The use of machine learning in the search for di-photons in association with missing energy

Student:

Theodore Cwere Gaelejwe (536500)

Supervisor:

Prof. Bruce Mellado

A research report submitted in fulfilment of the degree of MSc e-science in the

School of Computer Science and Applied Mathematics

May 28, 2020

Declaration

I, Theodore Cwere Gaelejwe, declare that this research report is my own, unaided work. It is being submitted in fulfilment of the requirements for the degree of MSc e-science at the University of the Witwatersrand. It has not been submitted for any degree or examination at any other university. Where the discussion has been informed by previously-submitted work, this has been indicated as such.

Theodore Cwere Gaelejwe

May 28, 2020

Abstract

The Large Hadron Collider (LHC) generates petabytes of data per second during each data taking period and has long term data storage in the order of exabytes. Sophisticated machine learning (ML) techniques are used at the trigger and final state level to analyse this data. Boosted Decision Trees (BDTs) in particular, have been the default ML tool for this task. However, in the recent past, more modern techniques such as Deep Learning have emerged and there has been growing justification for their use in High Energy Physics (HEP). We conduct a comparative study between BDTs and (Deep Neural Networks) DNNs in classifying signal and background events in the $H \to \gamma\gamma + \chi$ decay channel. A comparison between a fully supervised and weakly supervised model is also conducted. Results suggest that DNNs outperform BDTs and the fully supervised model is outperformed by the weakly supervised model though it is more robust.

Acknowledgements

I would like to express my gratitude to Prof. Bruce Mellado, Dr. Xifeng Ruan and Dr. Kehinde Tomiwa, at the Wits Institute for Collider Particle Physics for their contribution during the preparation of this work. I would also like convey my utmost thanks to Dr. Helen Robertson, Prof. Turgay Celik and members of the School of Computer Science and Applied Mathematics for their input through suggestions, comments and corrections. The financial support of the DST-CSIR National e-Science Postgraduate Teaching and Training Platform (NEPTTP) towards this research is hereby acknowledged. It is to be noted that opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to the NEPTTP.

Contents

Declaration i

Abstract i

Acknowledgements i

List of Figures **v**

List of Tables **vii**

1 Introduction **1**

1.1 Background 1

1.2 Problem Statement 4

1.3 Research Aims and Objectives 4

1.3.1 Research Aims 4

1.3.2 Research Objectives 5

1.4 Definitions 5

1.4.1 Missing Transverse Energy 5

1.4.2 Fake Missing Transverse Energy 6

1.4.3 Event selection 7

1.5 Limitations 7

1.6 Overview 7

2 Literature Review **8**

2.1 Supervised, Semi-supervised and Unsupervised learning 8

2.2 Weakly Supervised Learning 9

2.3 Artificial and Deep Neural Networks 11

2.3.1 Weight Initialization 12

2.3.2 Activation Functions 14

2.3.3 Gradient Descent 19

2.3.4 Types of Gradient Descent . . . 19
2.3.5 Backward Propagation . . . 20
2.3.6 Vanishing and Exploding Gradients . . . 23
2.4 Decision Trees . . . 24
2.4.1 Bootstrapping and Bagging . . . 24
2.4.2 Information Gain and Entropy . . . 25
2.4.3 Boosted Decision Trees . . . 26

3 Research Methodology **28**
3.1 Research Design . . . 28
3.2 Data . . . 29
3.3 TensorFlow . . . 32
3.4 Features . . . 32
3.5 Hyperparameters . . . 33
3.6 Performance Metrics . . . 35
3.6.1 Receiver Operating Curves . . . 35
3.6.2 Confusion Matrix . . . 36

4 Results **38**
4.1 Supervised Learning Results . . . 38
4.2 Weakly Supervised Learning Results . . . 40
4.3 Discussion . . . 42

5 Conclusions & Future Work **43**
5.1 Conclusions . . . 43
5.2 Future Work . . . 44

A Univariate Distributions **45**
A.1 Fully Supervised Features . . . 45
A.2 Weakly Supervised Features . . . 48
A.3 Correlation Matrices . . . 51

B Confusion Matrices **52**
B.1 Confusion Matrices . . . 52

C Overfitting Check **54**

C.1 Loss Plots 54

Bibliography

List of Figures

1.1 Schematic depicting real and fake E_T^{miss} [1] 6
1.2 Schematic showing the reconstruction E_T^{miss} [2] 6

2.1 General Structure of a DNN [3] 12
2.2 Schematic of the ReLU function [4]. 16
2.4 Schematic representation of SELU function [5]. 19
2.5 An illustration of a decision tree algorithm [6] 24

3.1 An illustration of the data being split into different $S_{E_T^{miss}}$ categories . 32

4.1 Fully supervised BDT output for the low $S_{E_T^{miss}}$ category 38
4.2 Fully supervised BDT output for the intermediate $S_{E_T^{miss}}$ category . . . 38
4.3 Fully supervised BDT output for the high $S_{E_T^{miss}}$ category 39
4.4 Supervised GBDT ROC curves for Low, Intermediate and High $S_{E_T^{miss}}$. 39
4.5 Supervised DNN ROC curves for Low, Intermediate and High $S_{E_T^{miss}}$. 39
4.6 Weakly supervised GBDT output for the low $S_{E_T^{miss}}$ category 40
4.7 Weakly supervised GBDT output for the intermediate $S_{E_T^{miss}}$ category 40
4.8 Weakly supervised GBDT output for the high $S_{E_T^{miss}}$ category 40
4.9 Fully supervised GBDT ROC curves 41
4.10 Weakly supervised GBDT ROC curves 41

A.1 Fully Supervised 1D Distributions 45
A.2 Fully Supervised 1D Distributions 46
A.3 Fully supervised 1D Distributions 47
A.4 Weakly supervised 1D Distributions 48
A.5 Weakly supervised 1D Distributions 49
A.6 Weakly supervised 1D Distributions 50

B.1 Low $S_{E_T^{miss}}$: Confusion Matrices for A400Z, ggZH, and 275mx60 52
B.2 Intermediate $S_{E_T^{miss}}$: Confusion Matrices for A400Z, ggZH, and 275mx60 52
B.3 High $S_{E_T^{miss}}$: Confusion Matrices for A400Z, ggZH, and 275mx60 . . . 53
B.4 Low $S_{E_T^{miss}}$: Confusion Matrices for A400Z, ggZH, and 275mx60 53
B.5 Intermediate $S_{E_T^{miss}}$: Confusion Matrices for A400Z, ggZH, and 275mx60 53
B.6 High $S_{E_T^{miss}}$: Confusion Matrices for A400Z, ggZH, and 275mx60 . . . 53

C.1 Low $S_{E_T^{miss}}$: Train_Val Loss Plots 54
C.2 Intermediate $S_{E_T^{miss}}$: Train_Val Loss Plots 54
C.3 High $S_{E_T^{miss}}$: Train_Val Loss Plots 54

List of Tables

3.1 Data samples used 30

3.2 Table showing S_{ET}^{miss} categories. 30

3.3 Number of events in each data set 31

3.4 Table showing GBDT hyperparameters. 34

3.5 Table showing DNN hyperparameters. 34

3.6 DNN Model Architecture 34

3.7 A typical confusion matrix 36

4.1 Comparison of Area Under the Curve for BDT and DNN 39

4.2 Comparison of Area Under the Curve for fully supervised ($GBDT_F$) and weakly supervised ($GBDT_W$) GBDT 41

List of Abbreviations

h Higgs Boson

2HDM Two Higgs Doublet Model

AUC Area Under the Curve

BDTs Boosted Decision Trees

BSM Beyond the Standard Model

CERN European Organization for Nuclear Research

DNNs Deep Neural Networks

FPR False Positive Rate

GANs Generative Adversarial Networks

GeV Giga electron Volt

HEP High Energy Physics

HL-LHC High Luminosity Large Hadron Collider

LHC Large Hadron Collider

MC Monte Carlo

MIL Multi-Instance Learning

ML Machine Learning

MSE Mean Squared Error

PReLU Parametric Rectified Linear Units

ReLU Rectified Linear Units

ROC Receiver Operating Curve

RReLU Randomized Rectified Linear Units

SELUs Scaled Exponential Linear Units

SM Standard Model

SMOTE Synthetic Minority Oversampling TEchnique

SNN Self Normalising Network

TDAQ Trigger and Data Acquisition

TeV Tera electron Volt

TPR True Positive Rate

Chapter 1

Introduction

1.1 Background

The Standard Model (SM) is one of the most successful physics theories that describes fundamental forces in nature and categorizes fundamental particles governed by these forces[1]. The SM Higgs boson is the latest success of the SM and marked a milestone when it was discovered in 2012. According to the European Organization for Nuclear Research (CERN) website[2], the Higgs boson is an elementary particle that exists in the Higgs field where it has no electric charge, is scalar and has a spin of zero [7]. At the LHC, the Higgs boson is mainly produced by proton-proton collisions, however, it can also be produced by the fusion of two gluons, Vector Boson Fusion or the decay of W or Z bosons [8]. Another recent and promising discovery in High Energy Physics (HEP) is the Madala boson, which we discuss further in the next paragraph.

The Madala boson is a heavy scalar which was first hypothesised by physicists at the University of the Witwatersrand in 2015/2016 as a result of observed discrepancies in LHC data and SM predictions [9, 10]. According to the Madala Hypothesis, this heavy scalar has more than twice the mass of the SM Higgs boson at 272 GeV and interacts with dark matter. In the literature, the Madala Hypothesis is said to be an extension of the Two Higgs Doublet Model (2HDM) which postulates the existence of a new heavy scalar H that disintegrates into the Higgs boson (h) and a scalar boson (S) [1]. This model seeks to explain discrepancies found in LHC data from "Run 1

[1] https://home.cern/about/physics/standard-model

[2] https://home.cern/about/updates/2013/05/basics-higgs-boson

and Run 2" [9, 11, 12]. A recent compelling article by Von Buddenbrock et al. discusses evidence of this heavy scalar originating from anomalies in multiple leptonic final states in proton-proton collisions at the LHC [13].

At the beginning of December 2018, Long Shut Down 2 (which is the second shutdown of the CERN accelerator complex) commenced and upgrades to the LHC in preparation for Run 3 started in January 2019. These upgrades are aimed at increasing interactions per bunch-crossing which leads to higher energy levels and in turn leads to increased luminosity. This luminosity generates fake missing energy which pollutes the signal, making the discrimination of signal and background events more challenging given that trillions of events are produced per second [1]. Since it is impossible for available computing and storage systems to process and store all this data (which amounts to petabytes), the events are filtered using a system called the Trigger and Data Acquisition System (TDAQ) and only a couple of gigabytes remain for processing [14, 15]. In light of these large data volumes, machine learning techniques can be a useful tool in assisting physicists in their various analyses of LHC data.

Most HEP problems can be cast as ML problems because of the way in which the data is structured [16]. In general, the idea behind ML is to find a mapping $f : X \to Y$, which describes the relationship between X and Y and can optimize some predefined loss function, say $L(y, \hat{y})$, where X is the feature space, Y the output space, y the ground truth label and $\hat{y}$ the predicted label. An important objective in machine learning is to find a model that generalizes very well and not necessarily the one that achieves the highest accuracy on the specific data sets used to train and validate it [16, 17]. Tree Based and deep learning algorithms have been found to generalize better than most algorithms and for this reason, ML can make a significant contribution in HEP.

In Guest et al. [16] deep learning is described as a group of algorithms that create hierarchical representations of input data. Given that the processes that generate particle physics data at the LHC structure it in a hierarchical fashion, it is plausible to believe that Deep Neural Networks (DNNs) are well suited for HEP classification problems. For example, many LHC events are made up of jets, and jets are made up of hadrons.

Hadrons produce tracks that are normally reconstructed by detectors and those tracks are made up of hits. All this data is naturally organised into a hierarchical structure that can be fed into a DNN relatively easy [16].

Currently at CERN, ML is employed in track finding, various reconstruction processes, object identification (both at the trigger and final state level), data and detector simulations, as well as event classification to mention a few [18]. Since the High Luminosity LHC (HL-LHC) is expected to produce 100 times the current data volumes, simulation requirements are expected to exceed available computational resources [19]. Therefore, fast simulation through deep learning techniques such as Generative Adversarial Networks (GANs) can be probed to address this challenge. Already 3D convolutional GANs, CaloGAN and the GeantV project have shown promising results for fast and full simulation [19, 20, 21].

Often, the goal of recent HEP studies is to discover new particles Beyond the Standard Model (BSM) and/or measure properties of those already confirmed with a higher precision. This is a challenging task since most HEP problems are too complex for us to simply use rectangular cuts on event variables to make a classification. Effectively, highly reliable data sets and analysis tools are needed to achieve these goals. As mentioned above, machine learning can provide solutions for many of these problems, although with some acceptable degree of error.

In summary, we connect the motivation for the application of ML in HEP to the broader physics motivation. As is widely known, the convention in physics is to build a rigorous and mathematically sound theoretical model that can be used to hypothesize about the components of nature and the universe. Then experiments are designed and carried out to prove what the theory claims to be true. Since the SM is complete and there is no standing theory that accurately explains physics BSM, some physicists believe that a new approach, where the formulation of a theory is done in parallel to experimentation, can yield interesting results in a shorter period of time. This research follows this proposed approach in as far as experimentation is concerned, in an effort to confirm the Madala hypothesis. This is because it could very well take decades to formulate another theoretical model capable of explaining physics BSM.

1.2 Problem Statement

Ideally, we want to use ML algorithms that are easy to implement and are highly interpretable such as Decision Trees and Random Forests. To make training easy, we also require fully labelled data sets with no class imbalances. This situation is, however, a departure from the reality of most physicists and machine learning practitioners because simple models such as decision trees are often unable to learn complex non-linear relationships in HEP data sets, which are notoriously class imbalanced and have high dimensionality. Furthermore, generating reliable HEP data sets with ground truth labels is challenging and time consuming. As a result, we turn to deep learning models to try and build a robust model for discriminating between signal and background events. This is because these models are known to handle high dimensionality and large data volumes very well. To circumvent the problem of finding labelled data sets for a fully supervised algorithm, we explore weakly supervised learning, which allows us to train a model on data with either incomplete or inaccurate labels.

1.3 Research Aims and Objectives

This section contains the aims and objectives of this research. Section 1.3.1 outlines the research aims and Section 1.3.2 the research objectives in that order.

1.3.1 Research Aims

The aims of this research are twofold. The first is to compare a tree based model against a deep learning model. In particular, to compare BDTs implemented via the TMVA library against vanilla DNNs implemented using Keras[3] with TensorFlow as the backend. Secondly, it is to compare the performance of a fully supervised model against a weakly supervised model. We will perform a hyperparameter search to optimize the models and compare their respective performance.

[3] https://keras.io/

1.3.2 Research Objectives

- Train supervised BDTs and perform hyperparamter search to find the optimal configuration of the model.
- Train supervised DNNs and tune the hyperparameters to find the optimal configuration of the model.
- Generate ROC curves for both supervised models to compare their performance in terms of Area Under the Curve (AUC) and Recall.
- Train weakly supervised BDTs with optimal hyperparameters and generate ROC curves.
- Compare the performance of the fully supervised and weakly supervised BDTs in terms of AUC and Recall.

1.4 Definitions

In this section, we provide some definitions of important physics concepts that come up frequently in this document and are not necessarily related to machine learning.

1.4.1 Missing Transverse Energy

"Missing Transverse Energy, which we denote by E_T^{miss}, can be described as energy which is not recognised inside the detector but is known to be there due the laws of conservation of energy and momentum. E_T^{miss} can be calculated from the negative vector sum in the transverse plane of the momenta of all detected objects" [2]. Mathematically, E_T^{miss} can be expressed as

$$" E_{x(y)}^{miss} = E_{x(y)}^{miss,calo} + E_{x(y)}^{miss,\mu} \tag{1.1}$$

where

$$\begin{aligned} E_{x(y)}^{miss,calo} &= E_{x(y)}^{miss,calo,e} + E_{x(y)}^{miss,calo,\gamma} + E_{x(y)}^{miss,calo,\tau} \\ &+ E_{x(y)}^{miss,calo,jet} + E_{x(y)}^{miss,calo,\mu} + E_{x(y)}^{miss,calo,cellout} \text{ "} \end{aligned} \tag{1.2}$$

In Equation 1.2 above,

- $E_{x(y)}^{miss,calo,e}$ denotes missing energy from electrons in the calorimeter
- $E_{x(y)}^{miss,calo,\gamma}$ denotes missing energy from photons in the calorimeter
- $E_{x(y)}^{miss,calo,\tau}$ denotes missing energy from τ-leptons in the calorimeter
- $E_{x(y)}^{miss,calo,jet}$ denotes missing jets in the calorimeter.

1.4.2 Fake Missing Transverse Energy

"Fake E_T^{miss} occurs when unidentified physics objects are rejected by the jet vertex tagger (JVT) and pileup algorithm. The JVT algorithm was developed to remove pileup jets in the central region of the ATLAS detector". "While using this technique, physicists found that in some cases it failed due to false identification of physics particles. As a result, the E_T^{miss} reconstruction algorithm reconstructs fake E_T^{miss} due to rejection of physics objects" [1, 2]

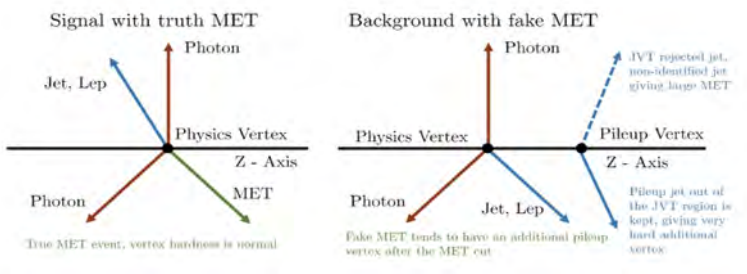


Figure 1.1 Schematic depicting real and fake E_T^{miss} [1]

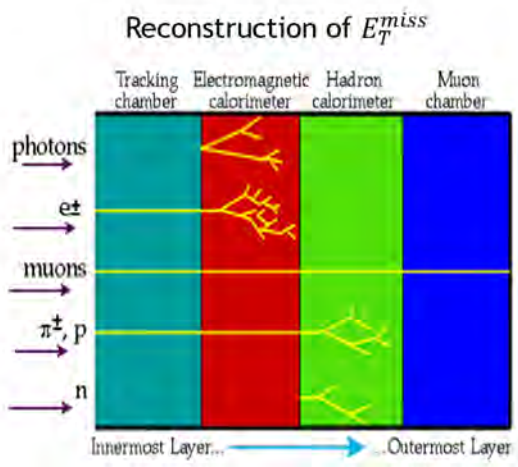


Figure 1.2 Schematic showing the reconstruction E_T^{miss} [2]

1.4.3 Event selection

"The event selection for both the signal and background data used was conducted in the same way. The nominal $h \to \gamma\gamma$ selection from event duplication check to events with invariant mass of the diphoton system in $m_{\gamma\gamma} \in [105, 160]$ GeV [1]. The transverse momenta of jets selected was 20 GeV and the JVT algorithm was not implemented in all data sets because JVT related variables are already incorporated in the analysis" [22].

