UNIVERSITY
OF OSLO

**Master's thesis**

# Search for heavy neutrinos in a 3-lepton final-state

A study of supervised machine learning models

**William Hirst**

60 ECTS study points

Department of Physics
Faculty of Mathematics and Natural Sciences


30th November 2022

# Abstract

This will be the abstract.

# Acknowledgments

Thank you, to nobody.

# Contents

# Introduction

The Standard Model (SM) is perhaps one of the most successful scientific theories ever created. It accurately explains the interactions of leptons and quarks as well as the force carrying particles which mediate said interactions. In 2012 the SM achieved one of its crowning achievements when we discovered the Higgs boson. Much of the accolade was rightfully given to the theoretical work on the SM, but another aspect of the discovery was equally important. Data analysis was and is a crucial part of any new discovery in physics. One of the most important and exiting tools is Machine Learning (ML).

**Outline of the Thesis**

# Chapter 1

# The Standard model of elementary particles and beyond.

The SM is the most successful scientific theory ever created. It accurately explains the interactions of leptons and quarks as well as the force carrying particles which mediate said interactions. The model is a result of over a century of work demanding the contributions of grat minds like Paul Dirac, Erwin Schrodinger and Richard Feynman. In 2012 the SM achieved one of its crowning achievements when we discovered the Higgs boson.

## 1.1 The bulding blocks

As early as ancienct greece, humans pondered the nature of the most elementary building blocks of the universe. They imagined a rope of a given length, with a pair of scissors of adjustable size. Then one could ask, how many times can you cut the rope in half? If the answer is less than infinite, what are you left with? In 1897, Joseph John Thomson (*1856-1940*) discovered the first elementary particle using the Cathode Ray Tube. This particle we later named the electron. Prior to the time of discovery, we believed atoms to be the smallest building blocks. After the discovery of the electron, the discovery of the proton and neutron quickly followed. It was not until more than 50 years after the discovery of the proton (by Ernest Rutherford) that we discovered that also protons and neutrons could be further dissected to smaller particles. We call these particles quarks. The "final-piece"[1] of the puzzle came in 1956 when we discovered the, at that time thought of as massless neutrino. Together with the electron, the neutrino is defined as a lepton. Together with the quarks and leptons are called fermions.
Upon the evolution of the quantum mechanics and a physics as a whole, we started to divert our focus from the what and over to the how. How can we explain all the complex interactions that emerge between these relatively simple particles? Through the creation of SM and countless experiment, we discovered that forces are nothing but interactions between particles and fields. The SM describes all forces as a field which are mediated through a particle, we call bosons.
The four forces responsible for all the forces in the universe are electromagnetism (Quantum Electro Dynamics (QED)), the weak-force, the strong-force(Quantum Chromo Dynamics (QCD)) and gravity. The boson most familiar to most is the photon. The photon is responsible for the mediation of QED and is responsible for all electromagnetic effect, such as the ones allowing us to sea objects using our eyes. Similarly, the W and Z bosons are responsible for the weak-force which allows for radioactive decay. And the gluon is responsible for QCD which holds protons and neutrinos together. Gravity is the only force not described in the SM, but would (if one day included) have its own force carrying particle, graviton.
The final building block in the universe introduced and described by SM is the Higgs boson. The Higgs boson was proposed by Peter Higgs in 1964 and discovered at CERN in 2012. The Higgs boson, also called the God particle is responsible for giving particles mass in a process called spontaneous symmetry breaking (more on this in later sections). Together the fermions and the bosons make up all the particles in the SM as it now stands.

[1] Given the nature of this thesis, the existence of further pieces is implied.

| Generation | *Flavour* | Mass [MeV] | Charge [Elementary charge] |
|---|---|---|---|
| $1st$ | e | 0.511 | $-1$ |
| $1st$ | $\nu_e$ | $< 0.001$ | 0 |
| $2nd$ | $\mu$ | 105.66 | $-1$ |
| $2nd$ | $\nu_\mu$ | $< 0.17$ | 0 |
| $3rd$ | $\tau$ | 1776.8 | $-1$ |
| $3rd$ | $\nu_\tau$ | $< 18.2$ | 0 |

**Table 1.1:** A list of all leptons along with their generation, flavor, mass and charge.

| Generation | *Flavour* | Mass [MeV] | Charge [Elementary charge] |
|---|---|---|---|
| $1st$ | u | 2.2 | $-2/3$ |
| $1st$ | $d$ | 4.7 | $+1/3$ |
| $2nd$ | $c$ | 1280 | $-2/3$ |
| $2nd$ | $s$ | 96 | $+1/3$ |
| $3rd$ | $t$ | 173100 | $-2/3$ |
| $3rd$ | $d$ | 4180 | $+1/3$ |

**Table 1.2:** A list of all quarks along with their generation, flavor, mass and charge.

### 1.1.1 The leptons

The leptons are all elementary particle with half-integer spin, $\pm 1/2$. A lepton can either be charged or neutral. For reasons that are yet to be known, the leptons come in 3 generations. Each generation containing a pair of charged and neutral lepton. The first generation contains the electron, $e^-$ and the electron-neutrino, $\nu_e$. The second contains the muon, $\mu$ and the muon-neutrino, $\nu_\mu$. And the third generation contain the tau, $\tau^-$ and $\nu_\tau$. The generations are numbered by the mass of the charged lepton, where the first generation is the lightest. As is often the case in particle physics, the heavier a particle, the rarer. This is due to the heavier particles (higher generations) quickly decaying into lighter particles (lower generation), in a process we call particle decay. This explains why particle physicists neglect the $\tau$ when speaking about leptons, given that this is by far the heaviest and also the rarest.
The charged leptons are all massive particles ranging from a fraction of 1eV to more than a thousand times that. The neutrinos were up until the turn of the millennia presumed to be massless. This was not only backed by experiments but also by the SM which seldom seemed to be wrong. In 1998 it was discovered that neutrinos in fact do have mass although being extremely light. Given the size of the masses we are yet to accurately measure the mass of the neutrinos, but we have found them all to be less than 20 MeV. The fact that the neutrinos in fact do have mass is a problem which will be discussed further in later section. In table 1.1, a summary of all leptons are found, along with the respective mass and charge.

### 1.1.2 The quarks

> 'Three quarks for Muster Mark!
> Sure he hasn't got much of a bark.
> And sure any he has it's all beside the mark.' [1]

The poem above was written by James Joyce in 1939, and was the motivation for Gell-Mann when naming the inner particles of hadrons, quarks. Quarks were introduced to explain some strong-force properties of hadrons. We categorize quarks as either up- or down quarks. All up-quarks have a negative charge equal to 2/3 that of the electron and all down quarks have a positive charge equal to 1/3 that of the electron. Similarly to leptons, all quarks have a spin equal to 1/2 and are divided in 3 generations. Each generation of quarks are made of a pair of one up- and one down-quark. The first generation contains the up, $u$ and the down, $d$ quark, the second the charm, $c$ and the strange, $s$ quark and third the top, $t$ and the bottom, $b$ quark. Also similarly to leptons, the higher the generation and mass the rarer the quarks. Similarly to how difference in spin allows leptons to stay in an otherwise similar quantum state, the quarks have color. The colors of quarks are what connects them to the strong-force. The strong force is what allows quarks to change color and also explains a phenomenon known as color confinement. Briefly explained, color confinement results in quarks never existing in isolation but always in pairs, mesons or in threes, baryons (like protons and neutrons). Given color confinement, quarks are never directly observed in experiments, instead we detect the signature of quarks forming hadrons in a process called hadronisation. We call these signature jets.

## 1.2 The Forces

### 1.2.1 Quantum electro dynamics

### 1.2.2 The Weak Force

### 1.2.3 The Strong Force

## 1.3 Beyond the Standard Model

### 1.3.1 Why look beyond?

> 'There is nothing new to be discovered in physics now.
> All that remains is more and more precise measurement.' [2]

The quote above is rumored to have been spoken by William Thompson(*1824–1907*), better known as Lord Kelvin when addressing the British Association for the Advancement of Science in 1900. The statement followed a long period of advancements in the field of physics by the likes of James Clerk Maxwell (*1831–1879*) and Michael Faraday (*1791–1867*). It would take less than half a decade before he would understand the magnitude of his miscalculation, when Einstein and Planck began the development of Quantum Mechanics. Just as Kelvin was wrong back then, would he be wrong today. For although SM explains a large range of phenomena, there are yet many mysteries to explain in the universe and even problems rooted in SM. In this section I will explain some of the problems we hope to tackle in the future.

As mentioned in previous section, SM is yet to explain *gravity*. The hope has been to integrate gravity into SM through the discovery of a gravity-carrying particle, the gravtion. So far, no-such particle is found. *Dark matter* and *dark energy* are also not described by SM, even though the two makeup more than 90% of the mass in the observable universe. Inflation is today the leading explanation to what happened in the early-stages (the first fraction of a second) of the universe. It explains a universe in which all space undergoes a rapid increase in rate of expansion. None of the fields explained by SM are capable of causing any such expansion. Finally, and the one most relevant for this analysis is the neutrino-mass and Charge-Parity (CP)-violation problems, but this will be discussed in the next section.

### 1.3.2 Neutrino-Mass problem

Neutrinos have a special place in physics. For one, they are the only particles that only interact by the weak force. This means that neutrinos rarely interact at all. It is often used as a (granted for many not incredibly exiting) conversation piece that a colossal amount ($ca.10^{14}$) of neutrinos pass through your body every second. This is harmless to us exactly because of the rarity of neutrinos interaction with anything. For this reason we call neutrinos ghostly.
Another reason neutrinos are special is that they exhibit flavor mixing. In 2015 Arthur McDonald (*1943-*) and Takaaki Kajita (*1959-*) were awarded the Nobel Prize for their contributions in the discovery. In simple terms, flavor mixing is a process where a particle oscillates between different flavors. For neutrinos this means oscillating between $\nu_e$, $\nu_\mu$ and $\nu_\tau$. Flavor mixing is in it of itself not special. Quarks have been observed to exhibit the same behavior. The reason this is interesting in the case of neutrinos, is that it implies that neutrinos are massive. Before this discovery, we believed neutrinos to be massless. But why are massive neutrinos a problem?
All (previously) known massive particles gain mass through interactions with the Higgs boson. For particles to gain mass they need to have a right- and left-handed[2] particle. So far, no right-handed neutrinos have been observed. Generally there are two schools of thought for why a right-handed neutrino has not been observed. Either, it is very heavy, and we are yet to generate energies large enough to recreate it, or it is indistinguishable to the left-handed neutrino. In the first scenario we would call the right-handed neutrino a *Dirac* fermion. This means that it is no different from any other right-handed lepton. In the second the right-handed neutrino is a *Majorana* fermion. A Majorana fermion is simply a lepton where the particle and the antiparticle are the same. In this thesis I have searched for a right-handed neutrino and considered both Majorana and Dirac neutrinos.

[2]The handedness of a particle is defined as a relation between the direction of spin and momentum for a particle. Right means the two are directed in the same direction and left corresponds to opposite direction.

