **neurons** - see Fig. 2 -

[a]e-mail: gcacademic@outlook.pt

that apply an affine map - a **weight** vector $\boldsymbol{w}$ and a **bias** b - to their input vector $\boldsymbol{x}$ and a non-linear **activation function** f on top, since non-linearity allows for a much wider range of applications. Its scalar output y is given by:

$$y = f(\boldsymbol{w} \cdot \boldsymbol{x} + b) \tag{1}$$

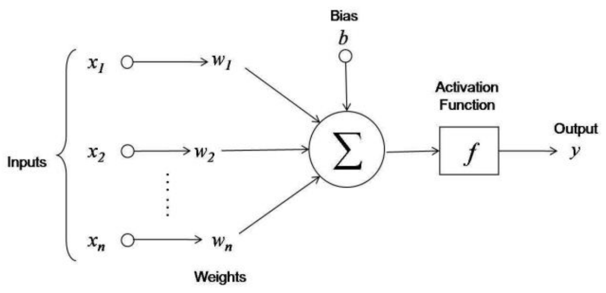


Figure 2. Representation of a Neuron. [4]

When you structure neurons in **layers**, which hold a variable size of neurons, you get a Neural Network - see Fig. 3. In NNs, each neuron of each layer is connected to every neuron of the next layer. Every connection has a weight and every neuron has a bias.

There are three types of layers in NNs:

- **Input Layer**: receives the data features.
- **Output Layer**: represents the classification classes.
- **Hidden Layer**: increases the Network's complexity.

When a NN has multiple hidden layers, it is called a Deep Neural Network.

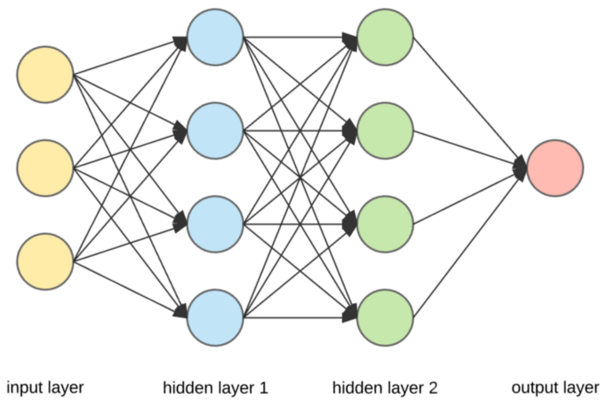


Figure 3. Representation of a Deep Neural Network. [5]

In a way, DNNs can be thought of as a box that receives data features as input and outputs numbers that convey important information for the task at hand. In classification tasks, these outputs generally represent the class probabilities of the data.

The vector output $\boldsymbol{x}^{(n)}$ of the n-th layer of a DNN can be mathematically described as follows:

$$\boldsymbol{x}^{(n)} = f(\boldsymbol{W}^{(n)}\boldsymbol{x}^{(n-1)} + \boldsymbol{b}^{(n)}) \tag{2}$$

Where the i-th component of $\boldsymbol{x}^{(n)}$ is the output of the i-th neuron of the layer, $\boldsymbol{W}^{(n)}$ is a matrix of weights where the i-th row and j-th column entry represents the weight from the i-th neuron of the previous layer to the j-th neuron of the current one, $\boldsymbol{x}^{(n-1)}$ are the outputs of the neurons of the previous layer and $\boldsymbol{b}^{(n)}$ the bias vector where the i-th component is the bias of the i-th neuron.

It is easy to use eq. 2 to calculate the outputs of the network from its inputs. This is what is called the **forward propagation**. To get the *correct* outputs, however, **backpropagation** is needed.

2.3 Backpropagation

An NN is really just a multivariable function - usually with a lot of variables - and the objective is to give the best values to those variables/parameters - weights and biases - such that the NN performs the best possible at whatever task it is given. To do this the NN is initially given some random parameters which are continuously adjusted using backpropagation so as to improve itself.