1.5 Limitations

The only limitation during the implementation of this research was computational resources. Initially, we had access to a GPU cluster based at the University of Chicago through CERN, however, the cluster administrators generally prioritise jobs originating from the university and jobs from outside are put on queue, which often took as long as four weeks. Another problem was that whenever the code met an error during execution, it got terminated and we had to fix it and wait again in the queue for weeks. Though the CERN lxplus cluster is available, it is mainly configured for ROOT and does not have GPU support. This means we have to set up our own environment for Python and all the tools required to run our jobs had to be installed manually which is time consuming. As a result, most of the results were generated in separate chunks on our personal machine.

1.6 Overview

The rest of this research project is structured as follows: Chapter 2 contains the literature review which discusses learning paradigms and the mathematical framework behind our chosen algorithms. Chapter 3 discusses the research methodology to be followed which includes data pre-processing, the selected features, tuneable hyperparameters, software used to implement the algorithms and performance metrics. Chapter 4 presents the results and discusses them. Chapter 5 presents the conclusions and future work. The appendices make up the last part of the document.

Chapter 2

Literature Review

2.1 Supervised, Semi-supervised and Unsupervised learning

Supervised learning describes the set of tasks aimed at training models using labelled data in order to predict targets on future data previously unseen by the model [23, 24, 25]. In the supervised learning paradigm, the objective is to learn a mapping $f : R^n \rightarrow R^m$, that captures the relationship between the inputs and outputs from some training data set $D = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$, where $X = \{x_1, x_2, ..., x_n\}$ is the input vector space and $Y = \{y_1, y_2, ..., y_m\}$ is the output vector space [26]. This means $X \subseteq R^n$ and $Y \subseteq R^m$. For binary classification, $m = 2$, and for multiple classification, $m = q > 2$.

Semi-Supervised learning (or incomplete supervision) refers to the task of training a classifier using data that consists of both labelled and unlabelled examples. An ensemble of clustering and supervised algorithms are normally used for semi-supervised learning tasks. Unsupervised learning on the other hand uses completely unlabelled data (i.e. data without ground truth labels) to train a classifier. This means the model has to learn the underlying distribution of the input data in order to make predictions of the desired output [23]. Clustering techniques such as K-means and its variants are normally used for this type of learning.

2.2 Weakly Supervised Learning

Traditionally, HEP data samples used for analysis are Monte Carlo (MC) simulations generated using simulation tools such as PYTHIA, HERWIG, MAD-GRAPH and SHERPA which make use of MC sampling [27, 28, 29, 30]. When generating data samples, the parameter settings of the simulator are known which means that simulations come with ground truth labels [16]. It is, however, important to note that the process of generating simulated data is laborious and computationally expensive. It is also true that in general, finding an accurate and fully labelled data set is challenging. For these reasons, we may want to make use of algorithms and learning paradigms that do not necessarily require a fully labelled data set. The concept of weakly supervised learning becomes relevant and we discuss it further below.

In general, weakly supervised learning is not well studied in the literature but it is an important subject matter because sometimes we only have labels for a few instances in the data set or we have poorly labelled instances. It can be described as a learning paradigm that attempts to train models by learning with weak supervision [31]. Weakly supervised learning was first applied to HEP in [32], where the authors discriminated jets initiated by quarks from those initiated by gluons using weak supervision.

Often the process of correctly labelling all instances in a data set can be tedious and expensive, and thus training by weak supervision is very attractive. In the article of Metodiev et al. [33] the authors demonstrate that algorithms trained by weak supervision can produce performance equivalent to that of fully supervised algorithms and that weak supervision is robust against mismodeling of discriminating features by state of the art simulation tools [32, 33]. According to Metodiev et al. this approach can increase performance and is robust in the absence of reliable simulated data.

In Zhou [31], three common types of weak supervision are reviewed, including incomplete supervision (where the training data sample is mostly unlabelled and has few labelled examples), inexact supervision (training data labels are high level and not specific) and inaccurate supervision (labels of the training data are not always true). Incomplete supervision is a type of weak supervision which refers to semi-supervised

learning in the traditional sense [25, 26, 34, 35], where the data has a small fraction of labelled instances and the majority of the data has unlabelled instances. In mathematical terms, the task is to develop a function $g : X \subseteq R^n \rightarrow Y \subseteq R^m$ from labelled training data $X_{\text{labelled}} = \{x_i, y_i\}$, where $i = \{1, .., k\}$, and unlabelled training data $X_{\text{unlabelled}} = \{x_j, y_j\}$ where $j = \{k+1, .., n\}$ and $n >> k$.

Inexact supervision is a way of supervising an algorithm based on inexact data. Suppose that we want a large image data set with ground truth labels. Ideally we want each object in the image to be labelled, however, we often only have a label for the image as a whole. A practical example could be an image with red apples, plums and peaches with the label "fruits" instead of a label for each one of the fruits. Inaccurate supervision on the other hand is the a learning paradigm where a model is trained on data with inaccurate or poor labels. This usually happens when the data annotator/s lack knowledge about the ground truth labels, for example when crowdsourcing is used to obtain labels and some of the individuals are spammers who are only interested in earning money [31, 36]. The authors emphasize that often in practice, these different types of weak supervision tend to occur simultaneously and are not necessarily separate.

It is sometimes the case that we have more knowledge about the proportions of the classes than the characteristics of each class. In [32] class proportions are the only input fed into the ML algorithm and the class labels are completely ignored. Another approach to weak supervision is Learning with Label Proportions which relies on the concept of Multi-Instance Learning (MIL) [37, 38]. The idea behind MIL is that we have bags of individual examples without labels. It has previously been demonstrated in [38] that labels are not necessary for classification. For example, suppose we have a binary classification problem with classes 1 and 0. For the training data, we have information about the number of instances from class 1 inside the bag. The algorithm is then configured in such a way that it can find at least one instance from class 1 in a previously unseen bag [32].

2.3 Artificial and Deep Neural Networks

Artificial Neural Networks (ANNs) are inspired by the biological architecture of the human brain consisting of neurons, axons and dendrites. Research into artificial intelligence has allowed ANNs to evolve to better mimic their biological counterparts though not with comparable performance [39, 40]. The first neural network that resembled the human brain was the perceptron network with a single layer. However, after some time, the limitations of such a simple network became clear to scientists when it could not learn complex non-linear relationships [24]. This is when the multilayer ANN was established. The main difference between the human brain and ANNs is the number of neurons, connectivity and overall complexity.

A Neural Network (NN) consists of connected layers of neurons [17, 23]. For a fully connected layer each neuron is connected to at least one other node and this connection is assigned a weight. Neurons can be categorized into three types, namely input, hidden and output. The input layer will have one neuron for each feature of the data we are passing into the NN. The hidden layers are usually dense or fully connected and their purpose is to establish the relationship that exists between the input-output pairs such that the output neuron with the highest probability score corresponds to the correct label. Every layer is activated by an activation function, which determines if that neuron's output will be passed onto the next layer or not. Activation functions are discussed in detail in section 2.3.2.

Deep Neural Networks (DNNs) can learn complex, non-linear relationships [41]. DNNs are very flexible and their power can be attributed to the fact that they can learn embeddings of the feature space and perform transformations that will categorize all data points as belonging to one of the classes of the output space [17, 24]. DNNs have a similar structure to a simple NN and only differ by depth and complexity. Figure 2.1 shows the general structure of a DNN. A simple NN may be thought of as a computational graph called a Directed Acyclic Graph made of tensors and operations [42, 43]. The aim is to optimize the loss function and learn optimal weights for the model.

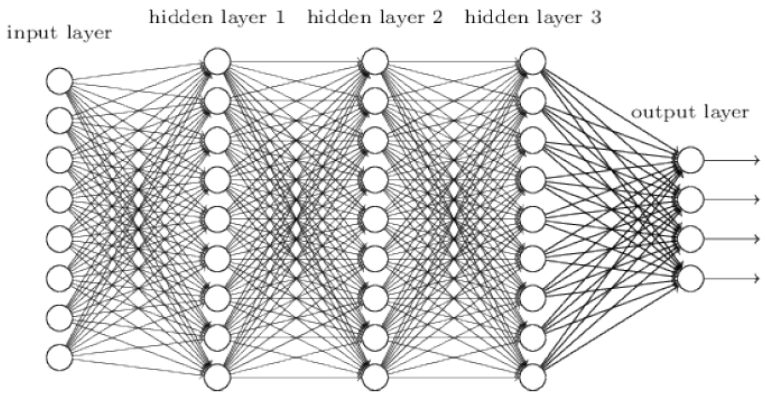


Figure 2.1 General Structure of a DNN [3]

2.3.1 Weight Initialization

The initialization of the weight matrix is a critical aspect of building a good neural network. Often people initialize weights randomly which may hamper the model's learning abilities. The following are two typical scenarios when initializing weights.

- Initialize all weights to zero

  This initialization makes your model no different from a linear model. When all weights are initially set to zero the derivative with respect to the loss function is identical for every w in the weight matrix W^l. Effectively, all weights will have the same values in the next epoch, making the hidden units symmetric and this pattern will propagate through the remaining epochs [17].

- Initialize weights randomly

  This type of initialization introduces stochasticity in the network but can expose it to the undesirable problems of vanishing and exploding gradients. By definition, random numbers are assigned to the weights, including very small values. In this case, we might encounter vanishing gradients. On the other hand, larger values could result in gradient explosions [17].

By initialising weights to zero, we may be starting them off too small such that the signal fades away with successive layers and the network is unable to learn properly. Similarly, by initialising weights randomly, we may start off with a very high signal which will grow with every successive layer and thereby affecting the networks ability to learn. As a result, machine learning practitioners have developed some generally

accepted techniques for handling weight initialization in a prudent way. A few common ones are listed below.

1. Heuristics

   Depending on the activation function applied, we could draw from a normal distribution with variance $\frac{k}{n}$ (where k is a value dependent on the activation function) instead of drawing from a standard normal distribution. This technique will not solve vanishing or exploding gradients completely but it should mitigate them significantly.

   - For ReLU activations

     Make use of He's weight initialisation [44]. Multiply the random weights with

     $$\sqrt{\frac{2}{n_h}}, \tag{2.1}$$

     where n_h is the size of hidden layers.

   - For Tanh activations

     Make use of Xavier (or Glorot) weight initialisation heuristic [45]. Multiply your random weights by

     $$\sqrt{\frac{1}{n_h}}, \tag{2.2}$$

     where n_h is the size of hidden layers.

   - For any other activation function, we can make use of the following weight initialisation heuristic. Multiply the random weights by

     $$\sqrt{\frac{2}{n_h + n_L}}, \tag{2.3}$$

     where n_h is the size of hidden layers and n_L is the size of all layers.

It is worth noting that the bias term can safely be initialized to zero. The gradients of the cost function with respect to the bias term ($\frac{d}{db}C(\theta)$) depends only on the linear activation function of that specific layer so the risk of fading or exploding gradients is non-existent.

2.3.2 Activation Functions

Activation functions are functions that give the output of neurons of a NN. There are mainly two types of activation functions, i.e. linear and non-linear. A key characteristic of activation functions is that they must be continuously differentiable [16]. Linear activation functions are normally used for relatively simple problems like regression. Non-linear activation functions are the most commonly used because of the complex nature of relationships in most data problems. Below we discuss a few popular activation functions, some of which we will use in our models.

1. Linear

   This activation function is suited to regression-type problems where the relationship between variables is mostly linear. Its main drawback, however, is that the functions' derivative is constant and thus, gradient descent is constant throughout. Since the activation is linear, having two or more hidden layers with a linear activation function will not improve learning anymore than a one layer NN would. The linear activation function can be mathematically expressed as

   $$h(x) = x. \tag{2.4}$$

2. Sigmoid (Logistic)

   The sigmoid is a monotonically increasing and continuously differential function with a range between zero and one. It is a logistic function with an S-shape and is given by the mathematical expression

   $$h(x) = \frac{1}{1 + e^{-x}}, \tag{2.5}$$

   where x is the result of the weight matrix multiplied by the input vector X.

   The sigmoid outputs a vector where each element is a probability score. It is most useful when you have a classification task where the label of each input is not mutually exclusive. The highest probability score in the vector is assigned to the predicted class. A drawback of the sigmoid function is that it suffers from the vanishing gradient problem [46].

3. Tanh

   The tanh function is a monotonically increasing and continuously differentiable function similar to the sigmoid function but it is bounded between -1 and 1. This is good because it gives us values of different signs which makes it easier to decide which scores to consider in the next layer and which to ignore. Unlike the sigmoid, tanh is symmetric about the origin. However, it shares the unfortunate weakness of vanishing gradients with the sigmoid activation function [46]. The tanh function slows down exponentially from $x = -2$ and $x = 2$, which implies smaller gradients and thus vanishing gradients. Like the sigmoid, the tanh activation is usually used for binary classification problems. Its mathematical formulation is given by

$$h(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{2.6}$$

4. Softmax

   The softmax activation function is a more generalized version of the sigmoid that takes a vector of N inputs and returns a vector of N probability scores. This vector represents a probability distribution and all N elements sum up to one. The softmax is an exponential form that magnifies differences by applying a transformation that pushes some values closer to one and others closer to zero. It is often used for problems with multiple mutually exclusive labels [17]. Softmax produces a vector that is non-negative with all its elements summing to 1. The mathematical expression for this function is given by

$$h(x) = \frac{e^x}{\sum_j e^{x_i}}. \tag{2.7}$$

5. ReLU

   The Rectified Linear Unit activation is currently the most popular activation function used in neural networks. It ranges between zero and infinity. ReLU is best known for being robust against vanishing gradients [4]. It deals with this problem in the following way; (i) It maps all negative values to zero and (ii) it maps all positive results to themselves, i.e. $y = x$. It is essentially a unit ramp function and is given by Equation 2.8 below

$$h(x) = \begin{cases} 0, & \text{for } x < 0 \\ x_i, & \text{for } x \geq 0 \end{cases} \tag{2.8}$$

and visualized by Figure 2.2

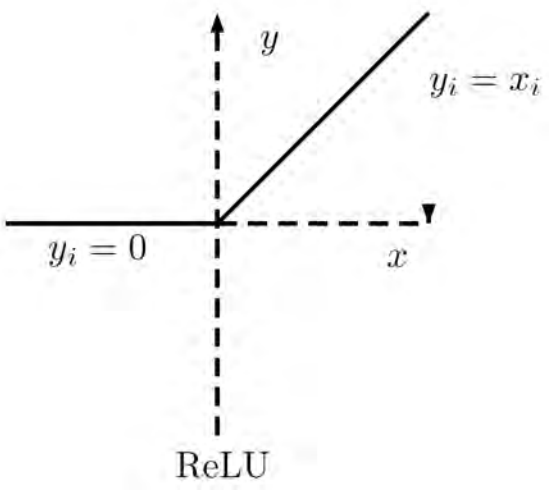


Figure 2.2 Schematic of the ReLU function [4].

This means ReLU has constant derivatives all over its domain [4, 44]. This is useful for controlling vanishing gradients, but may give rise to another problem where the ReLU gets stuck in a state of non-improvement over the domain $x < 0$ of the function, a phenomenon which some researchers refer to as the graveyard region of ReLUs. Essentially, when this happens some neurons die out and part of the network becomes unresponsive. This is called the 'dying ReLU' problem [46]. In 2015, scientists from Microsoft Research proposed a solution for this where instead of having zeros on the negative side of the ReLu function, a small gradient is permitted to enable learning throughout the training [4].

6. ReLU Variations

   Leaky ReLU is a generalisation of ReLU developed to solve the dying ReLU problem. For small datasets, it has been empirically shown that LeakyReLU outperforms ReLU [4]. Currently there are three variants, namely, Leaky ReLU, Parametric Rectified Linear Unit (PReLU) and Randomized Rectified Linear Unit (RReLU) [44]. Leaky ReLU is a ReLU function with a small predefined positive gradient, a, over the domain $x < 0$ in equation 2.9. In Maas et al. the

authors proposed setting a large value for a such as 100 [47]. The Leaky ReLU function is expressed as:

$$h(x) = \begin{cases} a\,x, & \text{for } x < 0 \\ x, & \text{for } x \geq 0. \end{cases} \tag{2.9}$$

PReLU was first proposed in 2015 and differs from ReLU in that the slope over the domain $x < 0$ is determined by backpropagation during training and not predefined as with Leaky ReLU [4, 44]. The equation for the PReLU activation is similar

$$h(x) = \begin{cases} a\,x, & \text{for } x < 0 \\ x, & \text{for } x \geq 0. \end{cases} \tag{2.10}$$

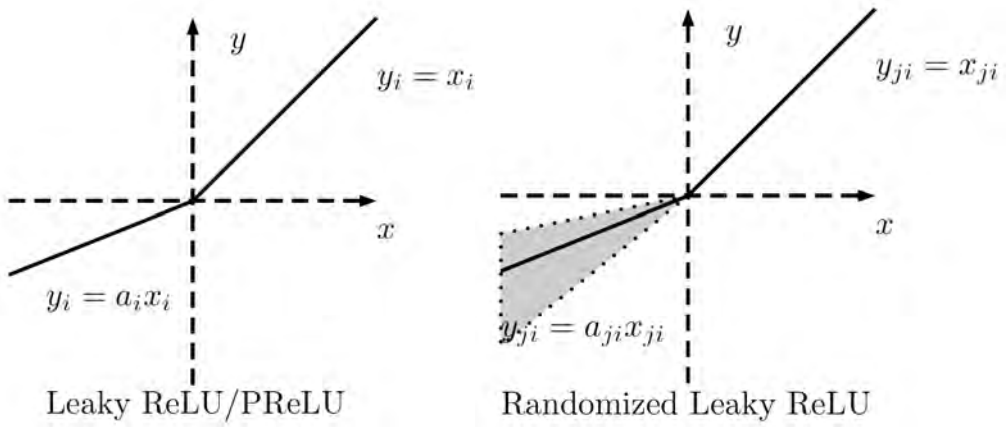
The RReLU activation function is similar to PReLU but the value of a is chosen randomly during training and then fixed during testing. RReLu was first proposed and implemented by the winners of the National Data Science Bowl in 2015 [1]. Its robustness comes from its random nature [4]. The mathematical formulation is given by

$$h(x) = \begin{cases} a\,x, & \text{for } x < 0 \\ x, & \text{for } x \geq 0, \end{cases} \tag{2.11}$$

where $a \sim \mathcal{U}(l, u), \quad l < u$ and $\quad l, u \in [0, 1)$.