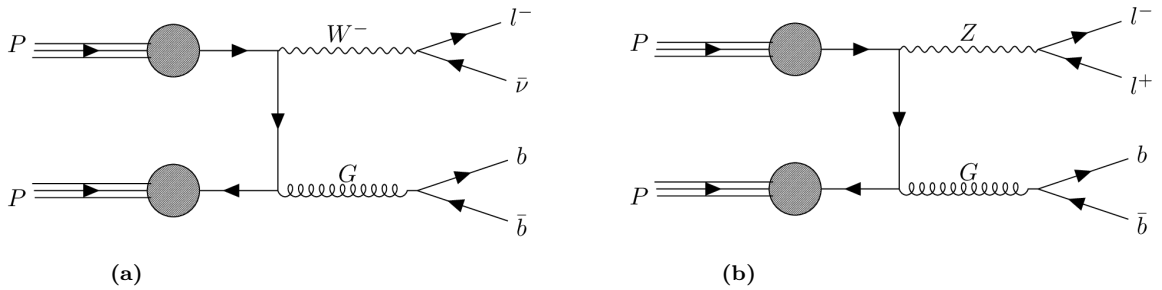



**Figure 1.1:** The Feynman diagram of both the W+jets 1.1a and the Z+jets 1.1b.

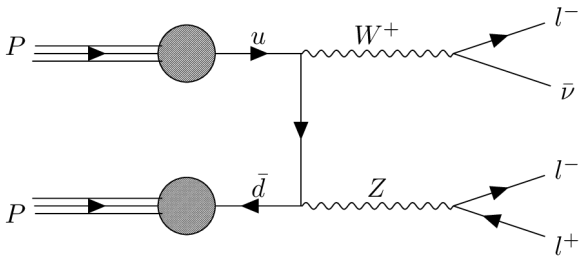



**Figure 1.2:** The Feynman diagram of the diboson WZ-channel.

## 1.4 The Background Channels

The dominant SM backgrounds can be divided into two categories: (i) from leptonic $\tau$ decays and (ii) from fake leptons. In the first category, the dominant process is the pair production of $WZ$ with $W$ decaying leptonically and $Z \to \tau\tau$. The trilepton final states with no-OSSF pairs can arise from the subsequent leptonic decay of $\tau$ 's. We estimate this background process via Monte Carlo simulations.
The dominant processes of the second category are $\gamma^*/Z+$ jets and $t\bar{t}$, where two leptons come from $\gamma^*/Z \to \tau\tau$ or the prompt decay of $t$ and $\bar{t}$, and a third lepton is faked from jets containing heavy-flavor mesons.

## 1.5 The Signal

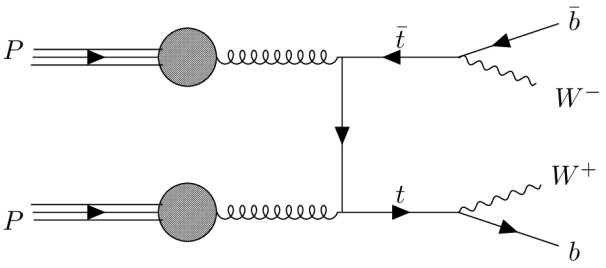



**Figure 1.3:** The Feynman diagram of the $t\bar{t}$-channel.

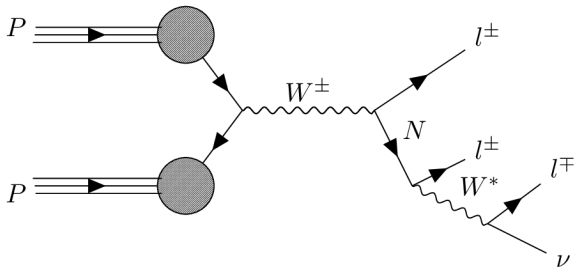



**Figure 1.4:** The Feynman diagram of the signal-channel.

# Chapter 2

# Introduction to Machine Learning and Data Analysis

Machine learning is rapidly becoming an overwhelming presens in many scientific fields. In areas ranging from cancer research to stock-trading, machine learning is being applied to problems once thought as impossible to solve. Particle physics, like many other fields is no exception. Jet flavor classification [3], separating jets from gluons [4] or using ML to create efficient Signal region (SR) are just some examples where ML is a vital tool. The traditional approach for ML in high-energy physics is through the use of supervised learning. Deep Neural Networks (DNN)

## 2.1 Phenomenology

ML differs from other analysis tools in its ability to learn. Where a purely analytical model is static in both method and performance a ML model, $\mathcal{F}$ aims to be dynamic and self improving. ML utilizes data, $x_i$ to leverage towards an optimal model. The extent of utilization of data defines a ML model as being either *supervised*, *semi-supervised* or *unsupervised*. In the case of supervised ML, a set of *targets*, $t_i$ are provided along with the data which allows a ML model to learn to map directly from data to target. Targets are simply the measured values corresponding to the input of the ML-model. The target values can be both continues values, in which case the ML method aims to perform a *regression*, or discrete values, in which case the ML method aims to perform a *classification*.
The goal of supervised learning is that $\mathcal{F}$ can apply any attained knowledge from the study of ($x_i$, $t_i$) to predict the target, $\tilde{t}_i$ of a new dataset $\tilde{x}_i$. The success of any prediction is dependent on the quality of the data used during training, or *training-data*. The training-data is required to be both representative of any trends you hope to detect in the new data set, or test-data and be of sufficient amount. The latter point stems from a phenomenon known as *over-fitting*, which is a problem where the training data becomes overly specialized to only predict the target of the training-data, and nothing else[3]. In this thesis the main focus will be on the application of supervised learning.
In the case of unsupervised ML, no target is provided. The motivation for unsupervised learning is to create a model which is independent of any target. Such models are often useful in the case where one is not certain what one is looking for. An independence of target means that the model is not overly sensitive to any specific trends or patterns, we call this being *unbiased*. Instead of learning to detect or predict specific phenomenon, unsupervised learning is often used to detect anomalies in the data, which means it relies heavily on statistics. Because of this, some use the term *anomaly detection* and unsupervised learning interchangeably.
Semi-unsupervised learning is a loose term which finds itself in the middle of the previous two. It often refers to methods where no concrete target is provided, but instead uses the data provided to create a target. The goal is to alleviate as much bias as possible, but at the same time converge towards the usually superior performance of supervised learning.

## 2.2 Optimization

For a general function $g$ dependent one a set of parameters $\boldsymbol{\theta} = \{\theta_0, \theta_1, ..., \theta_{N_\theta}\}$, the goal of optimization is to find optimal parameters as defined by a predicated goal. In our case we are interested in finding the set

[3] More on this in later sections.

of parameters corresponding to the minimum value of $g$. Several methods can be applied to optimization problems, all with their own advantages and disadvantages. In most methods the use of the gradient of the function, $\boldsymbol{\nabla}_\theta(g)$ is involved in one way or another. Many of the methods used in this analysis are based on one of the simplest optimization methods, the *gradient descent*-method.

### 2.2.1 Gradient Descent

The gradient descent method aims to obtain the optimal parameters $\tilde{\boldsymbol{\theta}}$ through the application of the derivative of $g$ with respect to $\boldsymbol{\theta}$. When evaluated in a given point in the parameter space $g(\boldsymbol{\theta}_i)$ the negative of the gradient $\boldsymbol{\nabla}_\theta(g)$, is used to move $\boldsymbol{\theta}_i$ closer to $\tilde{\boldsymbol{\theta}}$. The negative is used due that $-\boldsymbol{\nabla}_\theta(g)$ corresponds to the direction for which a small change $d\boldsymbol{\theta}$ in the parameter space will result in the biggest decrease in the cost function. Finding the minimum value is an iterative process, meaning the steps in the direction of $-\boldsymbol{\nabla}_\theta(g)$ is finite. The size of the step is a hyperparameters decided by the user and is called the learning rate, $\eta$. The evolution from a step i to $i+1$ becomes

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \eta \cdot \nabla_\theta g\left(\boldsymbol{\theta}_i\right). \tag{2.1}$$

Choosing the learning rate can drastically effect the performance of the gradient descent method. Too large and one risks ”jumping” over the true minimum or simply never allowing for parameters to reach a high accuracy. Too small and one risks having to spend computation time beyond reason.

### 2.2.2 Adam

## 2.3 Model assessment

### 2.3.1 Cost functions

How we define the performance of a ML model is not only important when evaluating the model, but is crucial during training. In the case of classification, it is natural to assume an appropriate metric should involve a comparison between the predicted classification and the true classification. The variation of performance metrics stems from the diversity of how one quantifies the comparison between the two. During training, we define an objective function used to guide the model towards optimal tuning. We call this function the *cost function*.

#### Mean Squared Error

Mean Squared Error (MSE)

#### Binary Crossentropy

$$\mathcal{C}\left(\{\mathbf{y}\}_{i=0}^{N}, \{\mathbf{t}\}_{i=0}^{N}\right) = -\sum_{i=1}^{N}\left[\mathbf{y}_i \log\left(\mathbf{t}_i\right) + \left(1-\mathbf{y}_i\right)\log\left(1-\mathbf{t}_i\right)\right] \tag{2.2}$$

### 2.3.2 The Rate of True-Positve - ROC Curve

A Receiver Operating Characteristic (ROC) curve is a tool used to measure and visualize a binary classifiers' ability to predict trends. The curve is plotted on an x-, y-axis where the x-axis represents true positive rates and the y-axis represents false positive. The different values for the curve are the rate of true positives with different thresholds, i.e. the value deciding whether an event is 1 or 0, signal or background. If a classifier has learned nothing and is simply guessing, the ROC curve will be a linear curve going from 0 to 1. This line is often drawn in ROC curve. The better the classifier is, the higher the ROC curve will bend towards the upper-left corner of the graph. The worse it is, the more it will bend to the lower right corner.
A metric often used to measure a classifiers' ability create an output which effectively separates two categories, is the Area Under the Curve (AUC). The higher the area, the larger the separation. An ideal classifier which perfectly separates two categories will achieve a AUC of 1. A classifier which simply guesses, will achieve an AUC of 0.5. Both this cases assume an equal weighting of both signal and background.

### 2.3.3 Interpretability

### 2.3.4 Generalizability

## 2.4 Hyperparameters

The tuning of parameters is a vital part of building an optimal ML model, though not all parameters are set during training. Parameters that are chosen prior to training are called *hyperparameters*. We differentiate between two types of hyperparameters; model hyperparameters and algorithm hyperparameters. Model hyperparameters refer to parameters used to define the architecture of the model. Examples of this could be the size and deepness of a Neural Network (NN) or the maximum deepness of a Decision Trees (DT). These parameters are not tuned during training, but will nonetheless have a great impact on the performance of the model. Algorithm hyperparameters on the other hand, do not have an impact on the performance of the model. These are parameters that mainly effect the effectiveness and quality of the training process. Examples of this are the learning rate of a NN, or the batch-size used during training. Regardless of if we are discussing model- or algorithm hyperparameters, it is in our interest to find the optimal choice of parameters. The choice of parameters is made prior to the final training, and will be discussed in the following section.