In order to improve the network, an objective and **differentiable** way to evaluate its performance is needed. This is done using a **loss function**, a function that defines the error between the predicted outputs $\hat{\boldsymbol{y}}$ of a **batch** - a collection of input data points - and that batch's desired outputs (or labels) $\boldsymbol{y}$. The output of a loss function is a scalar and the smaller it is the better the performance. Then the task of finding the best model is the task of **minimizing** the loss function.

The loss is usually calculated for a batch and not each data point because this is faster and results in less loss fluctuation per prediction, therefore a better chance of finding loss minima.

Multivariable calculus says that the direction opposite to a function's gradient is the direction of **highest decrease** in the function's output. Applying the chain rule to eq. 2, it is possible to calculate the derivative of the loss function J with respect to each network parameter, thus finding the gradient vector - this is why J must be differentiable. All that is left is to take a step η - the **learning rate** - in the opposite direction of the gradient to update all parameters $\boldsymbol{\theta}$ - this update is the **gradient descent**:

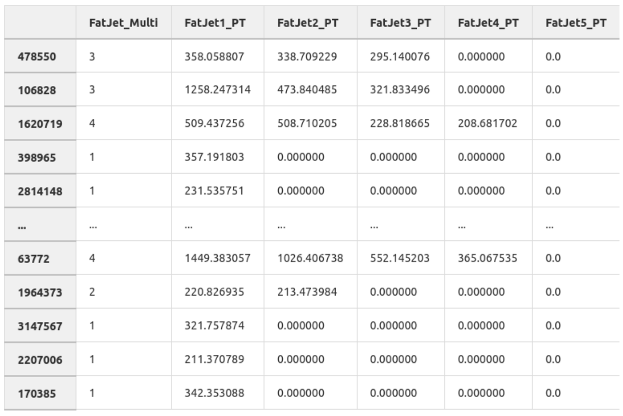
$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \eta \boldsymbol{\nabla}_{\boldsymbol{\theta}} J \tag{3}$$

Where t just means it is the t-th iteration of gradient descent. In truth eq. 3 is just one way to update network parameters. There are many more sophisticated ways to do this implemented using different **optimizers** - algorithms that update the network parameters - such as Adam.

3 Data description and analysis

3.1 Data structure

The data used for this classifier is tabular, meaning there are some fixed variables/features that describe each event. Some examples are the number of measured electrons, jets or muons, the η and ϕ orientations of the measured jets or the missing transverse momentum [1]. After pre-processing - which will be discussed later - there are a total of 72 features for every event - including a label, a cross-section and a sample.



	FatJet_Multi	FatJet1_PT	FatJet2_PT	FatJet3_PT	FatJet4_PT	FatJet5_PT
478550	3	358.058807	338.709229	295.140076	0.000000	0.0
106828	3	1258.247314	473.840485	321.833496	0.000000	0.0
1620719	4	509.437256	508.710205	228.818665	208.681702	0.0
398965	1	357.191803	0.000000	0.000000	0.000000	0.0
2814148	1	231.535751	0.000000	0.000000	0.000000	0.0
...	...	...	...	...	...	...
63772	4	1449.383057	1026.406738	552.145203	365.067535	0.0
1964373	2	220.826935	213.473984	0.000000	0.000000	0.0
3147567	1	321.757874	0.000000	0.000000	0.000000	0.0
2207006	1	211.370789	0.000000	0.000000	0.000000	0.0
170385	1	342.353088	0.000000	0.000000	0.000000	0.0

Figure 4. Representation of the structure and some features of the data.

3.2 Samples and cross-sections

Every event comes from a **sample**. Different particles can arise from the same interactions, and these different products of an interaction constitute different samples. Each sample has a **cross-section**, a physical quantity that is a proxy for the event's probability of occurrence.

It has been said before that the data utilized is generated from physical simulations. In this data generation, the cross-section of each event is added as a feature of each data point. To get the physical distributions of the data, each data point must therefore be weighted according to its sample cross-section, since some samples are more likely to occur than others.