Equation 2.9 and Equation 2.10 are both described by Figure 2.3a, and Equation 2.11 is described by Figure 2.3b.

[1] https://www.kaggle.com/c/datasciencebowl



(a) Schematic of Leaky ReLU and PReLU functions [4]

(b) A schematic representation of RReLU function [4]

7. SELU

Scaled Exponential Linear Units (or SELUs) is a recent class of activation function that is very promising. Though not popular, SELUs have more rigorous mathematical framework than ReLUs [5, 46]. SELUs are usually used to construct Self Normalizing Network (SNN) architectures. SNNs are based on SELUs because of their self normalizing characteristics and because other activation functions such as sigmoid, softmax, tanh, ReLU or Leaky ReLU cannot be used to construct them. SELUs are powerful because they perform internal normalization, that is, the output of the activation is already normalized. SELUs can adjust variance in a way that prevents vanishing and exploding gradients [5]. The mathematical expression for SELUs is given by

$$h(x) = \lambda \begin{cases} x, & \text{for } x \leq 0 \\ a(e^x - 1), & \text{for } x > 0, \end{cases} \tag{2.12}$$

where λ is a scaling parameter. When λ is larger than one ($\lambda > 1$), the gradient is larger than one and the activation function increases the variance. On the other hand, near zero gradients can be used to decrease the variance.

Figure 2.4 shows a schematic of the general shape of Equation 2.12

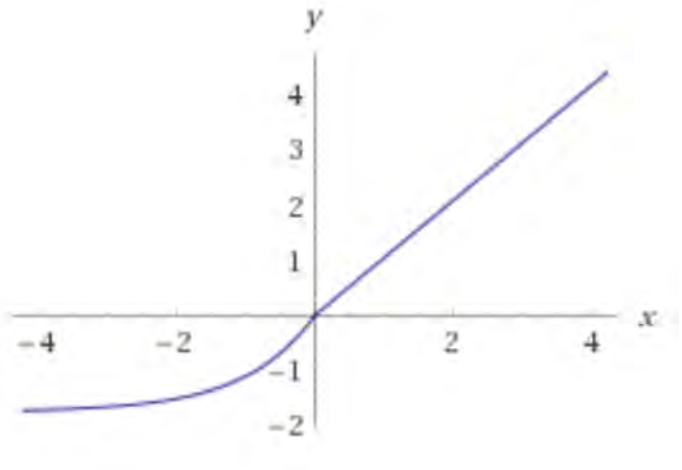


Figure 2.4 Schematic representation of SELU function [5].

2.3.3 Gradient Descent

Gradient descent is a widely used technique for optimising the objective function of a problem [3, 24, 48]. In the context of neural networks, it is used to find the optimal weights and biases of a network iteratively. Gradient descent is generally performed in the following manner:

- Initialize parameters (weights, biases, learning rate)
- Perform forward propagation
- Compute gradients of the loss function
- Perform backpropagation
- Update parameters

2.3.4 Types of Gradient Descent

1. Batch gradient descent

   This calculates the gradient of the cost function for the whole training dataset in order to make a single update to the weights of the network. This is computationally expensive and intractable when large data sets are involved. A mitigating factor here is that batch gradient descent guarantees the attainment of the global minimum for convex functions and local minimum for non-convex functions [48].

2. Stochastic gradient descent

   This computes the gradient of one randomly chosen example from the training data instead of using all examples as batch gradient descent does. This technique gives faster convergence but is erratic and requires many corrections. By reducing the learning rate in a controlled manner, stochastic gradient descent can exhibit a similar convergence pattern as batch gradient descent [48]. A recent paper (still under review) has claimed that SGD can find the global minimum of non-convex functions through a concept called star convexity [49]. This is interesting because it promises epochwise convergence which could reduce training time significantly.

3. Mini batch gradient descent

   Mini batch gradient descent is very similar to vanilla gradient descent discussed in point one above but the difference is that small batches of the training data are used instead of the whole data set. This means we have frequent gradient updates, more weight updates and therefore better convergence time.

2.3.5 Backward Propagation

Backward propagation is a technique for optimising ANNs that was first developed in the 1960's but only gained popularity at the end of the 1980's through an article by Rumelhart et al. [24, 50]. It is an algorithm used to iteratively compute gradients of the loss function in a ANN, which will in turn determine how weights are updated [3, 23, 24, 43]. This computation uses the gradient descent algorithm to find the best weights and this process is repeated until the optimal weights are learned. The word backward comes from the fact that the computation of gradients is done backwards after a process of forward propagation. Forward propagation is the process of transmitting an input x_i through the network and its parameters (weights, biases and activations) so that a prediction is made at the output layer. Using the loss function, the error between this prediction and the ground truth label is calculated, and then backpropagation will transmit the information about the error backwards in order to update the network's weights [23, 24, 43, 50].

Suppose we have an input vector

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}, \tag{2.13}$$

where m is the number of input features.

We can define the weight matrix W^l as

$$W^l = \begin{bmatrix} w^l_{11} & w^l_{12} & \dots & w^l_{1m} \\ w^l_{21} & w^l_{22} & \dots & w^l_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ w^l_{n1} & w^l_{n2} & \dots & w^l_{nm} \end{bmatrix}, \tag{2.14}$$

where each entry w^l_{jk} is the weight of the connection from the k^{th} neuron in the $(l-1)^{th}$ layer to the j^{th} neuron in the l^{th} layer. The bias vector b^l can be defined as:

$$b^l = \begin{bmatrix} b^l_1 \\ b^l_2 \\ \vdots \\ b^l_m \end{bmatrix}, \tag{2.15}$$

where each entry b^l_j is the bias in l^{th} layer. Similarly, we can define the activation vector a^l as:

$$a^l = \begin{bmatrix} a^l_1 \\ a^l_2 \\ \vdots \\ a^l_m \end{bmatrix}, \tag{2.16}$$

where each a^l_j is the activation in the l^{th} layer.

The activation a^l_j can be related to the previous activations in the $(l-1)^{th}$ layer with the equation

$$a^l_j = f\left(\sum_k^m w^l_{jk} a^{l-1}_k + b^l_j\right). \tag{2.17}$$

Having written the expressions for the weights, biases and activations compactly in equations 2.14, 2.15 and 2.16, we can now rewrite 2.17 as

$$a^l = f\left(W^l a^{l-1} + b^l\right). \tag{2.18}$$

Before calculating a^l we first need to calculate the weighted input of the neurons in layer l, which is usually denoted by z in the literature. The neuron-wise expression of z in layer l, is

$$z_j^l = \sum_k^m w_{jk}^l a_k^{l-1} + b_j^l, \tag{2.19}$$

and the more compact expression is given by

$$z^l = W^l a^{l-1} + b^l. \tag{2.20}$$

This means we can simplify equation 2.18 further by substituting the argument of f with z^l such that

$$a^l = f\left(z^l\right). \tag{2.21}$$

As mentioned earlier, before we can implement backpropagation, we must first forward propagate. The process of forward propagation involves the computation of $\mathbf{a}$, $\mathbf{W}$, $\mathbf{b}$ and $\mathbf{z}$ for all intermediate layers until the output layer, where the prediction of the input data point is made. The next step is then to compute the error associated with the prediction using the loss function [3, 23].

Suppose we have chosen the Mean Squared Error, denoted MSE, as our loss function. Let L denote the loss function defined by

$$L = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right)^2, \tag{2.22}$$

where n is the total number of instances in the training data, y_i the ground truth labels and $\hat{y}_i$ the predicted labels. Thus, for each instance x_i, of the training data we have

$$L_{x_i} = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right)^2, \tag{2.23}$$

and we can take the average of all L_{x_i}'s as the cost function

$$C = \frac{1}{n} \sum_{i=1}^{n} L_{x_i}. \tag{2.24}$$

An important assumption of any cost function is that it should be differentiable over the entire feature space. Backpropagation computes gradients of the cost function by taking partial derivatives $\frac{\partial C}{\partial w}$ and $\frac{\partial C}{\partial b}$ of C with respect to W and b. In particular, it performs element-wise partial derivatives of the form $\frac{\partial C}{\partial w^l_{jk}}$ and $\frac{\partial C}{\partial b^l_j}$. In order to compute these partial derivatives, we need to define an error term, E^l_j, where j is the j^{th} neuron in layer, l [3]. Define the error in the j^{th} neuron in layer, l, to be

$$E^l_j \equiv \frac{\partial C}{\partial z^l_j}. \tag{2.25}$$

Similar to before, we can rewrite equation 2.25 in a vectorised form, where E^l denotes the vector of errors in some layer, l. By implementing the backpropagation algorithm we will be able to compute the errors E^l for each layer l of the network. It is worth noting that the quantities

$$\frac{\partial C}{\partial w^l_{jk}} \qquad \text{and} \qquad \frac{\partial C}{\partial b^l_j} \tag{2.26}$$

are related to the weighted input, z^l_j, since it is a function of both the weight and bias as shown by equation 2.19. Having all these tools we can successfully implement backpropagation and update the weights of the network.

2.3.6 Vanishing and Exploding Gradients

One drawback of backward propagation is that it tends to "vanish" gradients. Vanishing gradients tend to occur when the weights of a NN are too close to zero and they become successively smaller with each update [51, 52]. This causes the model to update marginally and take too long to converge. For any activation function, the absolute change in weights will get smaller and smaller as we back propagate and the earlier layers are the slowest to train [53, 54]. The exploding gradients problem is the opposite of vanishing gradients. It is caused by an accumulation of error gradients which cause large gradients and then large weight updates [55]. The explosion happens when you have exponentially increasing weights that causes the model not to learn and ultimately becomes unstable [53, 55].

2.4 Decision Trees[2]

A decision tree is a rule based algorithm that classifies data by splitting it recursively based on its features. Well known decision tree algorithms include the ID3, ASSISTANT and C4.5. Decision trees organise data on the basis of a rule from the root node to the leaf node (or terminal node) which makes the final classification. This process works as follows: first test the feature in the root node and then proceed down the tree through the branches to the value that matches the feature [6]. Figure 2.5 shows an example of a tennis player contemplating tennis practice on a day with different weather conditions.

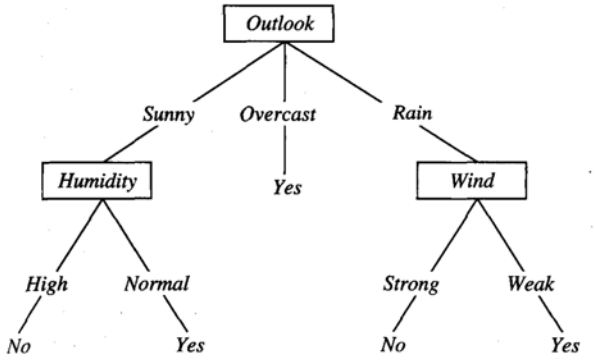


Figure 2.5 An illustration of a decision tree algorithm [6]

2.4.1 Bootstrapping and Bagging

Bootstrapping refers to random sampling with replacement where multiple models are tested on resamples with different characteristics such as the mean, median and variance. Bagging solves the overfitting problems of decision trees by creating multiple resamples and multiple versions of a classifier. This means that bagging makes use of bootstrapping to build new training data sets from the original training data set. Bagging is particularly good at producing varied classifiers by taking advantage of unstable learning algorithms to improve accuracy. It then tests all these classifiers on each of the resamples similar to a multiple hypothesis testing problem [56].

[2]The discussion in this section is informed by a discussion that appears in T.C Gaelejwe, *"Supresing Fake Missing Transverse Energy using Multivariate Analysis with the ATLAS Detector"*, University of the Witwatersrand, 2018.

2.4.2 Information Gain and Entropy

Decision trees are constructed from the top-down. We first have to establish which attribute must be tested at the root node and then make the same determination for all other attributes to be tested at each node going down the tree. In general, the attribute with the most discriminating power should be used at the root node. This is where the concepts of information gain and entropy become relevant. We define and discuss these concepts in the next paragraphs.

"Entropy is a property that characterises the purity or impurity of an arbitrary collection of instances. Mathematically, it is formulated as follows:
Let C be an arbitrary collection of examples with both negative and positive examples of some target concept. The entropy of C relative to this Boolean classification is

$$Entropy(C) = -p_{\oplus}\, log_2\, p_{\oplus} - p_{\ominus}\, log_2\, p_{\ominus} \tag{2.27}$$

where $p_{\oplus}$ and $p_{\ominus}$ are proportions of positive and negative examples in C respectively" [6, 22].

"Information theory states that entropy is a property that specifies the minimum number of pieces of information required to determine the classification of an arbitrary element of C. For cases where the target variable has to assume more than two different values, the definition of entropy is stated as

$$Entropy(C) \equiv \sum_{j=1}^{d} -p_j\, log_2\, p_j \tag{2.28}$$

where p_j is the proportion of C belonging to class j and d is the number of different values the attribute can assume" [6, 22].

"Information gain is a metric used to measure the effectiveness of an attribute in classifying the input data. It is the expected reduction in entropy resulting from separating the examples based on this attribute" [6, 22]. "The information gain of some attribute K relative to a collection of examples C can be formulated mathematically as

$$G(C,K) \equiv Entropy(C) - \sum_{v \in \Omega} \frac{\mid C_v \mid}{\mid C \mid} Entropy(C_v) \tag{2.29}$$

where Ω (= values in K) is the set of all possible values for attribute K and C_v is the subset of C for which attribute K has value v. $G\,(C,K)$ is the expected reduction in entropy caused by knowing the value of attribute K" [6].

2.4.3 Boosted Decision Trees

Boosting can be thought of as a form of bagging where more attention is assigned to poorly performing aspects of the decision tree classifier. BDTs are an ensemble of learners where poorly modelled instances are given a weight and returned as training data for the next learner [56, 57]. Decision trees are well known for being fast, scaling easily to large data sets and being robust against outliers, missing values as well as irrelevant input features. Decision trees, like many other algorithms, have their own drawbacks, which includes overfitting. This is because larger trees tend to be unstable and the piecewise approximation of smaller trees [58, 59].

As a result, a method called boosting was introduced to the improve the robustness of decision trees and remedy these challenges. The strengths of BDTs come from its learning speed, the fact that its parameters are easily tuned and its insensitivity to scale [60]. The main weakness of BDTs is overfitting, but this can be corrected by performing cross-validation which is computationally expensive in large data sets. There are various ways of boosting weak learners and below we list some well known boosting examples.

1. Gradient Boosting (GBoost)

   - "This makes use of the gradient descent algorithm to find new learners and has two main parameters, i.e. learning rate and number of iterations. Gradient boosting is invariant under monotone transformations of inputs" [60].

2. Adaptive Boosting (AdaBoost)

   - "This algorithm boosts weak decision learners by adding their probabilistic predictions and adapts to the errors of the weak learners. It requires no

prior knowledge of the accuracies of the previous learners" [58].

3. eXtreme Gradient Boosting (XGBoost)

    - "This boosting technique is very popular in Kaggle competitions where it usually outperforms many other models. XGBoost scales very well and it has been optimized for high dimensional data which is characterized by sparsity [61]. The XGBoost algorithm takes advantage of computing architectures such as distributed computing to increase performance" [61].

Chapter 3

Research Methodology

In this study, we have signal events with real missing transverse energy and background events with fake missing transverse energy. We consider two types of learning paradigms, fully supervised and weakly supervised learning. The choice of ML algorithms are Gradient Boosted Decision Trees, which we will refer to as GBDT, and DNNs respectively. As mentioned in section 1.3, we want to compare supervised GBDT with supervised DNNs first and then compare supervised GBDT with weakly supervised GBDT.

3.1 Research Design

This research aims to develop an experiment where two ML algorithms are trained through a fully supervised and weakly supervised learning paradigm. This experiment can be described as follows, (i) explore and pre-process the data (ii) train two algorithms by full supervision with a labelled data set and (iii) train one algorithm by weak supervision using a data set with inaccurate labels. The reason for training by full supervision is so that we can classify SM signals (whose labels are already known) with a much higher precision than traditional physics techniques could. For example, new data on the SM Higgs boson can provide new information which allows us to measure its properties with higher accuracies using ML. On the other hand, we train by weak supervision because we do not have accurate labels for BSM signals such as the Madala Boson. Weak supervision allows us to exploit the usefulness of unlabelled or inaccurately labelled instances of data. This method is justified because often in

HEP, the signal is unknown.

3.2 Data

All data used here are Monte Carlo (MC) simulations of proton-proton collisions with the LHC running at a centre-of-mass energy of $\sqrt{s} = 13$ TeV. In this research, we will use the word *data* to refer to these MC simulations. We make this known to avoid confusion because the convention in HEP is to refer to real data from the experiment as data and simulated data as MC samples. This data is made available to us by the Institute for Collider Particle Physics at the University of the Witwatersrand through its affiliation with CERN.