### 2.4.1 Grid Search

## 2.5 Data Handling

### 2.5.1 Scaling

The distribution of values for different features can vary immensely, and for most cost functions, this is a problem. For some features, a deviation on the magnitude of $10^3$ can be a good approximation, whereas for others it can be a vast overestimation. When a model is to define which direction it wants to tune, it is crucial that the errors across all features are weighted equally. Scaling aims levitate this problem by transforming all features to have a relatively equal range of values while simultaneously preserving all information regarding each feature. The choice of how one chooses to scale the data will heavily affect the performance of the model and is therefore a part of the model.

#### Standard Scaler

The *Standrad Scaler* implemented in this rapport uses Scikit-learns's *StandardScaler* [5]. The standard scaler function scales each feature individually by subtracting the mean and dividing by the standard deviation. In doing so the resulting scaled data has a mean of 0 and a standard deviation of 1. Mathematically the standard scaler, $\mathcal{S}$ transforms a dataset $\{\mathbf{x}\}_{i=0}^{N}$ as

$$\mathcal{S}\left(\{\mathbf{x}\}_{i=0}^{N}\right) = \frac{\{\mathbf{x}\}_{i=0}^{N} - \boldsymbol{\mu}_x}{\boldsymbol{\sigma}_x}, \tag{2.3}$$

where $\boldsymbol{\mu}_x$ and $\boldsymbol{\sigma}_x$ are vectors with the elements being the mean and standard deviation respectively for each feature.

### 2.5.2 Principal Component Analysis

## 2.6 Regularization

In ML, overfitting occurs when a model becomes overly tuned to the training data and as a consequence fails to extract trends which would allow it to predict previously unseen data. The architecture of a NN, the maximum depth of a boosted-DT or even the size of the dataset can all contribute to overfitting. In the case of deep learning especially, overfitting can be a large problem and is therefore of focus in this thesis. Apart from predicting on a new data set, there are no rigid methods to detect overfitting. Instead, there exists many attempts to minimize the risk of it, we call these methods' *regularization*. In ML, regularization is known as any attempt to reduce the error in a prediction by reducing overfitting. Generally one can categorize regularization as being either implicit or explicit. Explicit regularization means adding terms to the optimization problem. This is a very direct way of insuring no part of the model becomes overly dominant. Examples of this could be adding a penalty in the cost function of a NN to ensure no weights

become to large. Implicit regularization is a less direct attempt of hindering overfitting. This could be changing the depth of the ML-model, varying the cost function or altering the data itself.

### 2.6.1 Early stopping

A simple implicit regularization method is to introduce *early stopping* in the training pipeline. Early stopping simply means to stop training before the parameters of the model are allowed to over tune. The usual approach is to introduce a goal for the training, which when reached, ends the training. Examples of these goals could be a predetermined loss value on the training set or training until lack of progress on a second unseen dataset. The latter approach is the one I will use in this thesis.

### 2.6.2 Ensembles

When comparing different ML methods by performance, more often then not the top performing model includes ensembling in one way or another. Like the word suggests, ensembling in ML means using a collection of ML models to create one complex model. There are many ways to create ensembles of models, most methods fall in to one of three categories; *bagging*, *boosting* and *stacking*. Creating an ensemble of models through bagging, means to use several models each trained on their own sample from the same dataset. The overarching new model is created by averaging the predictions from the ensemble of models. The method seeks to create a unique set of models through exposing each individual model to different training sets. Boosting is different to bagging in that it uses the same training data on all the models. The diversity in the models when boosting, stems from intentionally choosing the architecture of the models such that it reduces the error made by the previous ensemble of models. Finally, stacking uses a predetermined model to decide how to combine the predictions made by the ensemble. More on the specifics of bagging, boosting and stacking in later sections.

## 2.7 Neural Networks

The concept of a NN has been around for more than 80 years, and today they are one of the most popular and successful ML methods. The key to its popularity stems from its versatility, achieving high performance in a large range of both regression and classification problems. One of the defining qualities in a NN is the possibility of diverse architecture, meaning that a there are many categorize of NN, where each category has an even deeper selection of networks. Categories ranging from Convolutional Neural Network (CNN), Recursive Neural Network (RNN) to simple Feed-Forward Neural Network (FFNN), where each category is specified for their own set of problems. In this section I will introduce some fundamental definitions in regard to NN, as well go through the underlying algorithm of the back- and forward propagation.

### 2.7.1 General structure

There are often drawn comparisons between the structure of the neural network, and the way the human mind operate, hence neural. Similarly to the human mind, a NN is composed of different neurons communicating information backwards and forwards in different regions In the case of a neural network we call these regions layers. All layers are composed of a specified number of neurons. A NN has three types of layers; *input-layer*, *hidden-layer* and *output-layer*. There is only one input layer, and it has the same number of nodes equal to the number of features for each data point. There can be an arbitrary number of hidden layers, with each hidden-layer containing an arbitrary number of nodes. Finally, the NN has an output layer. The output-layer contains a number of nodes equal to the number of features for the target.
The neural network functions by passing information in between the different layers though nodes. The nodes are simply pockets of information, each containing a value. All the nodes in the input layer are (in most cases) connected to all nodes in the nearest hidden layer, and likewise said hidden layer is connected to the next hidden layer. This structure continues until we reach the final layer, the output layer. The structure is illustrated in figure 2.1. The figure shows a simple NN with a 2 dimensional data set (2 nodes in input-layer), 2 hidden layers with three nodes each and a 1 dimensional target value. It also illustrates how a NN aims to map from data, $\{\mathbf{x}\}_{i=0}^{N}$ to a prediction $\{\mathbf{y}\}_{i=0}^{N}$. In figure 2.1 we can see how all the different nodes are connected, illustrated by the arrows. The passing of values between different nodes are controlled by a set of weights and bias parameters. These parameters are defined for each connection and are what will be tuned during training. The weights and biases for a given connection of two nodes, defines the effect on node has on the other.
In a traditional FFNN the information is passed linearly (in figure 2.1, from left to right) in a process we call

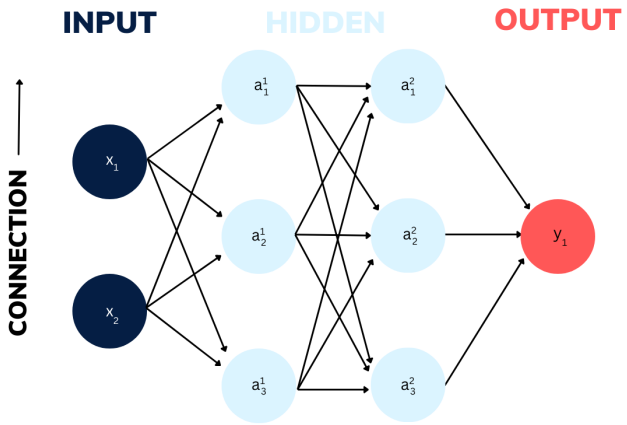



**Figure 2.1:** An illustration of a NN with two hidden layers.

*forward-propogation*. Other variants can include the information taking a more complex route. It is often the route from input- to output-layer that categorizes the type of NN. In this report I used a simple FFNN.

### 2.7.2 Feeding Forward

With the structure described in the previous section, a trained model, $\mathcal{F}$ produces a prediction, $\{\mathbf{y}\}_{i=0}^{N}$ for a data set, $\{\mathbf{x}\}_{i=0}^{N}$, where N is the dimensions of the input, by passing information from input-layer, through all hidden-layers then to output-layer, which we call forward-propagation. In this section I aim to explain the underlying algorithm and math used by the NN to map input to output.
We imagine the passing from hidden-layer $l-1$ to $l$, where $l \in \{2, ..., L\}$[4] and $L$ is equal to the number of hidden layers. The value of a node in layer $l$, is defined as $a_j^l$ (as indicated by figure 2.1), where $j \in \{0, 1, ..., N_l\}$ and $N_l$ is equal to the number of nodes in $l$. The value of $a_j^l$ is defined as the *activated* sum of all nodes in the previous layer, $a_j^{l-1}$ where the sum is weighted by a parameter $\mathbf{w}_\mathrm{j}^\mathrm{l}$ and scaled by the bias, $b_j^l$ for $j \in \{0, 1, ..., N_{l-1}\}$. The inactivated value of for a node j in layer l is defined as

$$z_j^l = \sum_{k=1}^{N_{l-1}} w_{kj}^l a_k^{l-1} + b_k^l, \tag{2.4}$$

where $w_{kj}^l$ corresponds to the weight in $\mathbf{w}_\mathrm{k}^\mathrm{l}$ specific for the connection between node k and j.
To attain the full activated value, $a_j^l$ we pass $z_j^l$ through the *activation function*. The activation function, $\sigma^l$ is a generally non-linear function used to control the limit or expand the value range for the node values. The activation function is general for all nodes in a given layer, but can vary in between layers. Therefore, we find $a_j^l$ by the equation

$$a_j^l = \sigma\left(\sum_{k=1}^{N_l} w_{kj}^l a_k^{l-1} + b_k^l\right) = \sigma^l(z_j^l). \tag{2.5}$$

A more detailed illustration of the information pass from one layer to a node in the next is displayed in figure 2.2. In figure 2.2, we see all steps described in the process; *(1)* all nodes in l-1 are summed with a corresponding weight, *(2)* the sum is scaled by a constant term (bias), *(3)* the scaled and weighted sum defines the inactivated value $z_j^l$, *(4)* we define $a_j^l$ by passing the sum through $\sigma^l$. This method is used for information pass between all layers, except the between the first and the second. In this case we simply replace the activated term, $a_k^{l-1}$ in equation 2.5, by the input data, $x_i$ for $i \in \{0, 1, ..., N\}$, where N is equal to the number features for the data. In the case $l = 1$, 2.5 becomes

$$a_j^1 = \sigma\left(\sum_{k=1}^{N} w_{kj}^1 x_k + b_k^1\right) = \sigma(z_j^1). \tag{2.6}$$

And in the case where $l = L$, $a_j^L$ is equal to the final output.

[4]There is a special case for when $l = 1$ which will be addressed in the next paragraphs.

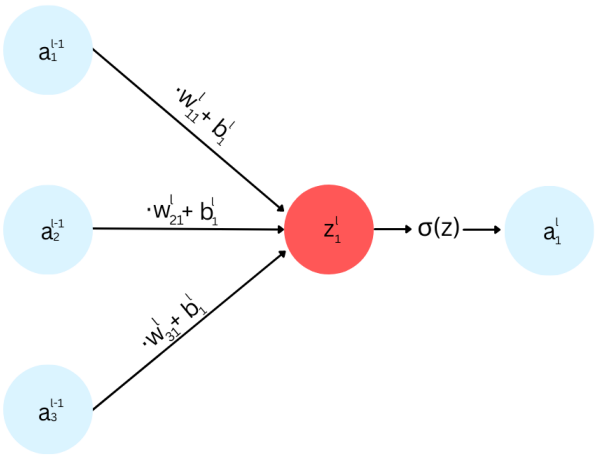



**Figure 2.2:** An illustration information pass for one layer to another.

### 2.7.3 Back Propagation

The backward propagation acts as the engine that drives the training of a neural network. It has the purpose of tuning the weights and biases of the network to achieve maximum performance, as defined by some *cost function*, $\mathcal{C}$. The cost function defines to which metric we aim to optimize the network. In the case a neural network produces a prediction, $\{\mathbf{y}\}_{i=0}^{N}$ which aims to recreate a target $\{\mathbf{t}\}_{i=0}^{N}$ the error in the prediction is defined by the cost function as $\mathcal{C}\left(\{\mathbf{y}\}_{i=0}^{N}, \{\mathbf{t}\}_{i=0}^{N}\right)$. To minimize $\mathcal{C}$, the backwards propagation utilizes the gradient with respect to the weights and biases in the network. Instead of a direct calculation of the gradient, which is very computationally heavy, the backward propagation aims to calculate the gradient through a recursive algorithm which traces the error backwards through the network. It is this algorithm I will describe in this section.