However, the cross-section corresponds to the weight of the sample, not of each individual event. To get the weight of each event, one must then divide the cross-section of each sample by the number of events of that same sample. This "event weight" is denoted by **gen weight**. For each sample s with N_s events and cross-section σ_s, the gen weight ω of each event of that sample is given by:

$$\omega = \frac{\sigma_s}{N_s} \tag{4}$$

An important note is that the gen weight is a generated feature, which means that it cannot be passed into the developed model when predicting data points as either background or signal, since the cross-section of an event cannot be obtained by particle collider measurements. Only measurable features can be used by the model to make predictions.

3.3 Excessive memory usage and cuts

One problem with the data is that it is quite large, which is impractical for most personal computers. This is solved with **cuts** - a way of filtering out data fairly for both background and signal, usually carefully chosen to filter more background due to its overwhelming predominance.

The data describes a physical distribution across all features, cuts simply remove parts of the distribution. Say it is known that most of the time the signal should have 2 electrons, then one cut example would be to remove all events that have less than 2 electrons. For the VLQ signal, the applied cuts - based on Fig. 1 - were:

- At least 2 leptons
- At least 1 fat-jet

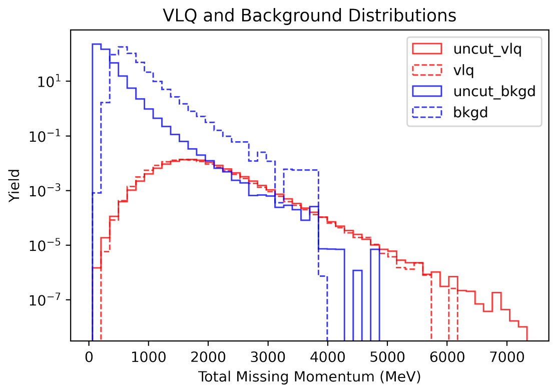


Figure 5. Cut and uncut VLQ and background weighted total missing momentum distributions.

3.4 Data pre-processing

Almost all data needs some clean-up and this case is no exception. Firstly, some columns with information relative to data generation - which cannot be obtained from measurements - have to be dropped and columns relative to the labels and samples have to be added. Secondly, the already described cuts have to be applied and the gen weights calculated for each sample. Finally, as the data comes in multiple files relative to each sample, they have to be concatenated. In short:

1. Add and remove necessary columns per sample
2. Apply cuts per sample
3. Calculate gen weights per sample
4. Concatenate all samples

After applying this pre-processing to the data, its structure is as described in Sec. 3.1.

4 Deep neural network model

4.1 Train, validation and test data

Not all data can be used to train the model, it has to be split *equally* into three datasets (so as to ensure all datasets have the same statistical significance):

- **Training set:** used in backpropagation to train the model
- **Validation set:** used to evaluate the model while training and tuning hyperparameters - number of hidden layers, number of features per hidden layer, etc.
- **Test set:** used only for the final evaluation of the model, after it is completely trained and all hyperparameter studies have been conducted

4.2 The model

With the goal of classifying events as signal or background, a deep learning model was developed using **Keras**. The model architecture follows the presented scheme:

1. 69 neurons in the input layer
2. Batch Normalization layer on top of the input layer to mimic data standardization
3. Hidden layers with relu activations
4. Dropout layer on top of every hidden layer except for the last
5. 1 neuron in the output layer with sigmoid activation

The input layer has 69 neurons because the label, cross section and sample features are not experimentally measurable.

The Batch Normalization layer, which changes the data's mean and standard deviation, using them as learnable parameters, is used to mimic standardization. This is useful because most features have very distinct ranges of values - number of electrons is usually very low, but missing momentum can reach extremely high values - which might result in higher range features producing higher activations, creating a bias in the model.