Before making choices about suitable algorithms, we must begin by exploring the data, understand it and organise it in a format most suited to our purposes. A notable characteristic of the data is that it is highly class imbalanced, where the background constitutes the majority of the data set and the signal constitutes only a small fraction (usually less than one percent) of that data set. There is also a strong overlap between the two classes due to a noise factor called pile-up that pollutes the signal giving it background-like properties making the discrimination more difficult. Given this information, the first step of the data exploration was to visualize the univariate distributions for all the features in the data set. These visualizations are shown by Figures A.1, A.2 and A.3 in Appendix A.1 and A.4, A.5 and A.6 in Appendix A.2. The second step was to check for correlations between the features. We make use of Pearson's correlation for this task. The correlation matrices are shown by Figures A.7a and A.7b in Appendix A.

For the weakly supervised BDT, the data is organized as follows. We have two samples for training, i.e. sample 1 and sample 2. Sample 1 contains sideband or pure background (which is defined as $\gamma\gamma$ events with the invariant mass outside $240 - 280$ GeV) and sample 2 contains a mixture of sideband and signal data in the mass region of $240 - 280$ GeV. The sideband data is well known so we have ground truth labels for it but the signal is not very well known or sometimes we do not know it at all so its labels are poor. For this reason, we implement inaccurate supervision since it is the

most suitable learning paradigm given these labels.

For the DNNs, the data is corrected for class imbalance before training. We make use of the Synthetic Minority Oversampling TEchnique (SMOTE) for this task [62, 63, 64]. For the GBDT model, this is not necessary because it can deal with class imbalances relatively well.

We have three data sets as detailed in Table 3.1. We also consider three missing transverse energy significance ($S_{E_T^{miss}}$) categories for each data set which implies that we will have nine data sets to train. Table 3.2 shows these $S_{E_T^{miss}}$ categories.

Table 3.1 Data samples used

Data	**Description**
R21_A400_Z_vv_H250_yy_comb	Gluon fusion sample with Z decaying to two neutrinos and H to two photons containing 16631 events.
R21_ggZH125_comb	Signal sample with real missing energy from neutrinos containing 37236 events.
R21_HHDMmr275mx60br50_comb	Heavy scalar sample with 49998 events.
R21_Background_comb	Sample of combined background events with 4167613 events.

Table 3.2 Table showing $S_{E_T^{miss}}$ categories.

Category	**Pre-selection**
Low $S_{E_T^{miss}}$	$S_{E_T^{miss}} > 2.5$ & $S_{E_T^{miss}} < 3.5$ & $N_{jet} \geq 1$
Int $S_{E_T^{miss}}$	$S_{E_T^{miss}} > 3.5$ & $S_{E_T^{miss}} < 5.5$ & $N_{jet} \geq 1$
High $S_{E_T^{miss}}$	$S_{E_T^{miss}} > 5.5$ & $N_{jet} \geq 1$

After the pre-selection cuts, each data set in Table 3.1 is reduced to just a few thousand events. Specifically, we are left with the data sizes shown in Table 3.3.

Table 3.3 Number of events in each data set

A400Z		
Category	GBDT	DNN
Low S_{ET}^{miss}	1767	1767
Int S_{ET}^{miss}	3866	3866
High S_{ET}^{miss}	3840	3840

ggZH125		
Category	GBDT	DNN
Low S_{ET}^{miss}	2744	2744
Int S_{ET}^{miss}	1968	1968
High S_{ET}^{miss}	5442	5442

275mx60		
Category	GBDT	DNN
Low S_{ET}^{miss}	5720	5720
Int S_{ET}^{miss}	8063	8063
High S_{ET}^{miss}	5311	5311

Background		
Category	GBDT	DNN
Low S_{ET}^{miss}	193964	*
Int S_{ET}^{miss}	71640	*
High S_{ET}^{miss}	23839	*

In Table 3.3 above, the subtable with information about the background data sizes has asterisks (*) for the DNN because the actual data sizes are identical to that of the corresponding signal data sets in all S_{ET}^{miss} categories. These data sizes are a result of applying the SMOTE algorithm when correcting for class imbalance. Before training the data sets are split 70:30 for training and validation. After validating an independent test data set is used to test the model's performance.

As mentioned earlier we train the GBDT and DNN on nine data sets. Figure 3.1 shows a schematic of how the algorithms are trained. Each signal data set is combined with a background data set in its corresponding S_{ET}^{miss} category. In particular, the heavy scalar (275mx60) signal data set will have three categories as shown by the red boxes in the diagram. Similarly, the neutrino and gluon fusion signal samples will have three categories as shown by the green and black boxes in the diagram.

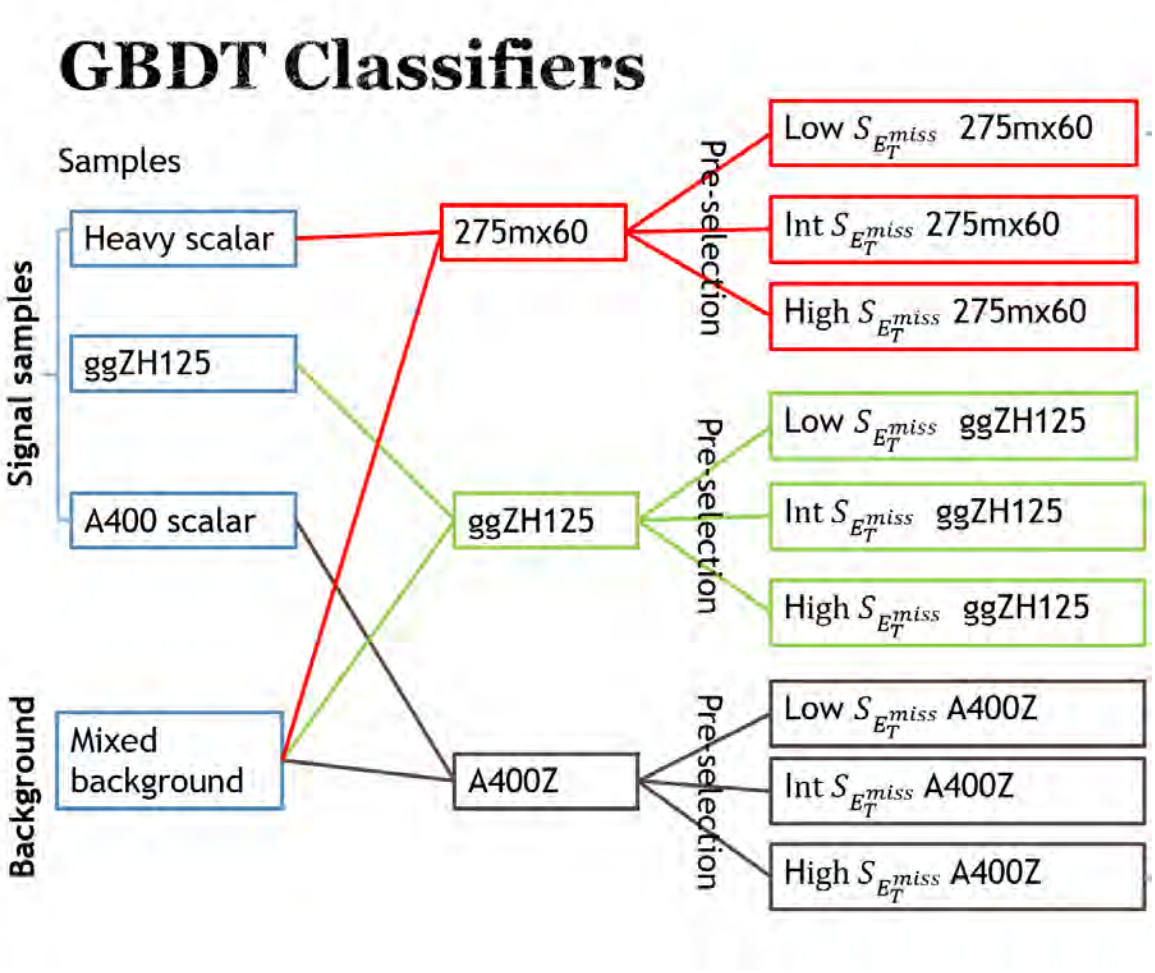


Figure 3.1 An illustration of the data being split into different $S_{E_T^{miss}}$ categories

3.3 TensorFlow

TensorFlow[1] is an open source ML library often used for deep learning tasks. It is one of the most popular ML libraries currently. We train our DNN model using the high level Keras API integrated into TensorFlow.

3.4 Features

The following features are the main input variables for the models trained in this research project. The visualisations for each of these variables can be found in Appendix A.

- "**Photon pointing Vertex** $\sum p_T^2$.

[1] https://www.tensorflow.org/

- **NPV:** Number of vertex reconstructed in an event.
- $JVT^{J_n}_{corr}, n = 1, 2, 3$**:** Corrected jet vertex fraction is the ratio of track p_T and p_T of jets in the calorimeter.
- $R^{J_n}_{p^T}, n = 1, 2, 3$**:** Scalar sum of jet primary vertex track and jet track transverse momentum, p_T, associated with the diphoton system.
- $\Delta\phi\,(J1, E_T^{miss})$**:** Angular distance between jets system and missing transverse energy
- $\Delta\phi\,(\gamma\gamma, E_T^{miss})$**:** Angular distance between diphoton system and missing transverse energy
- $\Delta\phi\,(\gamma\gamma, jet1)$**:** Angular distance between diphoton system and leading jet1
- $\Delta\phi\,(softjets, E_T^{miss})$**:** Angular difference between soft jets system (jets with p_T less than 30 GeV) and missing transverse energy
- $\Delta\phi\,(forjets, E_T^{miss})$**:** Angular distance between forward jets (jets outside the central region of the detector $\eta \leq 2.4$) and missing transverse energy
- **N_j_central:** Number of central jets".

3.5 Hyperparameters

Hyperparameters are parameters of the model that can be adjusted (randomly or algorithmically) to find the optimal configuration of the model that results in the highest performance and robustness. For the GBDT, we tune the following hyperparameters: number of trees, maximum depth and learning rate. For the DNN we tune the number of nodes/neurons, number of hidden layers, learning rate and activation function. For the DNN, we make use of Xavier's weight initialisation where we multiply randomly generated weights by Equation 2.2 [45]. Initially, we had used He's weight initialization, however, it caused some overfitting despite evidence in the literature suggesting it is most suited to ReLU activated layers [44]. In the end, we settled on Xavier's technique. The hyperparameters for both the GBDT and DNN are tuned using the GridSearch function from scikit-learn. The optimal hyperparameters are listed in Tables 3.4, 3.5 and 3.6.

Table 3.4 Table showing GBDT hyperparameters.

Hyperparameter	**Optimal Setting**
Number of trees	800
Learning rate	0.1
Maximum depth	2

Table 3.5 Table showing DNN hyperparameters.

Hyperparameter	**Optimal Setting**
Optimiser	Adaptive Momentum
Loss Function	binary cross entropy
Batch size	100
Initial learning rate	0.001

Table 3.6 DNN Model Architecture

Input Layer		
Layer	Nodes	Activation
1	16	Leaky ReLU

Hidden Layers		
Layer	Nodes	Activation
2	128	Leaky ReLu
3	128	Leaky ReLu
4	128	Leaky ReLu
⋮	128	Leaky ReLu
8		
9	128	Leaky ReLu
10	48	Leaky ReLu

Output		
Layer	Nodes	Activation
11	1	Sigmoid

3.6 Performance Metrics

We make use of two performance metrics to gauge and analyse the performance of the models trained. These are explained in detail and contextualised in the sense of HEP data analysis below.

3.6.1 Receiver Operating Curves

"The Receiver Operating Curves (ROC) show the discrimination power of a binary classifier. A ROC curve plots the True Positive Rate (TPR) against the False Positive Rate (FPR) where TPR is the ratio of correctly predicted positive data points to all positive data points in the data set and FPR is the ratio of incorrectly predicted positive data points to all negative data points in the data set. Essentially, a ROC curve shows the trade-off between true positives and false positives. A very good prediction would lie in the upper left corner of the canvas and a poor prediction would lie in the bottom left corner of the canvas (for the kind of ROC curves we consider here. See Figures 4.4 and 4.5). ROCs usually have a curve that runs along the x-y axis. This is called the No Improvement or Random Guessing curve. Any prediction along this curve yields signal acceptance and background rejection which is no better than randomly guessing the class of a given data point". Mathematically,

$$TPR = \frac{\text{True Positives}}{\text{All Positives}} = \frac{\text{True Positives}}{\text{True Positives + False Negatives}} = 1-\text{False Negative Rate} \tag{3.1}$$

and

$$FPR = \frac{\text{False Positives}}{\text{All Negatives}} = \frac{\text{False Positives}}{\text{False Positives + True Negatives}} = 1-\text{True Positive Rate.} \tag{3.2}$$

In the context of HEP, the ROC plots background rejection efficiency on the y-axis against signal acceptance efficiency on the x-axis. This makes sense because TPR is signal efficiency and FPR is background efficiency. By taking $1 - FPR$, we obtain background rejection, and so our ROC will run from the top left corner of the canvas to the bottom right corner of the canvas. In this case, the region in the upper right corner will be the best in terms of background rejection and signal efficiency. This

can be thought of as a reflection of the usual ROC curve about the $x = 0$ line. Signal efficiency and background rejection can be expressed mathematically in terms of TPR and FPR as given by equations 3.3 and 3.4.

$$\text{Signal efficiency} = TPR = \frac{\text{True Positives}}{\text{All Positives}} = \frac{\text{True Positive}}{\text{True Positives + False Negatives}} \tag{3.3}$$

$$\text{Background Rejection} = 1 - FPR = 1 - \frac{\text{False Positives}}{\text{False Positives + True Negatives}}\,. \tag{3.4}$$

3.6.2 Confusion Matrix

A confusion matrix is tool that provides a summary of prediction results on a model's performance in matrix form. It shows the count of all correct and incorrect predictions for each class. For a binary classification problem this will be a 2×2 matrix. Typically, the top left entry, c_{11}, is the true positive count, the top right entry, c_{12}, is the false negative count, the bottom left entry, c_{21}, is the false positive count and the bottom right entry, c_{22}, is the true negative count. This metric provides us with a lot more information about a model's performance compared to accuracy and AUC. Essentially, it tells the user the extent to which the model is confused when making classifications. We can define the components of the matrix as follows:

Positive (P) : Event is positive (i.e. signal event)
Negative (N) : Event is not positive (i.e. background event)
True Positive (TP) : A positive event is predicted to be positive (i.e. signal event is correctly predicted as signal event)
True Negative (TN) : A negative event is predicted to be negative (i.e. background event is correctly predicted as background event)
False Negative (FN): A positive event is predicted to be negative
False Positive (FP) : A negative event is predicted to be positive

Table 3.7 A typical confusion matrix

	Signal Predictions	**Background Predictions**
Actual Signal	TP	FN
Actual Background	FP	TN

1. Classification Accuracy

   Classification accuracy is the ratio of correct predictions (positive and negative) to the total number of predictions made. It can be expressed mathematically as

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN} \tag{3.5}$$

   The main problem with classification accuracy is that it is not robust against high class imbalances. That is, the model can achieve an accuracy of 95% but 95 events out of every 100 could be background.

2. Recall

   An important metric from the confusion matrix is Recall. It is the ratio of all correctly classified positive events to all positive (signal) events. Ideally, we want to have a high recall which would mean the model is recognising the signal very well.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{3.6}$$

3. Precision

   Precision is the ratio of all correctly classified positive (signal) events to the total number of events predicted as positive (both correct and false). As the name suggests, this metric tells us how precise the model is in predicting positive (signal) events. It is defined as

$$\text{Precision} = \frac{TP}{TP + FP} \tag{3.7}$$

Chapter 4

Results

4.1 Supervised Learning Results

Below are GBDT output plots showing the performance of the supervised BDT in classifying background and signal events on the test data set. Figures 4.1, 4.2 and 4.3 show the GBDT output for the low, intermediate and high $S_{E_T^{miss}}$ categories respectively. The GBDT output plots the number of normalised events on the y-axis against the predicted probability estimate on the x-axis. The corresponding ROC curves for the GBDT and DNN models are shown by Figures 4.4 and 4.5. Since we have three different $S_{E_T^{miss}}$ categories for each of the three data sets, we have effectively trained nine different models and we should expect nine plots in total.