When minimizing the error defined by $\mathcal{C}$, we can apply several optimization algorithms described in section 2.2. Common for the algorithms is the use of the gradient of the cost function with regard to the tunable parameters. In our case these parameters are the weights and biases, $w_{k,j}^{l}$ and $b_k^l$. The goal of the backwards propagation is therefore to calculate the gradients $\partial\mathcal{C}/\partial w_{k,j}^{l}$ and $\partial\mathcal{C}/\partial b_k^l$.

To begin with we can derive an expression for $\partial\mathcal{C}/\partial w_{k,j}^{l}$. We use the chain-rule to define

$$\frac{\partial\mathcal{C}}{\partial w_{k,j}^{l}} = \frac{\partial\mathcal{C}}{\partial z_k^l}\frac{\partial z_k^l}{\partial w_{k,j}^{l}},$$

which we can simplify further by using equation 2.4 to calculate the second term, which becomes

$$\frac{\partial\mathcal{C}}{\partial w_{k,j}^{l}} = \frac{\partial\mathcal{C}}{\partial z_k^l}a_j^{l-1}.$$

We can redefine the first term in the equation above as $\delta_j^l$ and write

$$\delta_k^l = \frac{\partial\mathcal{C}}{\partial z_k^l} = \sum_j \frac{\partial\mathcal{C}}{\partial z_j^{l+1}}\frac{\partial z_j^{l+1}}{\partial z_k^l} = \sum_j \delta_k^{l+1}\frac{\partial z_j^{l+1}}{\partial z_k^l},$$

where we have again used the chain rule for all contributing nodes.
To calculate the final partial derivative we write

$$\frac{\partial z_j^{l+1}}{\partial z_k^l} = \frac{\partial z_j^{l+1}}{\partial a_j^l}\frac{\partial a_j^l}{\partial z_k^l} = w_{jk}^{l+1}(\sigma^l(z_j^l))'.$$

This gives use the expression for $\delta_j^l$

$$\delta_k^l = \sum_k \delta_k^{l+1} w_{jk}^{l+1}(\sigma^l(z_j^l))' \tag{2.7}$$

Finally this gives us the expression

$$\frac{\partial \mathcal{C}}{\partial w_{k,j}^l} = \delta_k^l a_j^{l-1}. \tag{2.8}$$

Next we want to derive $\partial \mathcal{C}/\partial b_k^l$. We simply use the chain rule and derive

$$\frac{\partial \mathcal{C}}{\partial b_k^l} = \frac{\partial \mathcal{C}}{\partial z_k^l}\frac{\partial z_k^l}{\partial b_k^l},$$

which from equations 2.7 and 2.8 is simply

$$\frac{\partial \mathcal{C}}{\partial b_k^l} = \delta_k^l \cdot 1 = \delta_k^l, \tag{2.9}$$

From all three derived expressions, equations 2.7, 2.8 and 2.9 we see that to calculate for all $l \in \{0, 1, .., N_l\}$ we must first calculate $\delta_k^L$ and apply a recursive propagation. To calculate $\delta_k^L = \partial \mathcal{C}/\partial z_k^L$ we simply multiply by $\delta_k^L = \partial a_k^L/\partial a_k^L$, and we find

$$\delta_k^L = \frac{\partial \mathcal{C}}{\partial a_k^L}\left(\sigma^L(z_k^L)\right)' \tag{2.10}$$

This expression, similarly to equation 2.7 is dependent on the choice of $\mathcal{C}$ and the activation functions. Now that equation 2.10 is defined, we can see that the gradient of the parameters in all other layers can be calculated.

### 2.7.4 Activation functions

As mentioned in previous sections, activation functions define how the nodes in the previous layer sum to define the value of the node in the current. There are many types of activation functions, where all have advantages and disadvantages. The choice of activation function for each layer is defined before training, making it a hyperparameter. The activation functions applied and tested in this thesis are the following:

- *Sigmoid*

$$\sigma(z) = \frac{1}{1+e^{-z}} = a \in [0, 1]$$

,

- *Leaky ReLU*

$$\sigma(z) = \begin{cases} z, & \text{if } z \geq 0 \\ \mu z, & \text{otherwise} \end{cases} = a \in (-\infty, \infty),$$

where $\mu$ is scalar.

,

where $z$ is an inactivated node which is activated to define $a$, the activation.

### 2.7.5 Network Ensembles, dropout and LWTA networks

So far in the thesis, I have only covered dense layers, meaning that every node in the layer before it is connected to its own nodes and likewise all its nodes are connected to the nodes in the layer that follows. This definition covers many, but not all hidden layer. Some layers do not function to pass value to and fro between nodes but instead functions by dynamically changing the architecture of the NN. The most popular type of layer that fits this definition, is the *dropout*-layer. The dropout-layer functions by randomly choosing a predetermined number of neurons in the layer before it, and dropping them from the network for the current round of training. By doing this the dropout-layer creates an architecture for the network for every round of training. Each unique architecture represents its own network in what becomes an ensemble of networks.

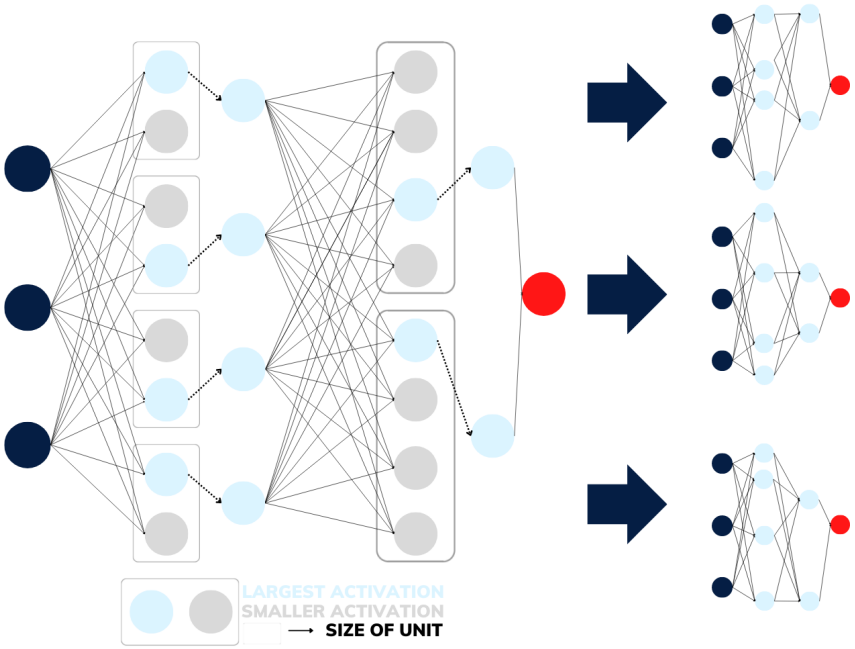



**Figure 2.3:** An illustration of a Neural network with two hidden layers, each with 8 neurons. The first hidden layer has a max-out activation layer with 4 units, the second has 2. The figure also illustrates the resulting ensemble of smaller neural networks as a consequence of the max-out activation layers.

As mentioned in previous sections, creating ensembles of models is a form of regularization. In the case of dropout, it minimizes the risk of overfitting by hindering a phenomenon known as complex co-adaptations. Complex co-adaptation happens when a neuron becomes overly dominant such that neighboring neurons no longer contribute (relative to the dominant neuron) and therefore lack the motivation to tune. When this happens, networks become fragile and overly specialized to the training data. By randomly dropping neurons, the neighboring neurons are no longer allowed to be passive and are forced to tune to compensate. During evaluation, the dropout layers no longer take action. Instead, the dropout rate (fraction of neurons to drop in the layer) $r$ is used to scale the weights by a factor $1 - r$. The prediction made by the resulting model can therefore be seen as the average of all the smaller networks. In other words, a neural network containing dropout-layers as a bagging ensemble.
Dropout-layers are not the only layers to dynamically change the architecture of a network. In this thesis I additionally explored the lesser known, *max-out*-layer. Max-out, similarly to dropout creates an ensemble of networks by removing a set of nodes for each round of training. Contrary to dropout, max-out does not choose the nodes to drop at random, but instead creates local units by grouping the nodes, and removes all nodes except the node with the largest activation. The goal of max-out is to create an ensemble of networks where each network is specialized for a given trend in the data. In this thesis I have implemented the layer such that upon prediction, the redimensionalization is still active. This means that for each data point, a neural network is chosen based on the specific activations in each of the nodes. In other words, the max-out layer creates something resembling a stacking ensemble.
In figure 2.3 I have illustrated a max-out network. The figure shows a NN with two hidden layers with 8 nodes each. In the first hidden layer the max-out layer creates 4 units, each containing two nodes and in the second layer the max-out creates two nodes with 4 nodes each. The resulting output from the first layer is the 4 nodes with the highest activation in their respected unit, likewise the second layers output is 2 nodes. To the right of the network in figure 2.3, I have illustrated how the different configuration of paths through the network creates an ensemble of networks with their own architecture.

## 2.8 Decision Trees and Gradient Boosting

### 2.8.1 Decision Trees

DT are, similarly to NN one of the most popular ML methods used today. Contrary to NN, DT are famously easy in theory. Despite the simplicity of DT, they are capable of a large range of complex problems. In this section I will cover the use of DT as applied to a supervised classification problem.

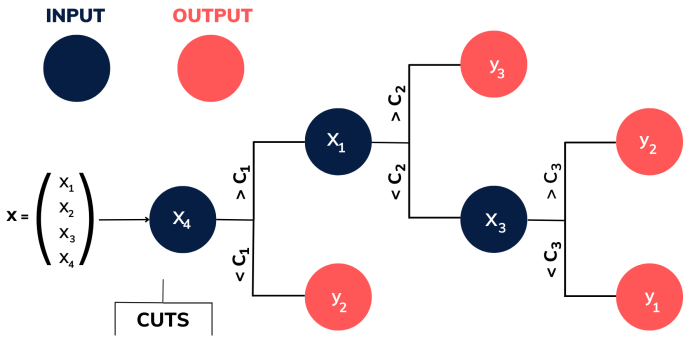



**Figure 2.4:** An illustration of a simple DT, mapping a 4 dimensional input to one of three values in the target space.