Dropout layers, which randomly sets to zero some neuron's weights of the respective layer, are also present since the main objective is to study the uncertainties of the NN using Monte Carlo Dropout.

There is also only one neuron with a sigmoid activation function in the output layer to force the output of the network to be within the $[0, 1]$ range, where 0 represents background and 1 represents signal - just as in the labels.

The chosen loss function for this model, since it is a binary classifier, is **binary cross-entropy**. It also uses the Adam optimizer.

4.3 Loss function, gen weights and class weights

With i being each event from a batch, $\boldsymbol{y_i}$ is the one-hot label encoding of the event and $\boldsymbol{\hat{y}_i}$ the predicted probability of the event i being of class n, the binary cross-entropy loss for that batch is generally computed as follows:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{n=0}^{1} y_{i,n}\log(\hat{y_{i,n}}) \tag{5}$$

There are two problems, however, with this equation for this particular dataset:

- It assumes all data points i have the same importance - it isn't a weighted average. This isn't true because every data point has its gen weight ω_i
- It also assumes both classes are equally represented, which also isn't true, since there are more background events than signal ones

Therefore, eq. 5 has to take into account the normalized gen weights $\tilde{\omega}_i$ and class weights c_n. The gen weights have to be normalized due to their small magnitude, specially for BSM events, which would lead to a very small parameter correction during backpropagation. Considering N_n the total number of elements of class n in the training set:

$$\tilde{\omega}_i = \frac{\omega_i}{\sum_{i=1}^{N}\omega_i}, \quad c_n = \frac{N_{bkgd}}{N_n} \tag{6}$$

Then eq. 5 can be altered as follows:

$$L = -\sum_{i=1}^{N}\sum_{n=0}^{1} c_n\,\tilde{\omega}_i\,y_{i,n}\log(\hat{y_{i,n}}) \tag{7}$$

4.4 Hyperparameter search

Despite the guideline restrictions of the model presented in Sec. 4.2, the following hyperparameters can be tuned in order to optimize the model's performance - by minimizing the validation loss:

- The batch size
- The number of hidden layers
- The number of neurons per hidden layer
- Whether or not Batch Normalization is applied to each hidden layer except for the last
- The dropout rate

There are a number of methods to achieve this, the most popular ones being **Grid search**, **Random search** and **Bayesian inference search**.

Grid search implies setting up a grid of possible parameter values - a list of batch sizes, another one of hidden layers, etc - and testing each one of those. Random search implies searching randomly in a defined space for those parameters - consider the same lists but sampled randomly. Bayesian inference, on the other hand, relies on predicting a distribution of model performance for each parameter and updating it with each performed run while choosing the next run parameters based on the most likely parameters to get a good performance in the predicted distributions [2].

For this purpose, Bayesian inference search was used with the optuna python package.

4.5 Monte Carlo Dropout

After having found the best model with the best hyperparameters, Monte Carlo Dropout can be applied. Monte Carlo Dropout consists in applying Dropout in the model not just during training, but also during prediction. This