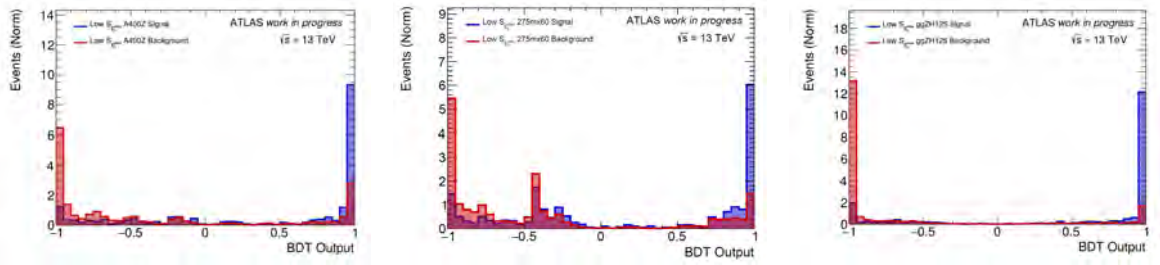


Figure 4.1 Fully supervised BDT output for the low $S_{E_T^{miss}}$ category

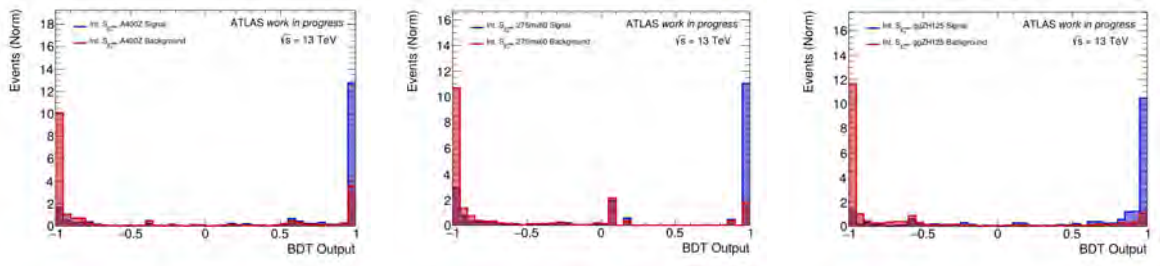


Figure 4.2 Fully supervised BDT output for the intermediate $S_{E_T^{miss}}$ category

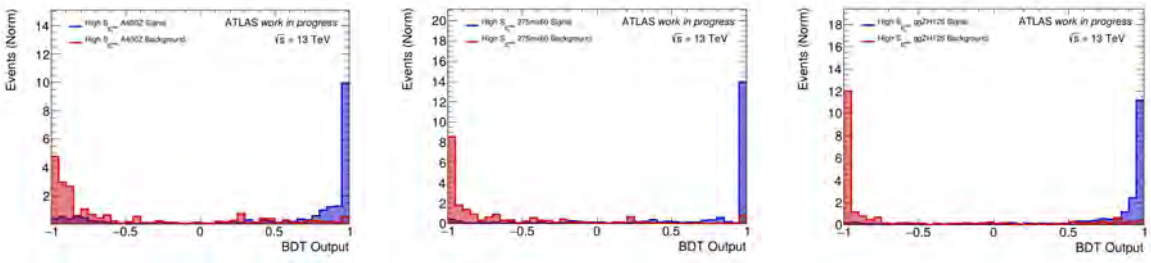


Figure 4.3 Fully supervised BDT output for the high $S_{E_T^{miss}}$ category

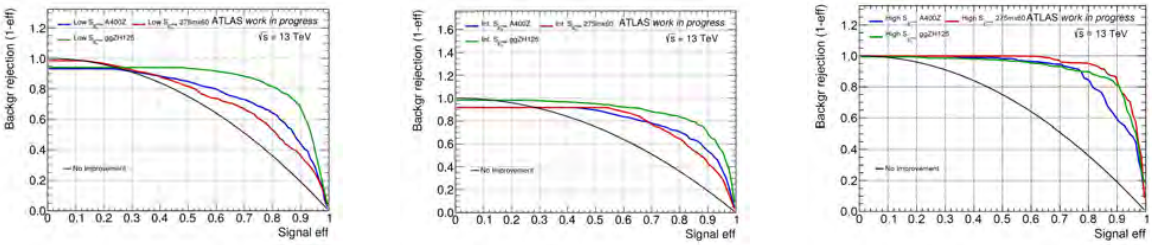


Figure 4.4 Supervised GBDT ROC curves for Low, Intermediate and High $S_{E_T^{miss}}$

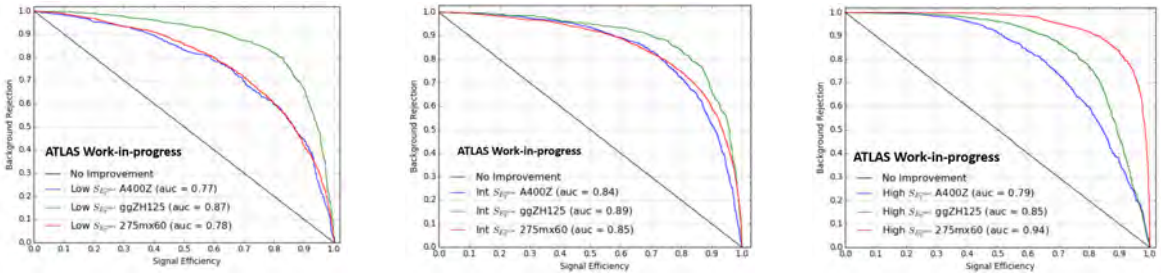


Figure 4.5 Supervised DNN ROC curves for Low, Intermediate and High $S_{E_T^{miss}}$

Table 4.1 below shows a comparison of the AUC scores for each supervised model. This table consists of three side-by-side subtables with the heading representing the data set used for training (written in bold text).

Table 4.1 Comparison of Area Under the Curve for BDT and DNN

A400Z		
Category	BDT	DNN
Low $S_{E_T^{miss}}$	0.733	0.772
Int $S_{E_T^{miss}}$	0.805	0.843
High $S_{E_T^{miss}}$	0.860	0.797

ggZH125		
Category	BDT	DNN
Low $S_{E_T^{miss}}$	0.780	0.873
Int $S_{E_T^{miss}}$	0.822	0.875
High $S_{E_T^{miss}}$	0.917	0.852

275mx60		
Category	BDT	DNN
Low $S_{E_T^{miss}}$	0.733	0.782
Int $S_{E_T^{miss}}$	0.779	0.854
High $S_{E_T^{miss}}$	0.938	0.942

4.2 Weakly Supervised Learning Results

Below are BDT output plots showing the performance of the weakly supervised model in classifying background and signal events on the same test data used for the fully supervised GBDT. Figures 4.6, 4.7 and 4.8 show the GBDT output for the low, intermediate and high $S_{E_T^{miss}}$ categories respectively. As in section 4.1, the GBDT output plots the number of normalised events on the y-axis against the predicted probability estimate on the x-axis. The corresponding ROC curves for the fully supervised and weakly supervised GBDT models are shown by Figures 4.9 and 4.10.

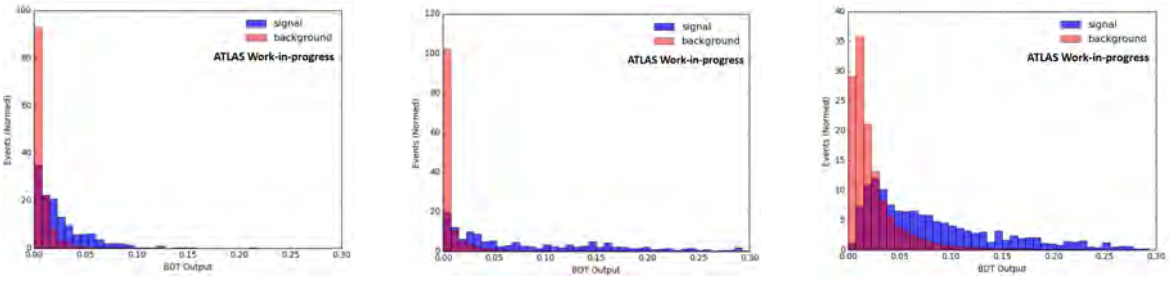


Figure 4.6 Weakly supervised GBDT output for the low $S_{E_T^{miss}}$ category

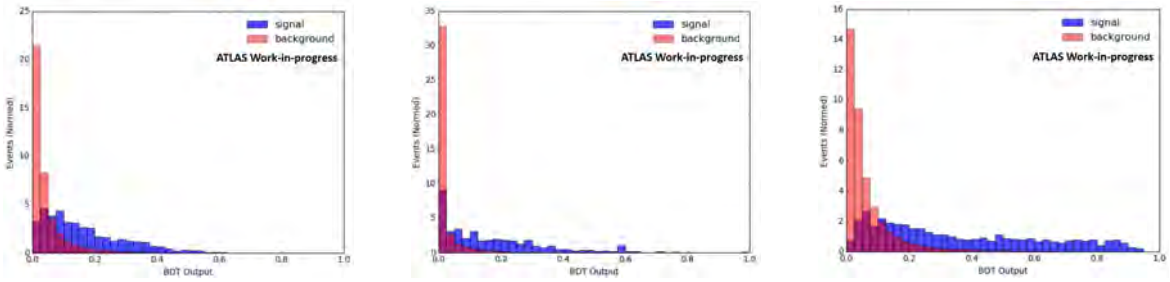


Figure 4.7 Weakly supervised GBDT output for the intermediate $S_{E_T^{miss}}$ category

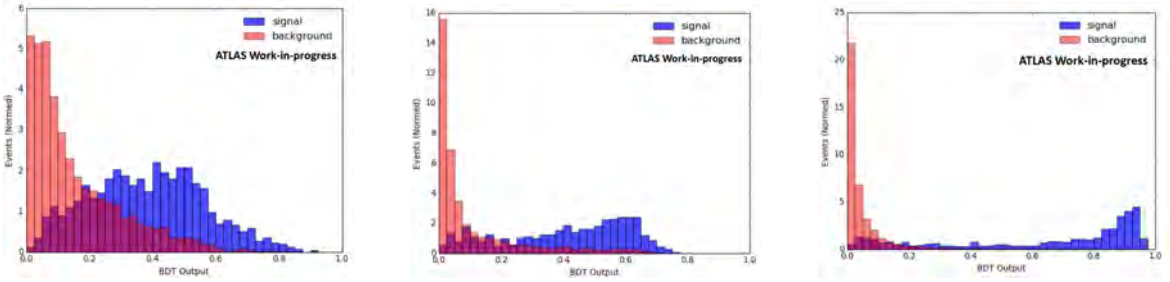


Figure 4.8 Weakly supervised GBDT output for the high $S_{E_T^{miss}}$ category

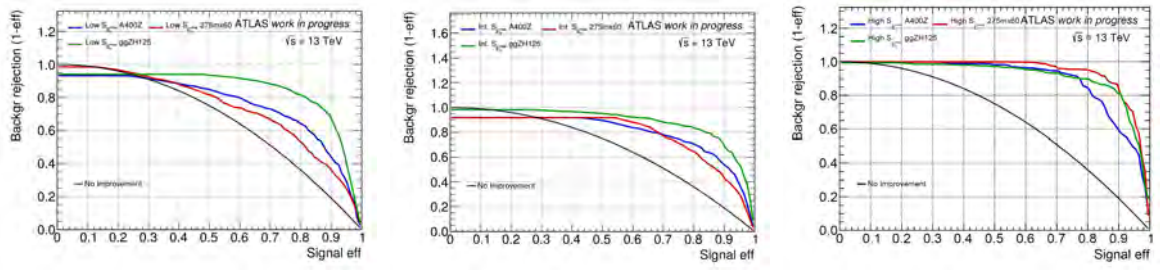


Figure 4.9 Fully supervised GBDT ROC curves

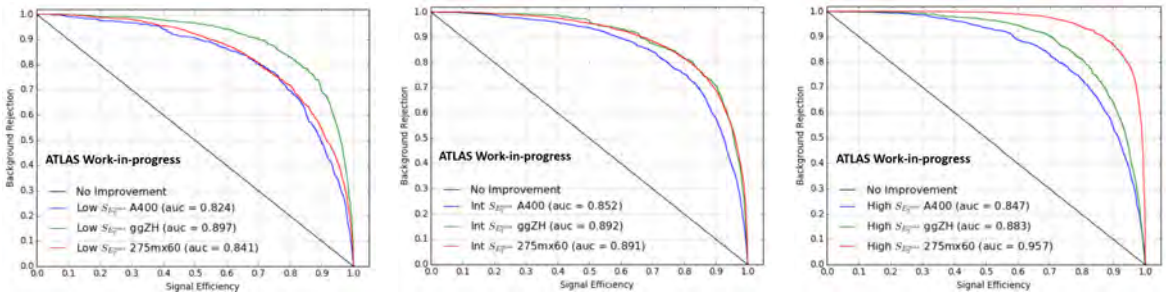


Figure 4.10 Weakly supervised GBDT ROC curves

Table 4.2 below shows a comparison of the AUC scores for each supervised model. This table consists of three subtables with the heading representing the data set used for training and testing (written in bold text).

Table 4.2 Comparison of Area Under the Curve for fully supervised ($GBDT_F$) and weakly supervised ($GBDT_W$) GBDT

A400Z		
Category	$GBDT_F$	$GBDT_W$
Low $S_{E_T^{miss}}$	0.733	0.824
Int $S_{E_T^{miss}}$	0.805	0.852
High $S_{E_T^{miss}}$	0.860	0.847

ggZH125		
Category	$GBDT_F$	$GBDT_W$
Low $S_{E_T^{miss}}$	0.780	0.879
Int $S_{E_T^{miss}}$	0.822	0.892
High $S_{E_T^{miss}}$	0.917	0.883

275mx60		
Category	$GBDT_F$	$GBDT_W$
Low $S_{E_T^{miss}}$	0.733	0.841
Int $S_{E_T^{miss}}$	0.779	0.891
High $S_{E_T^{miss}}$	0.938	0.957

4.3 Discussion

Based on Table 4.1, the fully supervised results show that the DNN outperforms the GBDT in terms of AUC in general. For the low S_{ET}^{miss} category, the DNN outperforms the GBDT by at least 5% across all data sets and for the intermediate S_{ET}^{miss} category, the DNN outperforms the GBDT by at least 4%. In the high S_{ET}^{miss} category, the GBDT outperforms the DNN on the A400Z and ggZH125 data sets but the DNN does better on the 275mx60 data set. Based on the confusion matrices in Appendix B Figures B.1, B.2 and B.3, the DNN has a decent recall (i.e. diagonal values of the matrices) for both the signal and background. This indicates that the model is robust and fairly accurate. Another set of plots that assist our analysis of the DNN's performance are the loss function plots which are given by Figures C.1, C.2 and C.3 in Appendix C. Ideally, we want the loss function to decrease asymptotically close to zero, however, we can accept training and validation loss functions that decrease together without a significant deviation from each other.

Looking at the results for the fully supervised GBDT and weakly supervised GBDT in section 4.2, it is clear that the weakly supervised model outperforms the supervised model in terms of AUC. Based on the GBDT output plots given by Figures 4.6, 4.7 and 4.8 it is evident from the observed overlap that the weakly supervised model misclassifies a lot. Also another noticeable feature on the plots is that the probability estimates for the signal (in blue) are mostly below 0.5 meaning that the model predicts the signal as background quite often. Correctly predicted signal events would have probability estimates above 0.5. This is in contrast to the GBDT output plots for the supervised model in section 4.1 where the signal probability estimates are mostly above 0.5 and there is a separation between the blue and red distributions. This, however, does not discount the fact that the supervised GBDT also misclassifies to some extent.

The misclassification of the weakly supervised model is also confirmed by the confusion matrices in Appendix B Figures B.4, B.5 and B.6. If we look at the diagonal entries which represent the recall of the model, we immediately see that the recall is excellent for the background but terrible for the signal. This is an important observation because it tells us that the model is often confused when classifying the signal and also reminds us that the AUC can sometimes be misleading.

Chapter 5

Conclusions & Future Work

5.1 Conclusions

Based on the ROC curves for GBDTs and DNNs in the Low, Intermediate and High $S_{E_T}^{miss}$ categories (corresponding to Figures 4.4 and 4.5), the DNNs performed relatively better than the GBDTs. This result is important since we had initially sought to investigate which of the two algorithms would perform best for our data sets. Table 4.1 shows the AUC scores for all these categories. It is worth noting, however, that the DNN outperforms the GBDT marginally and given the fact that the DNN takes longer to train, one might choose the GBDT without significantly compromising performance.

For the weakly supervised study the trained GBDT produces very good AUC scores but shows signs of high misclassification of the signal on the confusion matrix. That is, the weakly supervised model has poor recall for the signal. Figure 4.10 shows the ROC plots for all three data sets in the respective $S_{E_T}^{miss}$ categories using the weakly supervised model. Given these results, we can only make the conclusion that the weakly supervised GBDT was not able to match the performance and robustness of the fully supervised GBDT for the particular data sets and features we used. It is possible that a different algorithm with different features could yield a more positive result.

5.2 Future Work

For future work, an implementation of the Classification Without Labels technique (CWoLA) could be useful given its strong similarity with the technique used here for the weakly supervised learning paradigm [33]. In fact, the only difference between CWoLA and the technique used here is that the CWoLA uses two mixed samples for training whereas our technique uses one mixed sample and one unmixed sample. The CWoLA technique provides a rigorous mathematical proof for its claim that any supervised algorithm can be trained by weak supervision to produce comparable results to full supervision, yet the authors make no assertions about this being true for any training data set. This is especially important since not all algorithms are robust against high class imbalances.

The problem we have considered in this study is essentially an anomaly detection problem. Taking into consideration the structure of the data, it may be worthwhile exploring a one class Support Vector Machine or a Fuzzy K-Means clustering algorithm. A one class SVM could be trained on just the majority class (i.e. the background) so that it learns to recognize it very well and any instances of the minority class (i.e. signal) will be classified as an anomaly [65].

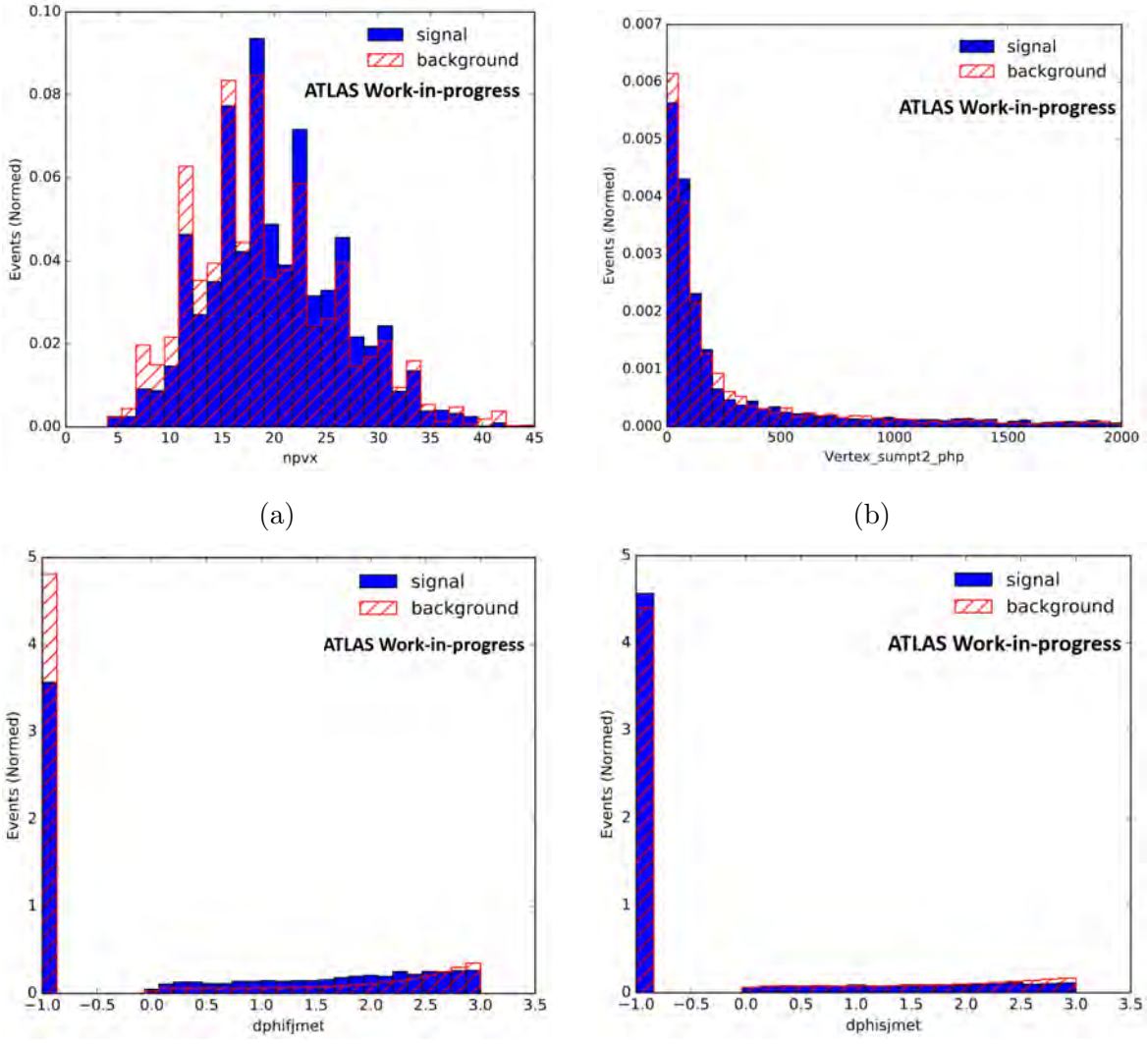
A fuzzy k-means clustering algorithm can also be used considering that there is a strong overlap between the two classes of the data and most instances cannot be easily assigned to one group or cluster. With fuzzy clustering a data point can possibly belong to more than one cluster and membership is determined by likelihood. For each data point, a likelihood score will be generated to establish which cluster does it belong to based on a predefined threshold [66].