Similarly to the traditional cut-and-count method used in particle physics, a DT aims to place *rectangular-cuts* on a data set, $\mathcal{X} = \{\mathbf{x}_i, \mathbf{t}_i\}_{i=0}^{N}$ to effectively map $\mathbf{x}_i \to t_i$. In other words, the goal of a DT is to use $\mathcal{X}$ to from a collection of thresholds $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_{\mathrm{N}_\mathrm{l}}\}$, which when applied to a new data set, $\tilde{\mathcal{X}} = \{\tilde{\mathbf{x}}_i, \tilde{\mathbf{t}}_i\}_{i=0}^{\tilde{N}}$ will sort $\tilde{\mathbf{x}}_i$ to a $\tilde{y}_i$. In figure 2.4, I have illustrated a simple DT classifying a 4-dimensional input data to one of three classifications. As is visualized in figure 2.4, the DT applies a set of thresholds on the data to find the route applicable to a target.
Just like most ML-methods, there are many kinds of DT each with their own architecture and benefits. In the case of DT the defining qualities can be summarized in its choice of *Deepness* and *Optimization*. When provided with a data set, $\mathcal{D}$ a DT can in theory create as many cuts needed to map each individual data point, $\mathbf{x}_i$ to the corresponding target, $t_i$. Doing so would not only be very computationally heavy, but would almost certainly lead to overfitting if applied to any new data. Instead, when building a DT one defines a maximum deepness. This means that we must define a limit to the length of $\mathcal{C}$, which subsequently leads to the need for a prioritization of cuts.
Building a DT and choosing which cuts to apply at what point, is the equivalent of training a network. How to decide by what standard one chooses to building the hierarchy of cuts is again the equivalent of choosing an optimizer.

### 2.8.2 An Introduction to Gradient Boosting

Gradient-boosting is an algorithm which uses a collective of "weak" classifiers in order to create one strong classifier. In the case of gradient-boosted trees the weak classifiers are a collective of shallow trees, which combine to form a classifier that allows for deeper learning. As is the case for most gradient-boosting techniques, the collecting of weak classifiers is an iterative process.
We define an imperfect model $\mathcal{F}_m$, which is a collective of m number of weak classifiers, estimators. A prediction for the model on a given data-points, $x_i$ is defined as $\mathcal{F}_m(x_i)$, and the observed value for the aforementioned data is defined as $y_i$. The goal of the iterative process is to minimize some cost-function $\mathcal{C}$ by introducing a new estimator $h_m$ to compensate for any error, $\mathcal{C}(\mathcal{F}_m(x_i), y_i)$. In other words we define the new estimator as:

$$\tilde{\mathcal{C}}(\mathcal{F}_m(x_i), y_i) = h_m(x_i), \tag{2.11}$$

where we define $\tilde{\mathcal{C}}$ as some relation defined between the observed and predicted values such that when added to the initial prediction we minimize $\mathcal{C}$.
Using our new estimator $h_m$, we can now define a new model as

$$\mathcal{F}_{m+1}(x_i) = \mathcal{F}_m + h_m(x_i). \tag{2.12}$$

Similarly to how we define a deepness of trees, we can define the degree of boosting. We define this as the amount of trees used in the iterative process, or M. This means that the final classifier becomes

$$\mathcal{F}_M(x_i) = \sum_{i=0}^{M} h_i(x_i) \tag{2.13}$$

The XGBoost [6] framework used in this analysis enables a gradient-boosted algorithm, and was initially created for the Higgs ML challenge. Since the challenge, XGBoost has become a favorite for many in the ML

community and has later won many other ML challenges. XGBoost often outperforms ordinary decision trees, but what is gains in results it looses in interpretability. A single tree can easily be analyzed and dissected, but when the number of trees increases this becomes harder.

# Chapter 3

# Implementation

## 3.1 Tools and Data

Every year technology for generating and measuring particle collisions is improved. As a consequence, the amount of data increases drastically. The ATLAS experiment is one of the largest particle detector experiments currently operating at the CERN laboratory near Geneva. ATLAS alone generates approximately 1 petabyte of raw data every second from proton-proton collisions at the Large Hadron Collider (LHC). With amounts of data this large, data handling and storing is a big challenge. Therefore, taking advantage of sophisticated numerical tools and data frameworks is pivotal if scientific development is to keep up with technological development.
In this section I will cover some tools and frameworks I have used to complete my analysis. Large amounts of details and explanations will not be covered. Instead, this section will highlight which tools were used and some motivation for choosing them. Additionally, I will cover some details regarding the data being used, both Monte Carlo (MC) and real.

### 3.1.1 Monte Carlo Data

### 3.1.2 ROOT, RDataframe and Pandas

ROOT [7] is at its core a large $C++$ library and data structure made specifically for big data analysis and computing, as well as data visualization. Today, all ATLAS data is stored as a ROOT-file along with more than 1 exabyte ($10^{12}$) of data worldwide. ROOT has many High Performance Computing (HPC) qualities which makes it ideal for particle physics analysis which demands heavy computations. Additionally, many particle physics-specific packages have been developed to make it an even better tool. Any function not already in library, can easily be added in a HPC-effective manner through $C++$.
All distribution plots made for this thesis were created using ROOT. ROOT has implemented a highly intuitive and effective Application Programming Interface (API) for data comparison and visualization. ROOT allows for quick and direct comparison between data through an advanced graphical user interface. Additionally, a lot of functionality for creating complex stacked histogram are implemented in the ROOT API, such as uncertainty calculations and ordering of histograms.
As for the data structure, the raw data was loaded as a ROOT-file. To easier handle the data and add new features, I used the RDataFrame structure [8]. RDataFrame allows for easy addition of new columns as well as filtering of events through native functionality. As a consequence I used RDataFrame to calculate all higher-level features such as $M_{T2}$, $M_{lll}$ etc. An example is shown in the following section.
After all data handling was complete, I used ROOT's $AsNumpy$ function to translate the data frame as a numpy object, which then allowed me to transform it to a Pandas-data frame [9]. This is done because Pandas, like most ML-tools work in a strict Python environment. Pandas, similarly to RDataFrame includes many deep computational libraries, and is optimal for analysis of big data. When all ML is completed, the data is transformed back to RDataFrame, to take advantage of plotting functionality in ROOT.

### 3.1.3 Computing features in ROOT: Example

In this section I will cover a simple example to highlight the steps taken in generating the dataset I used during analysis. As mentioned earlier, the two main frameworks used were RDataFrame and Pandas. In this example I will cover the case of a feature not already in the ROOT file, namely the trilepton invariant mass. All loading of data is done using the ROOT framework and is quickly made into a RDataFrame. To

effectively generate new features, we want to stay in ROOT and RDataFrame. Therefore, we create our $C++$-file, *helperfunction*. The helperfunction contains all additional ROOT function that are used in the analysis and is not already native to ROOT. In the case of computing the trilepton invariant mass, the $C++$-function is created like shown in 3.1.
In 3.1, we see a couple of measures taken to uphold to the ROOT environment. The first is the typecasting to $VecF\_t$. $VecF\_t$ is created to wrap floats in the native ROOT vector object, $RVec$. The same is done in other cases such as float and integers, $VecI\_t$ and $VecB\_t$. The second measure was using $TLorentzVector$ [10] to calculate the invariant mass. $TLorentzVector$ is a class native to ROOT with many built-in functions. In this case we use the class to create three vectors through the variables, $P_t$, $\eta$, $\phi$ and M. Then, through the $TLorentzVector$ class we can simply add all three vector together, and extract the invariant mass.

```
// Compute the trilepton invariant mass
float ComputeInvariantMass(VecF_t& pt, VecF_t& eta, VecF_t& phi, VecF_t& m) {
  TLorentzVector p1;
  TLorentzVector p2;
  TLorentzVector p3;
  p1.SetPtEtaPhiM(pt[0], eta[0], phi[0], m[0]);
  p2.SetPtEtaPhiM(pt[1], eta[1], phi[1], m[1]);
  p3.SetPtEtaPhiM(pt[2], eta[2], phi[2], m[2]);
  return (p1 + p2 + p3).M();
}
```

**Listing 3.1:** $C++$-function for $M_{lll}$.

With a helper function created in $C++$ we can move over to a Python and RDataFrame environment for calculation and plotting. In the code written in 3.4, I have shown a simple example of loading new $C++$-functions, filtering out bad events, calculating new features and adding said features to a histogram. The first three lines of code both process and include the helperfunction functions into the ROOT framework. Then I loop over all keys in the data frame, which in my case are the different channels (i.e Diboson, $t\bar{t}$ etc.). For each channel I filter out 'good' events, based on the criteria from section 3.2.1. Then, I use RDataFrame's $Define$ function to calculate and add a new feature using our $ComputeInvariantMass$-function. Finally, I save the feature as an $Histo1D$ object, which I plot later using ROOT plotting API's.

```
R.gROOT.ProcessLine(".L helperFunctions.cxx+");
R.gInterpreter.Declare('#include "helperFunctions.h"')
R.gSystem.Load("helperFunctions_cxx.so")

for k in df.keys():
    # Define good leptons
    isGoodLepton = "feature1 < cut1 && feature2 >= cut2"

    # Define good leptons in dataframe
    df[k] = df[k].Define("isGoodLepton",isGoodLepton)

    # Define number of good leptons
    df[k] = df[k].Define("nGoodLeptons","ROOT::VecOps::Sum(isGoodLepton)")

    # Demand 3 good leptons
    df[k] = df[k].Filter("nGoodLeptons == 3")

    # Define Invariant Mass (lll)
    df[k] = df[k].Define("mlll","ComputeInvariantMass(lepPt[isGoodLepton],
                                                      lepEta[isGoodLepton],
                                                      lepPhi[isGoodLepton],
                                                      lepM[isGoodLepton])")
    # Add to histogram
    histo["mlll_%s"%k] = df[k].Histo1D(("mlll_%s"%k,
                                        "mlll_%s"%k,40,50,500),
                                        "mlll",
                                        "wgt_SG")
```

**Listing 3.2:** Python-file for calling dataframe and calculating $M_{lll}$.

In this example I chose the trilepton invariant mass, but in the analysis all high-level features were calculated using a similar method. The workflow of the method is visualized in figure 3.1. We load data in ROOT format, data handling and plot feature distrobution in RDataFrame, perform ML- analysis in Pandas and plot results in RDataFrame.

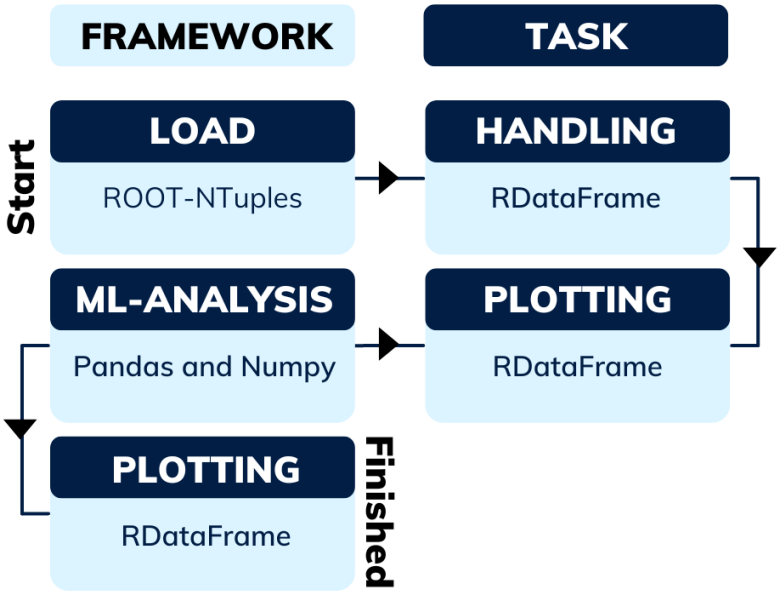



**Figure 3.1:** A visual summary of the workflow and framework use for the computational analysis.

## 3.2 Features

The choice of which features to study and which to neglect are crucial in a search for new physics. This is particularly true in the case of applying machine learning. The general motivation for including a given feature can be based on several factors. The first being its ability to provide a trend which we as researches can exploit when creating our regions. By this I mean that it is a variable were there is diversity in distribution between the different channels. The second motivation is grounded in physics. Often we as physicists tend to lean towards variables we know have some effect one the physics we are studying. For example the variable $E_T^{miss}$, can be directly used to either include or discard events were we do or do not expect final states with sufficient missing energy. The final motivation is grounded in the MC-simmulations ability to represent the variable. If there seems to be a clear deviation between the real and MC-data which does not stem from any new physics, we tend to discard them from the analysis.