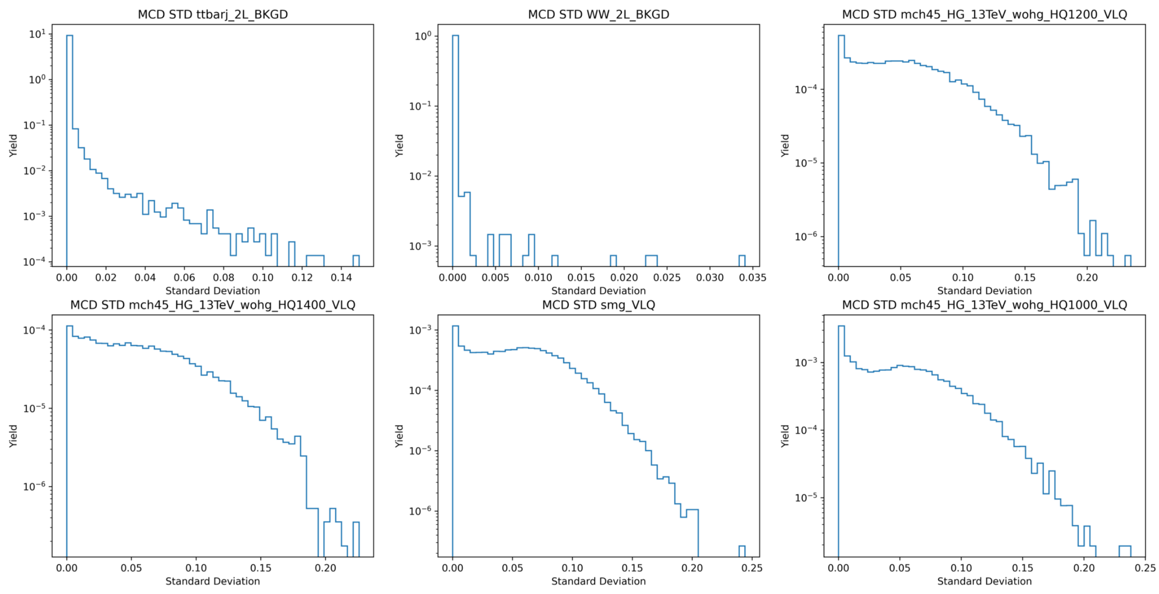


Figure 6. Standard deviation distributions by sample.

means that one single data point is classified more than once - say a hundred times - and since these predictions won't be the same due to Dropout, one can find the standard deviation of the predictions as a measure of the model's uncertainty (and the mean as the final prediction).

5 Results

The optimal model found with the Bayesian inference has a batch size of 256, 2 hidden layers with 84 and 49 neurons, respectively, no batch normalization besides the standardization layer and a dropout rate of 5%.

The model's performance was tested using both Dropout and Monte Carlo Dropout using the AUC metric. The model got an AUC of 0.99673 with Dropout and 0.99692 with Monte Carlo Dropout, a minute difference.

Model	ROC AUC
Regular	0.99673
MC Dropout	0.99692

Table 1. Area under the ROC curve for both models

Both the Monte Carlo and the regular Dropout models recover the physical distributions of the background and VLQ signal quite well qualitatively, although the background distributions are more accurate than the signal ones - see Fig. 7. This might suggest that the model didn't learn as well from VLQ data as with background.

Another important evaluation of the model is the background to signal ratio reduction. The developed model reduced this ratio by roughly 713% in both Dropout regimes while wrongly classifying 1.72% of signal and 13.73% of background - see Fig. 8. This was done by selecting an operating point of 2.78×10^{-4}, which maximized the difference between the true positive rate and the false positive rate in the ROC curve. Using this operating point, the model's prediction of the signal physical distribution also loses accuracy compared to the 0.5 threshold due to the increase in false positives.

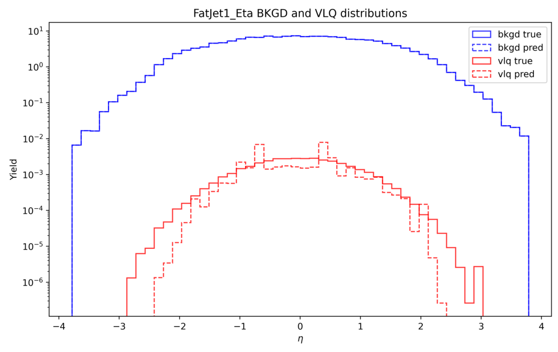


Figure 7. MC Dropout Fat Jet 1 η distributions with a threshold of 0.5.