Another technique that can be considered for training the weakly supervised model is to extend the problem from binary to multiple classification. Specifically, a three class problem where the first class corresponds to sideband, the second class pure signal and the third class a mixture of sideband and signal. The aim here will be to extract the signal directly from the weakly supervised model. The idea is that when the model assigns low probability estimates for the sideband and the mixture, it implies that the event under consideration has a higher probability estimate of being a signal.

Appendix A

Univariate Distributions

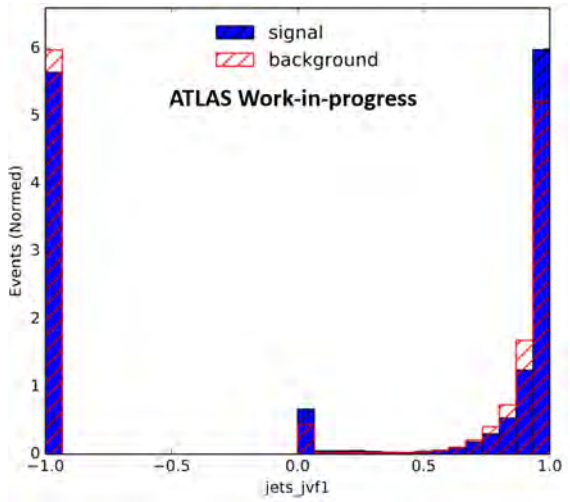
A.1 Fully Supervised Features



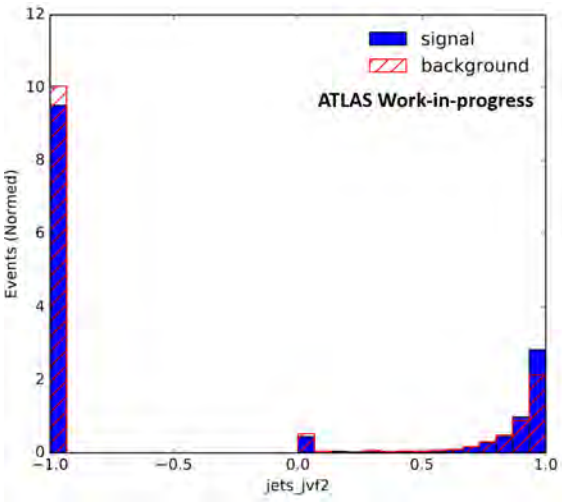
(a) (b)

(c) (d)

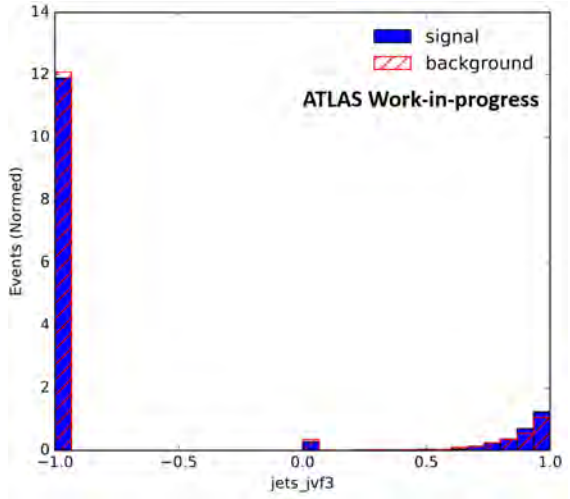
Figure A.1 Fully Supervised 1D Distributions



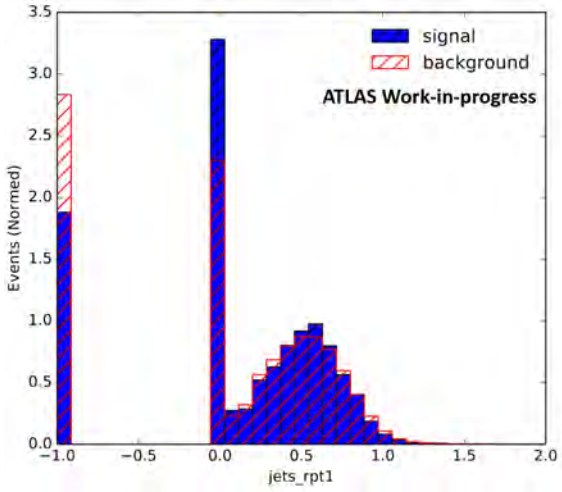
(a)



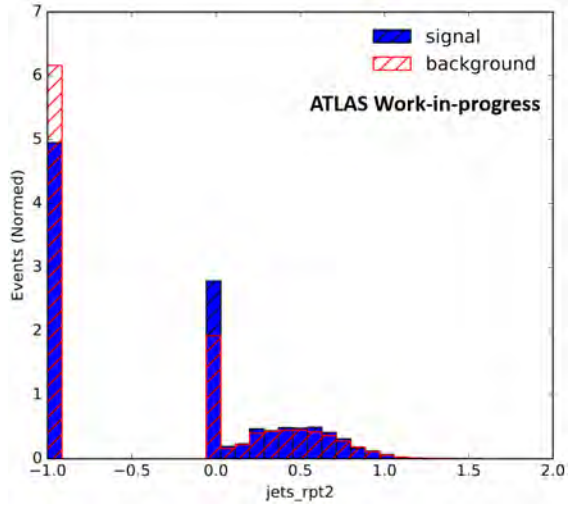
(b)



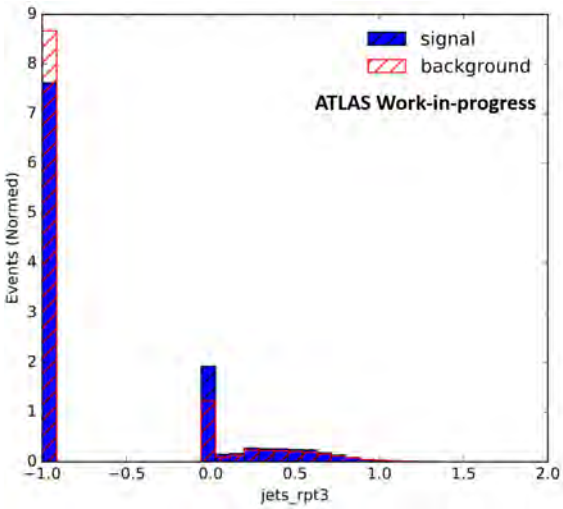
(c)



(d)



(e)



(f)

Figure A.2 Fully Supervised 1D Distributions

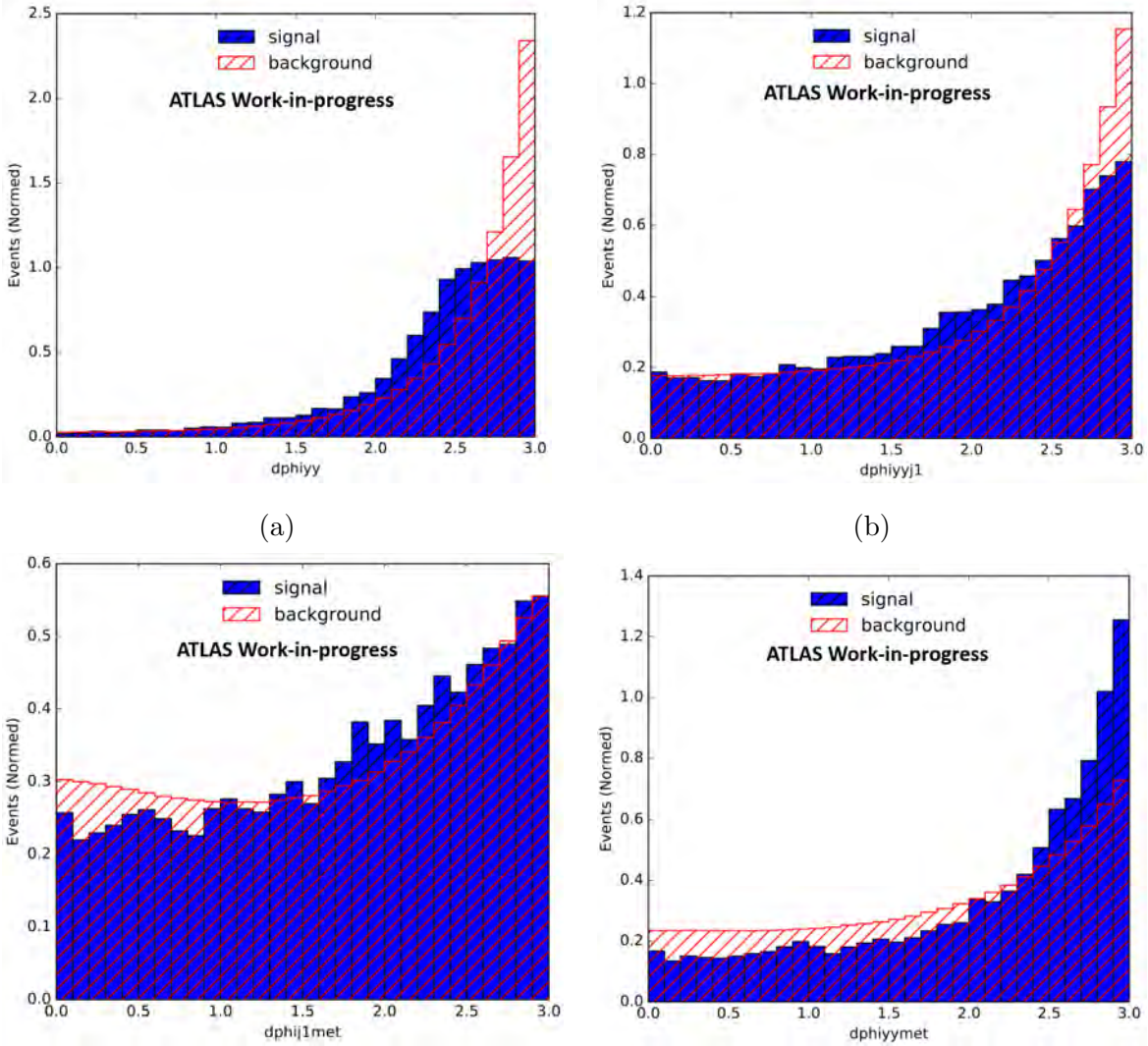
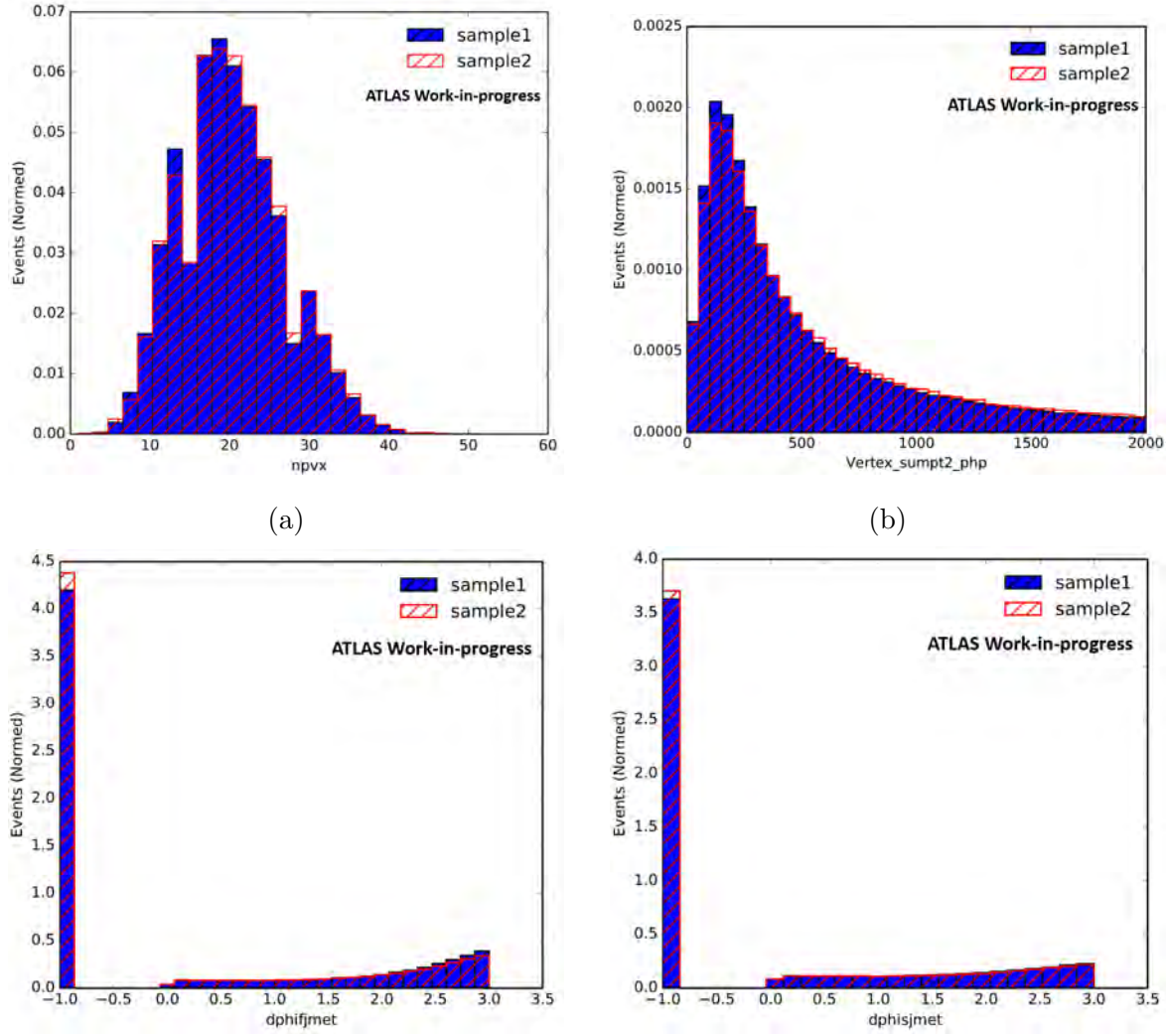


Figure A.3 Fully supervised 1D Distributions

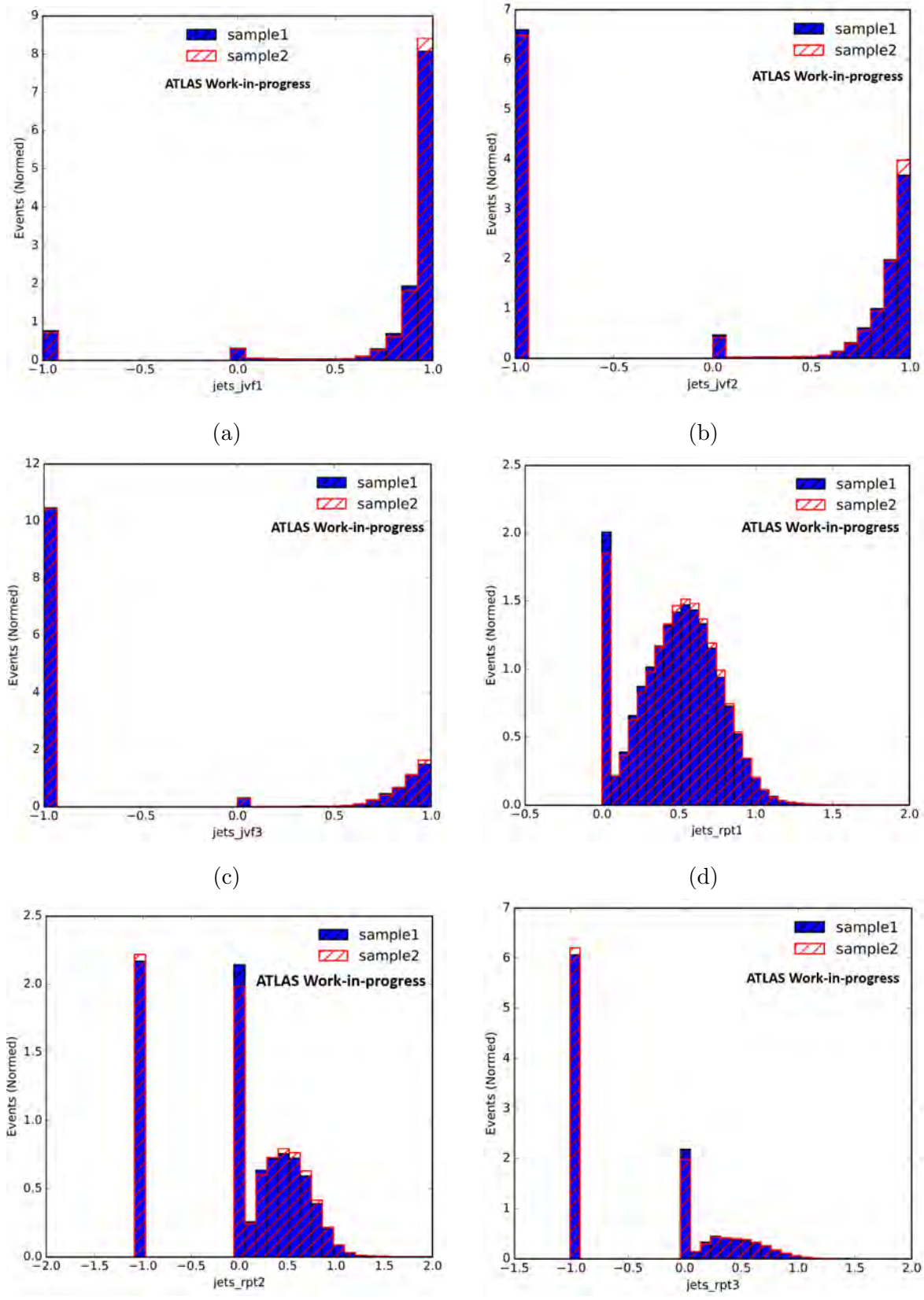
A.2 Weakly Supervised Features



(a) (b)