### 3.2.1 Cuts and triggers

To allow for deep learning and a thorough analysis one must try and keep as much of the data as possible. At the same time, including large amounts of irrelevant data can be both redundant and destructive. Therefore, simple cuts are necessary. The cuts applied in the analysis were grouped in two definitions, baseline and signal. The baseline requirements are written in table 3.1 and the signal requirements are written in table 3.2. Both sets of requirments were taken from tha ATLAS article from 2022 [11]. Given the definitions we demand that each event contains exactly three signal and three baseline leptons, thereby removing any event with more or less.

Leptons are identified in the detector by using a likelihood-based method combining information from different parts of the detector. The criteria of Loose or Tight identification are simply different thresholds in the discriminant, where Loose is defined as a lower threshold than Tight [12]. The overlap removal is used to solve any cases where the same lepton has been reconstructed as both a muon and an electron. The boolean of *lepPassOr* simply applies a set of requirements to avoid any double counting. The cut for the longitudinal track parameters, $z_0$ is applied to ensure that the leptons originate for the primary vertex.
As for the requirements for the signal leptons, we require all baseline requirements are passed with the addition of a few more. We require Loose isolation for both electrons and muons. This means requiring criteria for a cone around the lepton and is used to suppress QCD-background events. Similarly to the $z_0$-cut, the transverse track parameter is also used to ensure origin from primary vertex.

In addition to the simple cuts, we must insure a good comparison between MC- and real data. Often one finds large deviation between the two in the case of either very large or very small $P_t$. The latter case can often be caused by poor reconstruction or miss identification. These are issues we aim to solve by checking

| Requirement | Baseline electrons | Baseline muons |
|---|---|---|
| Identification | – | Loose |
| Overlap Removal | lepPassOR | lepPassOR |
| $\eta -$ cut | $\lvert\eta\rvert < 2.47$ | $\lvert\eta\rvert < 2.7$ |
| $\lvert z_0 \sin(\theta)\rvert$ cut | $\lvert z_0 \sin(\theta)\rvert < 0.5$ mm | $\lvert z_0 \sin(\theta)\rvert < 0.5$ mm |

**Table 3.1:** Requirments for baseline electrons and muons.

| Requirement | Signal electrons | Signal muons |
|---|---|---|
| Baseline | yes | yes |
| Identification | Tight | – |
| Isolation | LooseVarRad | LooseVarRad |
| $\lvert d_0\rvert / \sigma_{d_0}$ cut | $\lvert d_0\rvert / \sigma_{d_0} < 5.0$ | $\lvert d_0\rvert / \sigma_{d_0} < 3.0$ |

**Table 3.2:** Requirments for signal electrons and muons.

different triggers. Given our data set is composed of different data sets spread over many years, different triggers are used.

### 3.2.2 Lepton variables selection

Now we will have a look at what variables from the leptons that were included in the analysis. All low level information on the momentum of the leptons were added into the dataset: i.e the transverse momentum $P_t$, the pseudo rapidity $\eta$ and the azimuthal angle $\phi$. All momentum features were represented individually for each lepton. For example $P_t$ was added as three columns, $P_t(l_1)$, $P_t(l_2)$ and $P_t(l_3)$, where the ordering of the leptons were based on the momentum from highest ($l_1$) to lowest ($l_3$). Similarly I added information regarding the charge ($\pm$) and flavor (electron, muon) of each lepton. Based on the momentum variables the transverse mass $m_t$ of each lepoton was calculated and included along with the energy $E_t^{miss}$ and azimuthal angle $\phi^{miss}$ of the missing transverse momentum.
The variables described in the section above are often considered as low-level features. These are very useful in many (if not all) searches and contribute little to no bias to your analysis. But, in the case of final-state spesific searches such as mine, one can allow one self of adding physics motivated higher-level physics. The higher level features calculated in this thesis were inspired by [11] (ATLAS 2022).
Firstly I added different mass variables, namely $m_{lll}$ and $m_{ll}(OSSF)$. The first being the trilepton invariant mass and the latter being the dilepton invariant mass of the pair with Opposite Sign Same Flavour (OSSF). In the case of more than one possible OSSF-pair, the pair with the highest invariant mass was chosen. Secondly I added variables composed of the sum of different set of momentum. These variables are the sum of all three leptons $H_t(lll)$, of the pair with Same Sign (SS) $H_t(SS)$ and the sum of the momentum for all three leptons added with the missing transverse energy $H_t(lll) + E_t^{miss}$. Finally I added the significance of the $E_t^{miss}$, $S(E_t^{miss})$.

### 3.2.3 Jet variables selection

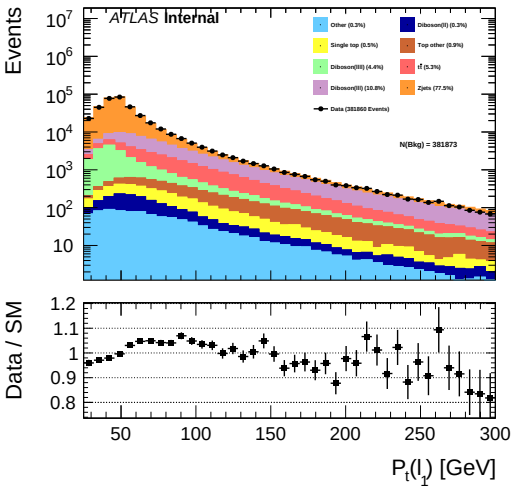
Now we can have a look at the jet-features. Given the final-state of interest should be independant of jets, there are not many features added with jet information. But, given the risk of missidentification and errors in reconstruction, some features were added. The first features were the number of jets, both all signal jets and number of b-jets. The latter information was divided in two columns based on the efficency of a multivarant analysis used to sepereata jet-flavors. The effeciencies used are 77% and %85. The last information added for the jets were the mass of the leading pair (based on $p_t$) di-jet mass.

| 2015 | 2016 | 2017 + 2018 |
|---|---|---|
| HLT_2e12_lhloose_L12EM10VH | HLT_2e17_lhvloose_nod0 | HLT_2e17_lhvloose_nod0_L12EM15VHI |
| HLT_e17_lhloose_mu14 | HLT_e17_lhloose_nod0_mu14 | HLT_e17_lhloose_nod0_mu14 |
| HLT_mu18_mu8noL1 | HLT_mu22_mu8noL1 | HLT_mu22_mu8noL1 |
| | | HLT_2e24_lhvloose_nod0 |

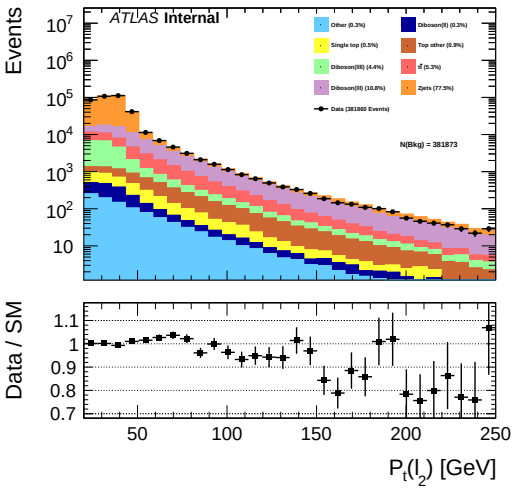
**Table 3.3:** Trigger requirments for events produced in their respective years.

### 3.2.4 Validation

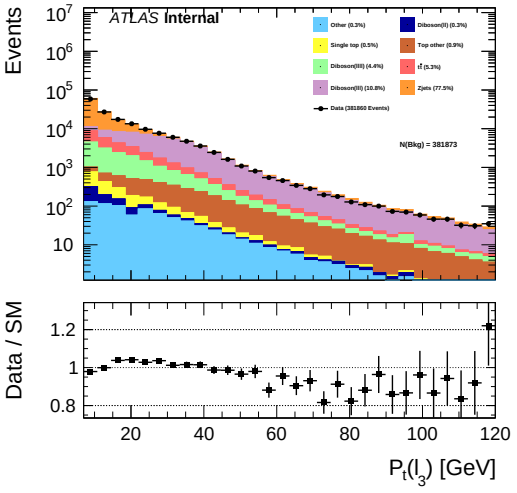
As mentioned in previous sections, the comparison between ML- and real data is a crucial part of the analysis, and we must therefore insure an adequate reconstruction of the real data. This is not only true for the low-level features taken directly from the ML simulation, but also for the the higher-level features. Therefore we will in this section compare both sets of data for all features included in the anlysis.



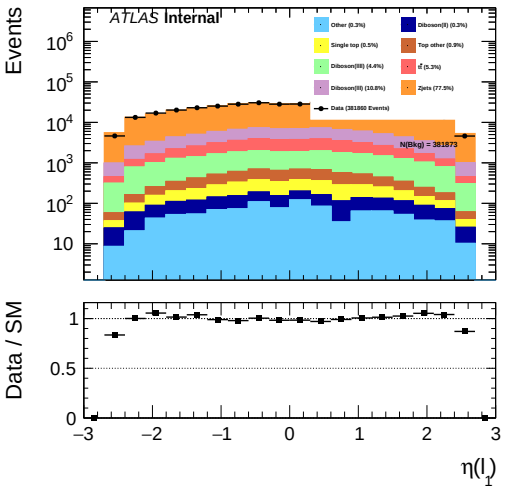

(a)



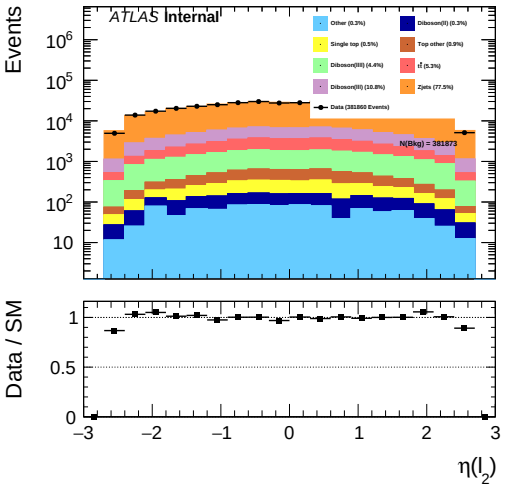

(b)