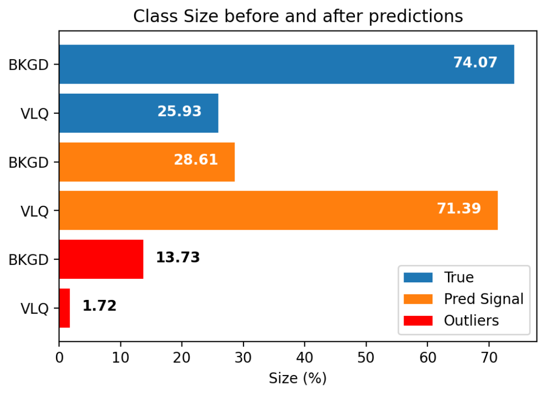


Figure 8. Background/Signal ratio reduction after predictions.

Evaluating only the Monte Carlo Dropout model, the standard deviation distributions - a proxy for the model's uncertainty - in classifications were plotted by class and by sample. Analyzing by class - see Fig. 9 - it can be noticed that VLQ is dominant in high prediction uncertainties. By taking a closer look - see Fig. 6 -, it can also be noticed that all VLQ samples have approximately the same shape, meaning they might all be contributing about the same to this unbalance in class uncertainty. Furthermore, only some background sample distributions have high uncertainty tails, which might indicate that the model is mixing these few background samples with the VLQ samples, but further analysis is required to state these claims as scientific facts.

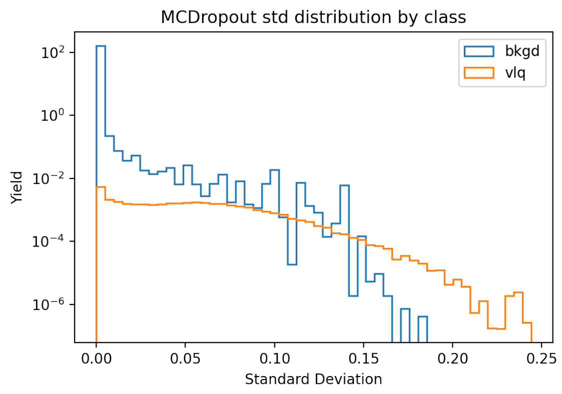


Figure 9. Standard deviation distributions by class.

6 Conclusions

This work shows that Deep Learning might have very useful applications in High Energy Physics, particularly in finding new physics in particle collider experiments.

For the signal of study, the developed Neural Network is able to reduce the background to signal ratio more than seven-fold while only losing a mere 1.73% of signal data. Considering how unbalanced the data measured in particle colliders can be, this can prove quite useful in tipping the scale in a more favorable direction.

Moreover, although Monte Carlo Dropout doesn't seem to improve the model in any meaningful way, it does guide to a new path of data analysis in the form of model prediction uncertainty which can show potential weaknesses either in the model or in the method used to perform the task at hand.

For future work, a deeper analysis of the Monte Carlo Dropout distributions would reveal more about the full potential of the technique. It would also be interesting to explore unsupervised Deep Learning methods, which would greatly simplify the use of direct measurement data in the model.

Acknowledgements

I would like to thank my supervisors Nuno Castro and Miguel Romão for the kind help they provided throughout the whole internship. It was a truly memorable and enlightening experience to work with and learn from them.

References

[1] Schwartz, M. D. (2017). TASI Lectures on Collider Physics. arXiv preprint arXiv:1709.04533, 75.

[2] Géron, A. (2019). Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems. O'Reilly Media, ISBN 978-1-492-03261-8.

[3] Aaboud, M., Aad, G., Abbott, B., Abeloos, B., Abhayasinghe, D. K., Abidi, S. H., ... Abulaiti, Y. (2018). Search for pair and single production of vectorlike quarks in final states with at least one Z boson decaying into a pair of electrons or muons in p p collision data collected with the ATLAS detector at s= 13 TeV. Physical Review D, 98(11), 112010.

[4] Arthur Arnx, "First neural network for beginners explained (with code)", https://towardsdatascience.com/first-neural-network-for-beginners-explained-with-code-4cfd37e06eaf

[5] Arden Dertat, "Applied Deep Learning - Part 1: Artificial Neural Networks", https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6