(c) (d)

Figure A.4 Weakly supervised 1D Distributions



(a) (b) (c) (d) (e) (f)

Figure A.5 Weakly supervised 1D Distributions

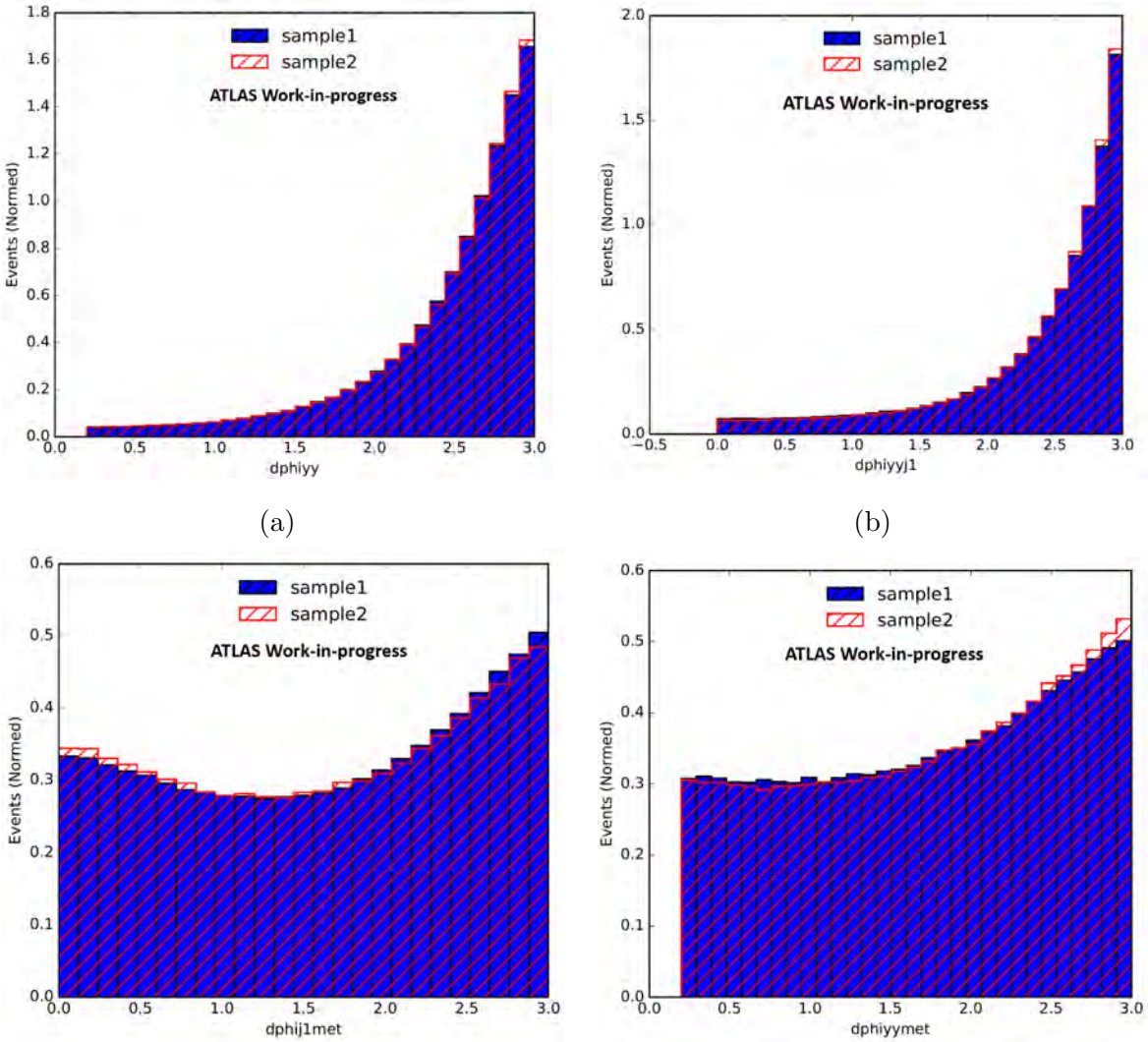


Figure A.6 Weakly supervised 1D Distributions

A.3 Correlation Matrices

	Vertex_sumpt2_php	npvx	metsig	N_j_central	dphiyy	weight	m_yy	dphij1met	dphiyymet	dphisjmet	dphifjmet	dphiyyj1	jets_rpt1	jets_jvf1	jets_rpt2	jets_jvf2	jets_rpt3	jets_jvf3
Vertex_sumpt2_php	1	0.008	-0.0091	0.21	-0.28	-0.023	-0.00037	0.15	-0.33	-0.022	-0.012	0.17	0.19	0.11	0.12	0.14	0.13	0.11
npvx	0.008	1	0.012	0.042	0.0068	-0.2	0.071	0.012	0.01	0.063	0.14	-0.026	-0.071	-0.066	0.087	0.041	0.12	0.038
metsig	-0.0091	0.012	1	0.002	0.03	-0.014	-0.041	0.19	-0.032	-0.0067	0.049	-0.17	-0.0072	-0.014	-0.019	-0.005	0.0062	0.0083
N_j_central	0.21	0.042	0.002	1	-0.19	-0.04	-0.007	0.17	-0.34	0.19	0.0046	0.12	0.13	0.15	0.51	0.56	0.57	0.58
dphiyy	-0.28	0.0068	0.03	-0.19	1	8.3e-05	0.036	0.12	0.25	0.032	-0.015	-0.4	-0.087	-0.1	-0.11	-0.11	-0.1	-0.1
weight	-0.023	-0.2	-0.014	-0.048	3e-05	1	-0.049	0.0025	0.0043	-0.054	-0.052	0.0023	0.0042	0.00041	-0.047	-0.029	-0.072	-0.055
m_yy	-0.00037	0.071	-0.041	-0.007	0.036	-0.049	1	-0.02	0.02	0.011	0.013	-0.0035	0.021	-0.0053	0.033	0.0047	0.018	0.0093
dphij1met	0.15	0.012	0.19	0.17	0.12	0.0025	-0.02	1	-0.58	0.025	0.056	-0.48	-0.037	-0.052	0.15	0.15	0.15	0.13
dphiyymet	-0.33	0.01	-0.032	-0.34	0.25	0.0043	0.02	-0.58	1	0.0007	-0.096	-0.39	-0.069	-0.082	-0.22	-0.23	-0.25	-0.21
dphisjmet	-0.022	0.063	-0.0067	0.19	0.032	-0.054	0.011	0.025	0.0007	1	0.017	-0.043	-0.042	-0.067	0.24	0.23	0.31	0.31
dphifjmet	-0.012	0.14	0.049	0.0046	-0.015	-0.052	0.013	0.056	-0.096	0.017	1	0.0066	-0.24	-0.33	0.36	0.066	0.4	0.15
dphiyyj1	0.17	-0.026	-0.17	0.12	-0.4	0.0023	-0.0035	-0.48	-0.39	-0.043	0.0066	1	0.14	0.17	0.034	0.046	0.036	0.036
jets_rpt1	0.19	-0.071	-0.0072	0.13	-0.087	0.0042	0.021	-0.037	-0.069	-0.042	-0.24	0.14	1	0.65	-0.07	-0.055	-0.057	-0.035
jets_jvf1	0.11	-0.066	-0.014	0.15	-0.1	0.00041	-0.0053	-0.052	-0.082	-0.067	-0.33	0.17	0.65	1	-0.17	-0.16	-0.099	-0.056
jets_rpt2	0.12	0.087	-0.019	0.51	-0.11	-0.047	0.033	0.15	-0.22	0.24	0.36	0.034	-0.07	-0.17	1	0.79	0.48	0.35
jets_jvf2	0.14	0.041	-0.005	0.56	-0.11	-0.029	0.0047	0.15	-0.23	0.23	0.066	0.046	-0.055	-0.16	0.79	1	0.36	0.27
jets_rpt3	0.13	0.12	0.0062	0.57	-0.1	-0.072	0.018	0.15	-0.25	0.31	0.4	0.036	-0.057	-0.099	0.48	0.36	1	0.8
jets_jvf3	0.11	0.038	0.0083	0.58	-0.1	-0.055	0.0093	0.13	-0.21	0.31	0.15	0.036	-0.035	-0.056	0.35	0.27	0.8	1

(a) Pearson Correlation Matrix for signal data

	Vertex_sumpt2_php	npvx	metsig	N_j_central	dphiyy	weight	m_yy	dphij1met	dphiyymet	dphisjmet	dphifjmet	dphiyyj1	jets_rpt1	jets_jvf1	jets_rpt2	jets_jvf2	jets_rpt3	jets_jvf3
Vertex_sumpt2_php	1	-0.044	0.039	0.21	-0.12	-0.037	0.069	-0.061	-0.014	-0.024	-0.058	0.066	0.2	0.13	0.11	0.11	0.1	0.1
npvx	-0.044	1	0.026	0.051	0.076	-0.064	0.091	-0.0084	-0.1	0.1	0.32	0.046	-0.079	-0.09	0.11	0.016	0.23	0.084
metsig	0.039	0.026	1	0.0023	-0.027	-0.003	-0.03	0.067	0.038	-0.033	-0.012	-0.09	-0.065	-0.065	-0.013	-0.0097	-0.0051	-0.014
N_j_central	0.21	0.051	0.0023	1	-0.15	-0.04	0.068	-0.071	-0.084	0.15	-0.019	0.083	0.13	0.17	0.42	0.48	0.48	0.5
dphiyy	-0.12	0.076	-0.027	-0.15	1	-0.011	0.022	0.36	-0.11	0.087	0.091	-0.3	-0.17	-0.14	-0.053	-0.067	-0.04	-0.056
weight	-0.037	-0.064	-0.003	-0.04	-0.011	1	-0.22	0.046	-0.011	-0.0024	-0.023	-0.037	-0.014	-0.0088	-0.037	-0.027	-0.042	-0.025
m_yy	0.069	0.091	-0.03	0.068	0.022	-0.22	1	-0.087	0.03	-0.0045	0.039	0.06	-0.00071	-0.0051	0.038	0.025	0.061	0.04
dphij1met	-0.061	-0.0084	0.067	-0.071	0.36	0.046	-0.087	1	-0.46	0.037	-0.011	-0.49	-0.34	-0.24	-0.0078	0.025	-0.052	-0.017
dphiyymet	-0.014	-0.1	0.038	-0.084	-0.11	-0.011	0.03	-0.46	1	-0.025	-0.19	-0.41	0.12	0.074	-0.09	-0.04	-0.14	-0.079
dphisjmet	-0.024	0.1	-0.033	0.15	0.087	-0.0024	-0.0045	0.037	-0.025	1	0.083	-0.061	-0.02	-0.067	0.23	0.23	0.28	0.29
dphifjmet	-0.058	0.32	-0.012	-0.019	0.091	-0.023	0.039	-0.011	-0.19	0.083	1	0.12	-0.081	-0.21	0.37	0.024	0.44	0.14
dphiyyj1	0.066	0.046	-0.09	0.083	-0.3	-0.037	0.06	-0.49	-0.41	-0.061	0.12	1	0.24	0.21	0.031	-0.032	0.097	0.036
jets_rpt1	0.2	-0.079	-0.065	0.13	-0.17	-0.014	-0.00071	-0.34	0.12	-0.02	-0.081	0.24	1	0.63	-0.018	-0.052	0.0056	0.0055
jets_jvf1	0.13	-0.09	-0.065	0.17	-0.14	-0.0088	-0.0051	-0.24	0.074	-0.067	-0.21	0.21	0.63	1	-0.12	-0.16	-0.075	-0.055
jets_rpt2	0.11	0.11	-0.013	0.42	-0.053	-0.037	0.038	-0.0078	-0.09	0.23	0.37	0.031	-0.018	-0.12	1	0.73	0.44	0.29
jets_jvf2	0.11	0.016	-0.0097	0.48	-0.067	-0.027	0.025	0.025	-0.04	0.23	0.024	-0.032	-0.052	-0.16	0.73	1	0.28	0.21
jets_rpt3	0.1	0.23	-0.0051	0.48	-0.04	-0.042	0.061	-0.052	-0.14	0.28	0.44	0.097	0.0056	-0.075	0.44	0.28	1	0.73
jets_jvf3	0.1	0.084	-0.014	0.5	-0.056	-0.025	0.04	-0.017	-0.079	0.29	0.14	0.036	0.0055	-0.055	0.29	0.21	0.73	1

(b) Pearson Correlation Matrix for background data

Appendix B

Confusion Matrices

B.1 Confusion Matrices

As mentioned in the results section, we take the confusion matrix of each model into consideration when analysing its ability to make predictions. Essentially, we want to know what the recall is for the background and signal to see how many times the model correctly classifies each of them. Below are confusion matrices for each one of these.

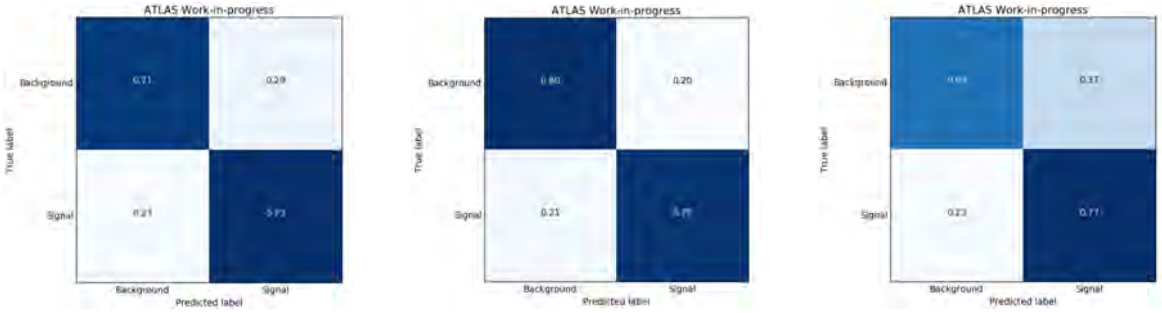


Figure B.1 Low $S_{E_T^{miss}}$: Confusion Matrices for A400Z, ggZH, and 275mx60

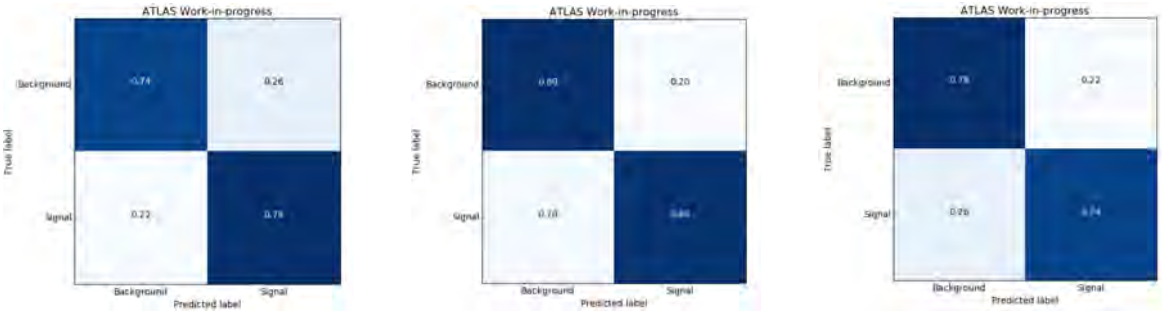


Figure B.2 Intermediate $S_{E_T^{miss}}$: Confusion Matrices for A400Z, ggZH, and 275mx60

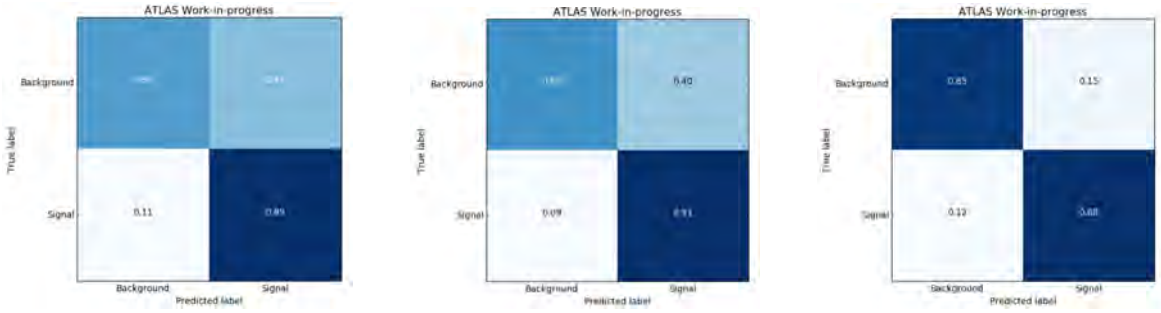


Figure B.3 High $S_{E_T^{miss}}$: Confusion Matrices for A400Z, ggZH, and 275mx60