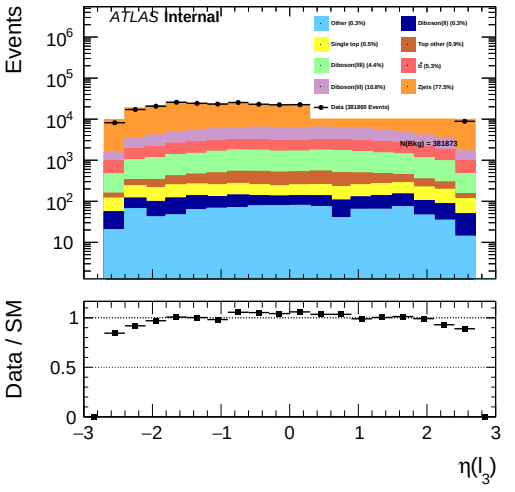

(c)




(d)




(e)




(f)

**Figure 3.2:** The event distribution for for each channel over $P_t$ for the first 3.2a , second 3.2b and third 3.2c lepton. Similarly the distribution over $\eta$ for the 3.2d, second 3.2e and third 3.2f lepton

### 3.2.5 Negative Weights

The negative weight problem is a well known problem in the world of ML-analysis for HEP, and is a consequence of higher perturbative accuracy. For the purpose of visualizing distributions, negative weights are not a problem. But, when using the weights in the training of the classifier, the XGBoost framework will not work.
Before deciding how to mediate the issue, I decided to plot the distribution for a small subset of features for all the events with negative weights.
In figure 3.8a I plotted the absolute value $P_t$ distribution of events with negative weights and compare it to all the events in figure 3.8b. Similarly I have plotted the same values for the second lepton in figures 3.8b and 3.8d respectively. From the two comparisons we see that the distribution of negative weights is relatively equal to that of all the events. The same comparisons have been made in figures 3.9a, 3.9b, 3.9d and 3.9d respectively but for $\phi$. The same observations are made for both $P_t$ and $\phi$. This means we can be justified in treating the negative weight distribution uniformally across the feature space.

How one chooses to deal with this problem can heavily effect results and many solutions are suggested as a consequence. Given the time main focus of this rapport, the simplest solution was chosen. Namely, I chose to normalize all the weights as a whole, conserving the total sum of the weights and at the same time changing all negative signs. The simple procedure is shown bellow in equation 3.1,

$$w_i = P \mid w_i \mid \tag{3.1}$$

$$P = \sum_{j=0}^{N-1} \frac{w_j}{\mid w_j \mid}, \tag{3.2}$$

where $w_i$ is the weight of an event i, $i \in [0, N-1]$ and N equals the total number of events. Although this is a very simplistic solution, our observations made for the distribution of negative weights can be used to justify the approach.

## 3.3 The search through neural networks

### 3.3.1 Creating custom layers

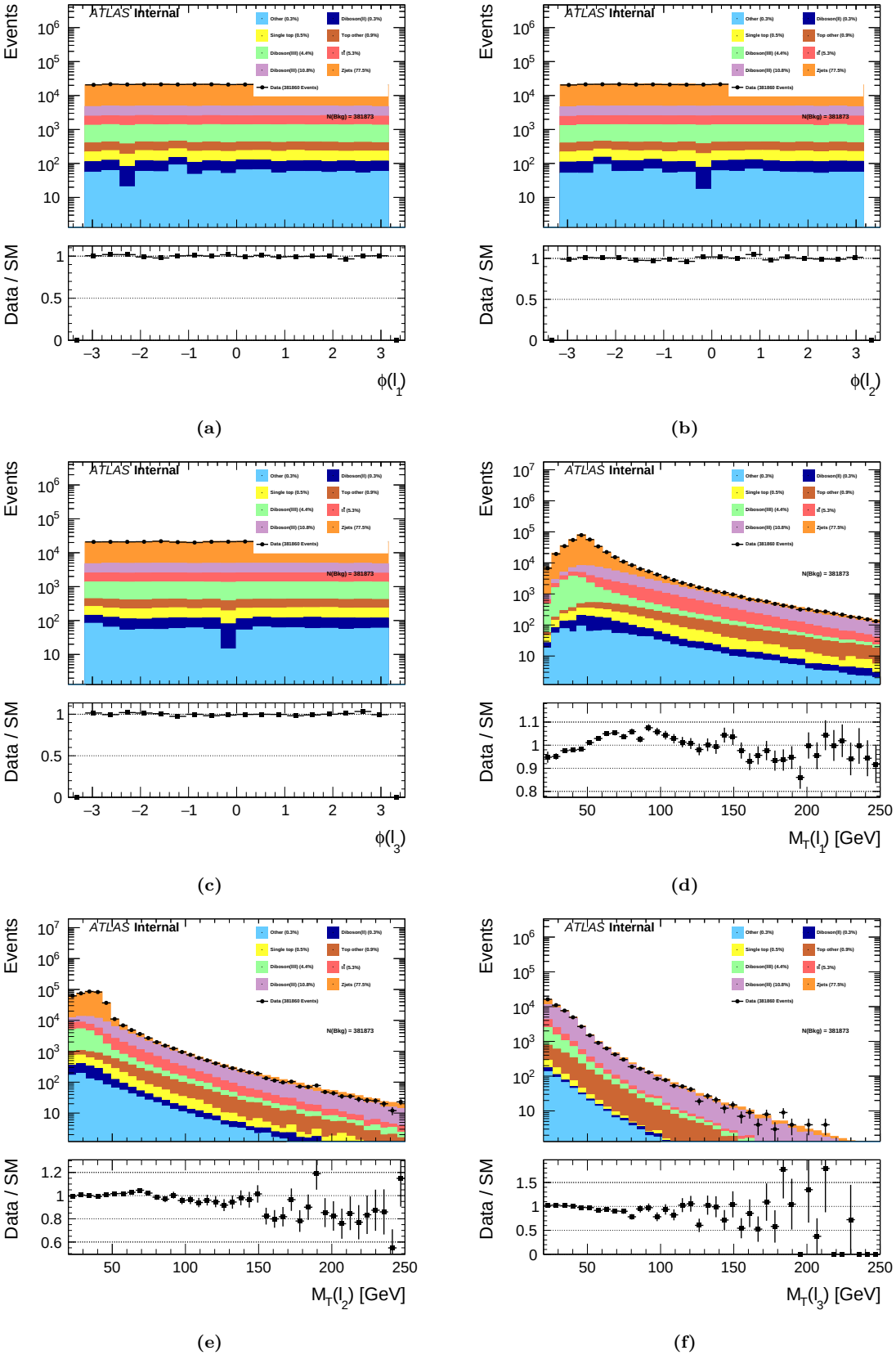
The field of ML is one of the most dynamic and fastest growing fields of research today. This means, that regardless of the brave attempt made by voluntary contributors and the people at Google [5], there will always be new and exciting ML tools, not yet implemented in their library. This was also the case in this thesis. Specifically, in the case of non-dense layers I was forced to dive into the world of ML development and create my own implementation.

#### Max-out

TensorFlow have already implemented a very similar layer called MaxPooling1D. This does exactly what max-out does, but with minor differences. Additionally, to wanting to avoid these differences, I wanted the freedom to experiment with the implementation of the layer. The implementation of both max-out and channel-out was done by creating a custom activation function which is called inside a dense-layer. In the case of max-out, it is in the activation function that the down scaling of nodes happens.

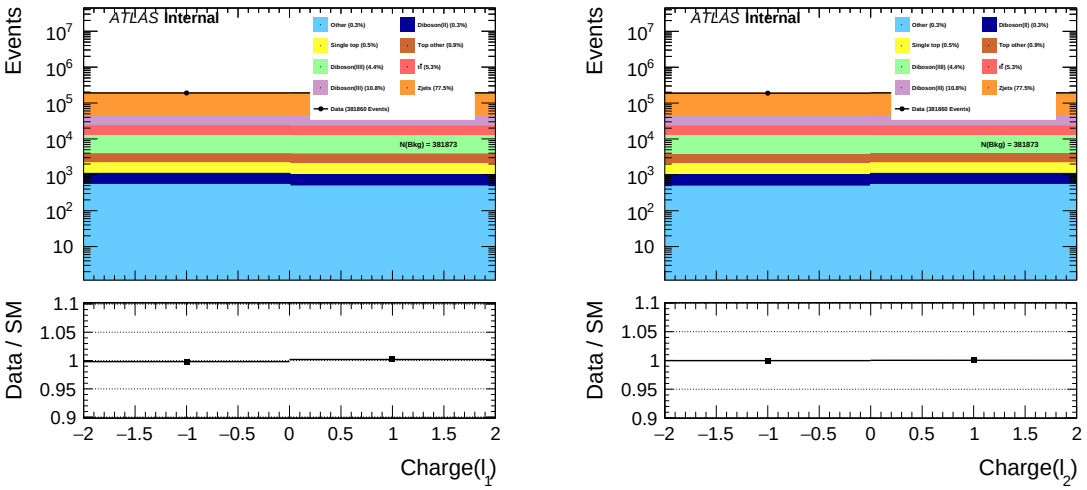
```
def max_out(inputs, num_units = 200, axis=None):
    # Calculate the new shape of the layer after max_out.
    shape = inputs.get_shape().as_list()
    if shape[0] is None:
        shape[0] = -1
    if axis is None:  # Assume that channel is the last dimension
        axis = -1
    num_channels = shape[axis]
    if num_channels % num_units:
        raise ValueError('Number of features is not a multiple of num_units')
    shape[axis] = num_units
    shape += [num_channels // num_units]

    # Calculate the largest value in each unit and discard
    # the rest.
    outputs = tf.reduce_max(tf.reshape(inputs, shape), -1, keepdims=False)
    return outputs
```

[5] The developers of TensorFlow




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

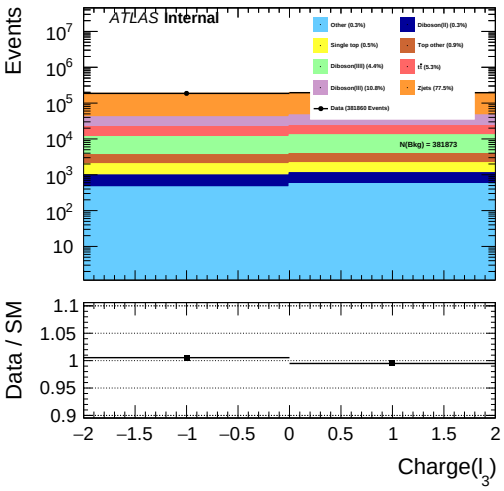
**Figure 3.3:** The event distribution for for each channel over $\phi$ for the first 3.3a , second 3.3b and third 3.3c lepton. Similarly the distribution over $m_t$ for the first 3.3d, second 3.3e and third 3.3f lepton



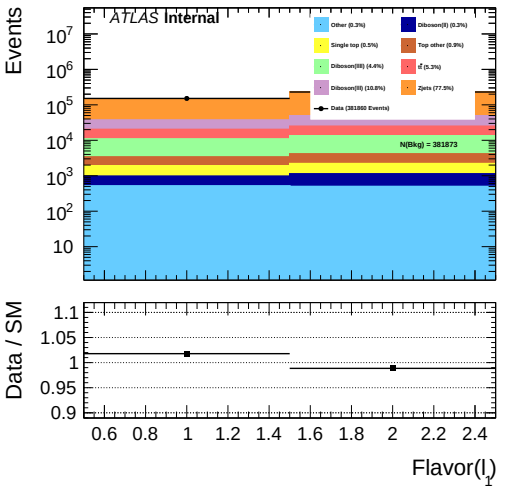

(a)