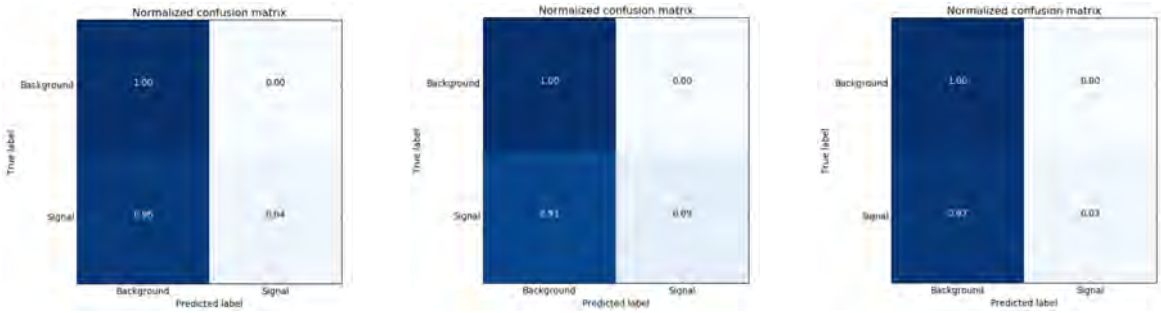


Figure B.4 Low $S_{E_T^{miss}}$: Confusion Matrices for A400Z, ggZH, and 275mx60

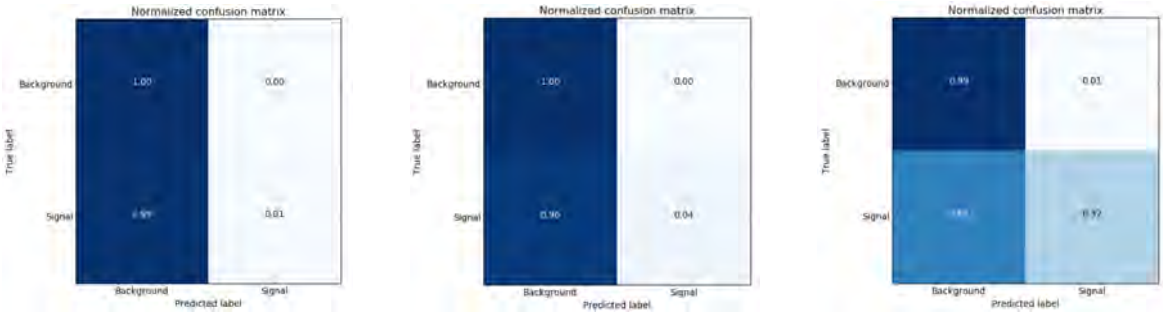


Figure B.5 Intermediate $S_{E_T^{miss}}$: Confusion Matrices for A400Z, ggZH, and 275mx60

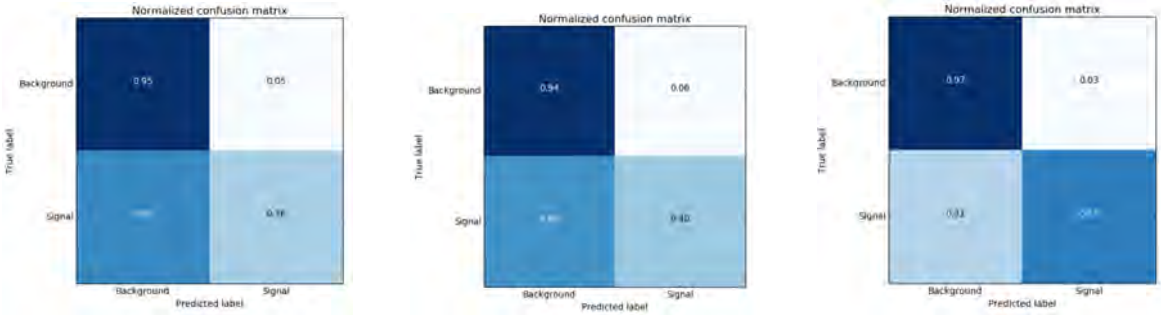


Figure B.6 High $S_{E_T^{miss}}$: Confusion Matrices for A400Z, ggZH, and 275mx60

Appendix C

Overfitting Check

C.1 Loss Plots

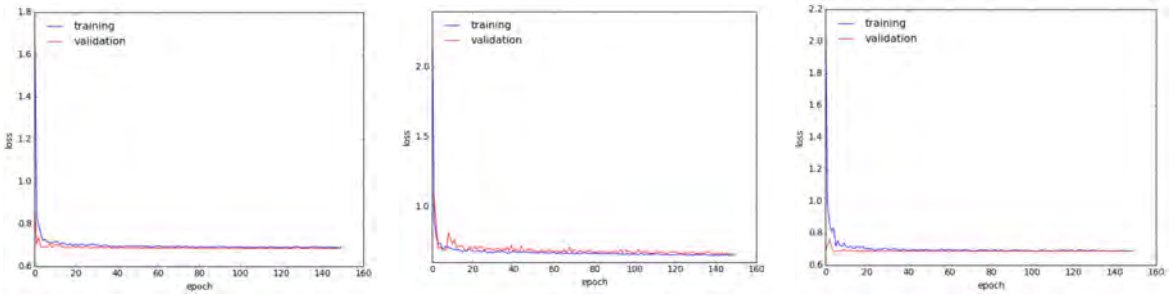


Figure C.1 Low $S_{E_T^{miss}}$: Train_Val Loss Plots

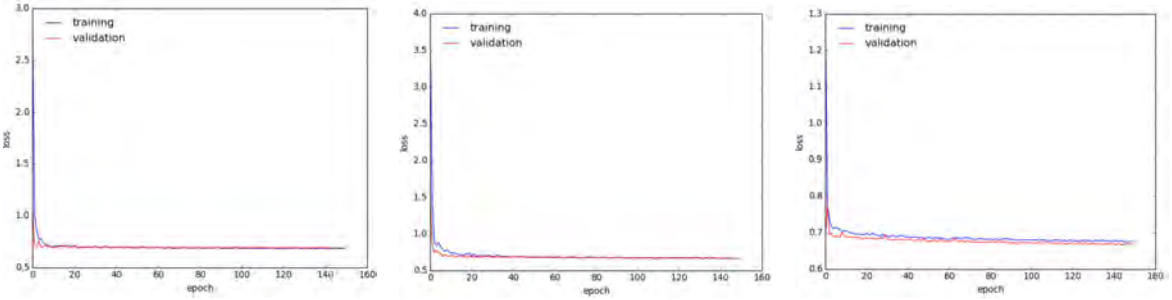


Figure C.2 Intermediate $S_{E_T^{miss}}$: Train_Val Loss Plots

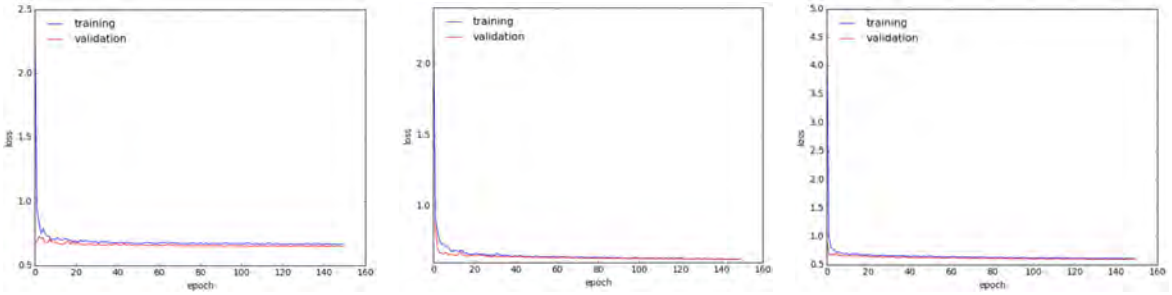


Figure C.3 High $S_{E_T^{miss}}$: Train_Val Loss Plots

Bibliography

[1] K.G. Tomiwa. Suppressing fake missing transverse energy using multivariate analysis with the atlas detector. *HEPP 2018 Proceedings,Cape Town.*

[2] ATLAS collaboration et al. Expected performance of missing transverse momentum reconstruction for the atlas detector at s= 13 tev. Technical report, ATL-PHYS-PUB-2015-023, 2015.

[3] Michael A Nielsen. *Neural networks and deep learning*, volume 25. Determination press San Francisco, CA, USA:, 2015.

[4] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

[5] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In *Advances in neural information processing systems*, pages 971–980, 2017.

[6] Tom M Mitchell et al. Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, 45(37):870–877, 1997.

[7] John Ellis and Dae Sung Hwang. Does the 'higgs' have spin zero? *Journal of High Energy Physics*, 2012(9):71, 2012.

[8] Bruce Mellado. Prospects of higgs physics at the lhc. In *Hadron Collider Physics 2002*, pages 348–355. Springer, 2003.

[9] Stefan von Buddenbrock, Alan S Cornell, Mukesh Kumar, and Bruce Mellado. The madala hypothesis with run 1 and 2 data at the lhc. In *Journal of Physics: Conference Series*, volume 889, page 012020. IOP Publishing, 2017.

[10] Stefan von Buddenbrock. Production of the madala boson in association with top quarks. *arXiv preprint arXiv:1901.08327*, 2019.

[11] Stefan Von Buddenbrock, Nabarun Chakrabarty, Alan S Cornell, Deepak Kar, Mukesh Kumar, Tanumoy Mandal, Bruce Mellado, Biswarup Mukhopadhyaya, and Robert G Reed. The compatibility of lhc run 1 data with a heavy scalar of mass around 270\, gev. *arXiv preprint arXiv:1506.00612*, 2015.

[12] Stefan von Buddenbrock. Exploring lhc run 1 and 2 data using the madala hypothesis. In *Journal of Physics: Conference Series*, volume 878, page 012030. IOP Publishing, 2017.

[13] Stefan von Buddenbrock, Alan S. Cornell, Yaquan Fang, Abdualazem Fadol Mohammed, Mukesh Kumar, Bruce Mellado, and Kehinde G. Tomiwa. The emergence of multi-lepton anomalies at the lhc and their compatibility with new physics at the ew scale. *Journal of High Energy Physics*, 2019(10):157, Oct 2019.

[14] Kostas Kordas, E Pasqualucci, G Gaudio, A Lowe, G Comune, H Garitaonandia, NG Unel, G Mornacchi, E Stefanidis, T Pauly, et al. The atlas data acquisition and trigger: concept, design and status. *Nucl. Phys. B, Proc. Suppl.*, 172(ATL-DAQ-CONF-2007-022):178–182, 2006.

[15] Nick Ellis. Trigger and data acquisition. *arXiv preprint arXiv:1010.2942*, 2010.

[16] Dan Guest, Kyle Cranmer, and Daniel Whiteson. Deep learning and its application to lhc physics. *Annual Review of Nuclear and Particle Science*, 68:161–181, 2018.

[17] Michela Paganini. Machine learning solutions for high energy physics: Applications to electromagnetic shower generation, flavor tagging, and the search for di-higgs production. *arXiv preprint arXiv:1903.05082*, 2019.

[18] Kim Albertsson, Piero Altoe, Dustin Anderson, Michael Andrews, Juan Pedro Araque Espinosa, Adam Aurisano, Laurent Basara, Adrian Bevan, Wahid Bhimji, Daniele Bonacorsi, et al. Machine learning in high energy physics community white paper. In *Journal of Physics: Conference Series*, volume 1085, page 022008. IOP Publishing, 2018.

[19] G Amadio, Ph Canal, J de Fine Licht, F Carminati, R Seghal, A Gheata, RL Iope, O Shadura, D Elvira, R Brun, et al. The geantv project: preparing the future of simulation. In *J. Phys. Conf. Ser.*, volume 664, page 072006, 2015.

[20] Federico Carminati, Andrei Gheata, Gulrukh Khattak, P Mendez Lorenzo, S Sharan, and S Vallecorsa. Three dimensional generative adversarial networks for fast simulation. In *Journal of Physics: Conference Series*, volume 1085, page 032016. IOP Publishing, 2018.

[21] Michela Paganini, Luke de Oliveira, and Benjamin Nachman. Calogan: Simulating 3d high energy particle showers in multilayer electromagnetic calorimeters with generative adversarial networks. *Physical Review D*, 97(1):014021, 2018.

[22] Theodore C Gaelejwe, Bruce Mellado, and Kehinde Tomiwa. Suppressing fake missing transverse energy using multivariate analysis with the atlas detector. *University of the Witwatersrand*, 2018.

[23] Daniel Svozil, Vladimir Kvasnicka, and Jiri Pospichal. Introduction to multi-layer feed-forward neural networks. *Chemometrics and intelligent laboratory systems*, 39(1):43–62, 1997.

[24] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[25] Olivier Chapelle, Bernhard Schölkopf, and Alexander Zien. A discussion of semi-supervised learning and transduction. In *Semi-Supervised Learning*, pages 473–478. MIT Press, 2006.

[26] Zhi-Hua Zhou and Ming Li. Semi-supervised learning by disagreement. *Knowledge and Information Systems*, 24(3):415–439, 2010.

[27] Torbjörn Sjöstrand, Stephen Mrenna, and Peter Skands. Pythia 6.4 physics and manual. *Journal of High Energy Physics*, 2006(05):026–026, May 2006.

[28] Manuel Bähr, Stefan Gieseke, Martyn A. Gigg, David Grellscheid, Keith Hamilton, Oluseyi Latunde-Dada, Simon Plätzer, Peter Richardson, Michael H. Seymour, Alexander Sherstnev, and et al. Herwig++ physics and manual. *The European Physical Journal C*, 58(4):639–707, Nov 2008.

[29] Johan Alwall, R Frederix, S Frixione, V Hirschi, Fabio Maltoni, Olivier Mattelaer, H-S Shao, T Stelzer, P Torrielli, and M Zaro. The automated computation of tree-level and next-to-leading order differential cross sections, and their matching to parton shower simulations. *Journal of High Energy Physics*, 2014(7):79, 2014.

[30] Tanju Gleisberg, Stefan Höche, F Krauss, M Schönherr, S Schumann, F Siegert, and J Winter. Event generation with sherpa 1.1. *Journal of High Energy Physics*, 2009(02):007, 2009.

[31] Zhi-Hua Zhou. A brief introduction to weakly supervised learning. *National Science Review*, 5(1):44–53, 2017.

[32] Lucio Mwinmaarong Dery, Benjamin Nachman, Francesco Rubbo, and Ariel Schwartzman. Weakly supervised classification in high energy physics. *Journal of High Energy Physics*, 2017(5):145, 2017.

[33] Eric M Metodiev, Benjamin Nachman, and Jesse Thaler. Classification without labels: Learning from mixed samples in high energy physics. *Journal of High Energy Physics*, 2017(10):174, 2017.

[34] Zhi-Hua Zhou. When semi-supervised learning meets ensemble learning. *Frontiers of Electrical and Electronic Engineering in China*, 6(1):6–16, 2011.

[35] Xiaojin Zhu. Semi-supervised learning literature survey contents. *SciencesNew York*, 10(1530):10, 2008.

[36] Vikas C Raykar and Shipeng Yu. Eliminating spammers and ranking annotators for crowdsourced labeling tasks. *Journal of Machine Learning Research*, 13(Feb):491–518, 2012.

[37] Felix X Yu, Krzysztof Choromanski, Sanjiv Kumar, Tony Jebara, and Shih-Fu Chang. On learning from label proportions. *arXiv preprint arXiv:1402.5902*, 2014.

[38] Thomas G Dietterich, Richard H Lathrop, and Tomás Lozano-Pérez. Solving the multiple instance problem with axis-parallel rectangles. *Artificial intelligence*, 89(1-2):31–71, 1997.

[39] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[40] RP Lippmann. An introduction to computing with neural networks. *IEEE ASSP Magazine*, 3(4):987, 1991.

[41] Yoshua Bengio et al. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

[42] Thomas Charman. Atlas detector event classification with tensorflow.

[43] Raul Rojas. The backpropagation algorithm. In *Neural networks*, pages 149–182. Springer, 1996.

[44] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[45] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[46] Mina Basirat and Peter M. Roth. The quest for the golden activation function. *CoRR*, abs/1808.00783, 2018.

[47] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3, 2013.

[48] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[49] Yi Zhou, Junjie Yang, Huishuai Zhang, Yingbin Liang, and Vahid Tarokh. Sgd converges to global minimum in deep learning via star-convex path. *arXiv preprint arXiv:1901.00451*, 2019.

[50] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[51] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 6(2):107–116, April 1998.

[52] Y Bengio, P Simard, and P Frasconi. Long short-term memory. *IEEE Trans. Neural Netw*, 5:157–166, 1994.

[53] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[54] Yoshua Bengio, Patrice Simard, Paolo Frasconi, et al. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[55] Ethem Alpaydin. Introduction to machine learning (adaptive computation and machine learning series). *The MIT Press Cambridge*, 2004.

[56] Thomas G Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine learning*, 40(2):139–157, 2000.

[57] Thomas G Dietterich. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15. Springer, 2000.

[58] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of online learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[59] Yu Shi, Jian Li, and Zhize Li. Gradient boosting with piece-wise linear regression trees. *arXiv preprint arXiv:1802.05640*, 2018.

[60] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[61] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016.

[62] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[63] Vicente García, Ramón A Mollineda, and J Salvador Sánchez. A bias correction function for classification performance assessment in two-class imbalanced problems. *Knowledge-Based Systems*, 59:66–74, 2014.

[64] Jeff Berry, Ian Fasel, Luciano Fadiga, and Diana Archangeli. Training deep nets with imbalanced and unlabeled data. In *Thirteenth Annual Conference of the International Speech Communication Association*, 2012.

[65] Alberto Munoz and Javier M Moguerza. One-class support vector machines and density estimation: The precise relation. In *Iberoamerican Congress on Pattern Recognition*, pages 216–223. Springer, 2004.

[66] Jinglin Xu, Junwei Han, Kai Xiong, and Feiping Nie. Robust and sparse fuzzy k-means clustering. In *IJCAI*, pages 2224–2230, 2016.