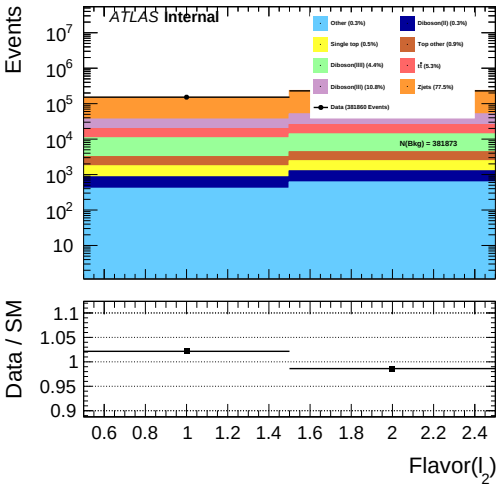

(b)



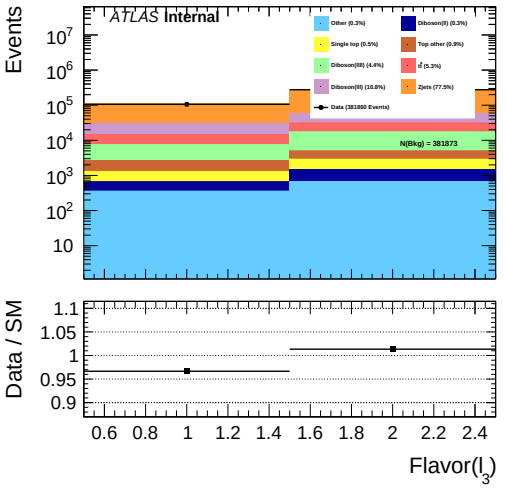

(c)




(d)

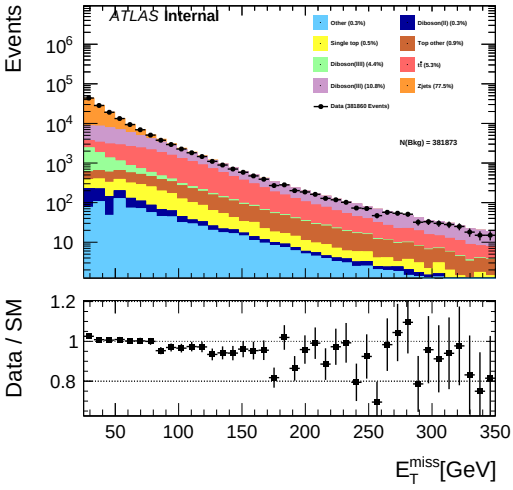



(e)

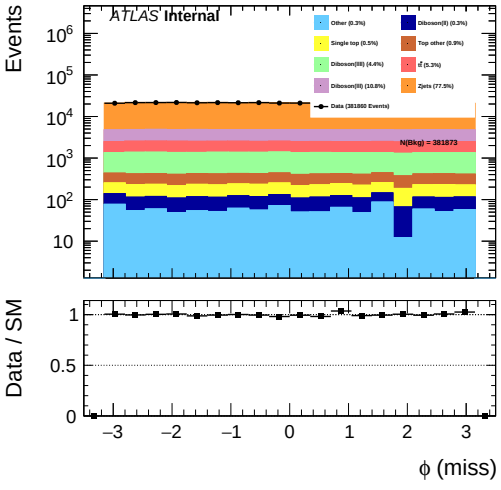



(f)

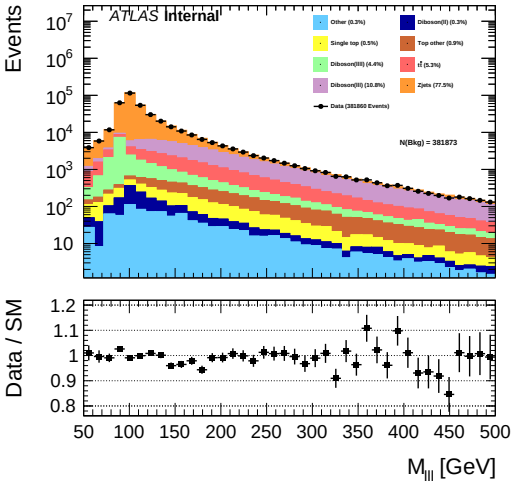
**Figure 3.4:** The event distribution for for each channel over the charge for the first 3.4a , second 3.4b and third 3.4c lepton. Similarly the distribution over the flavor for the first 3.4d, second 3.4e and third 3.4f lepton



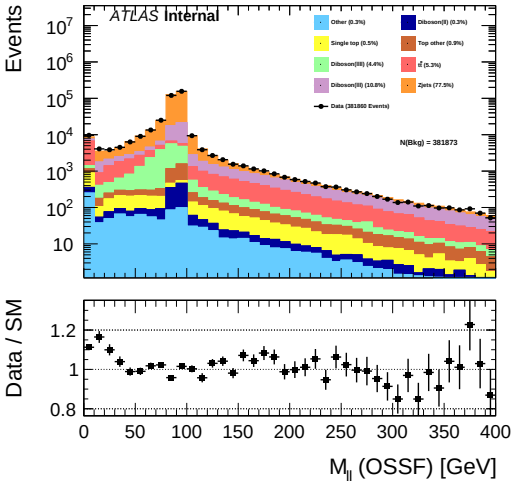

(a)



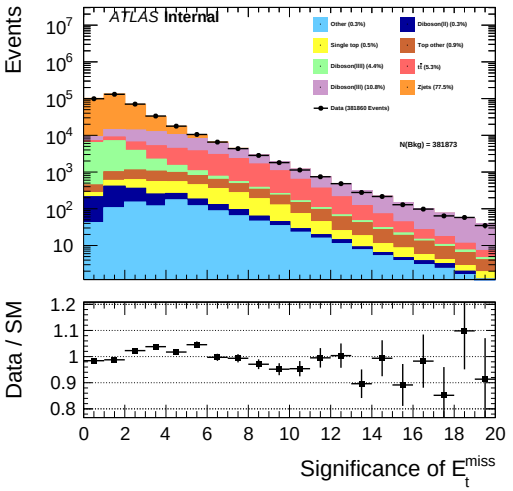

(b)



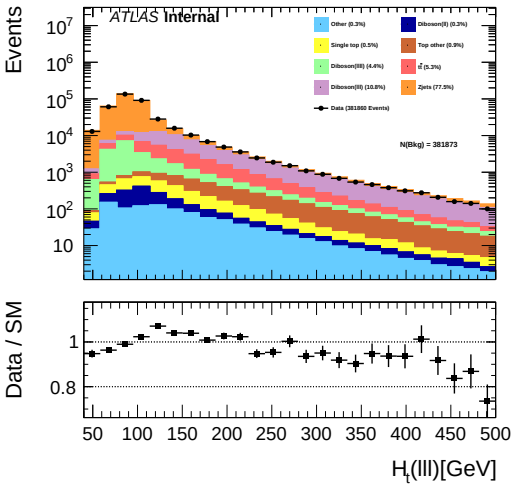

(c)




(d)

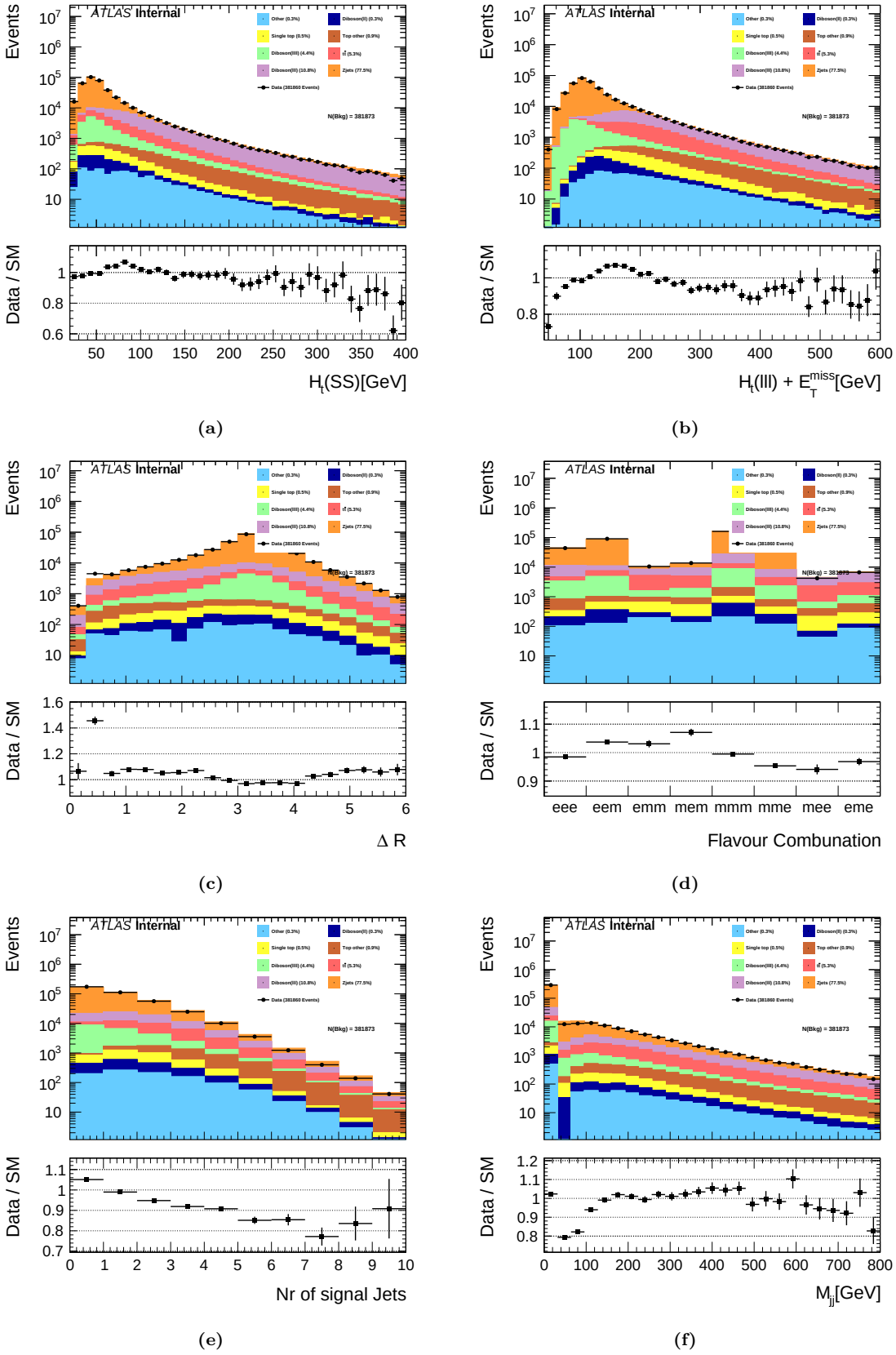



(e)



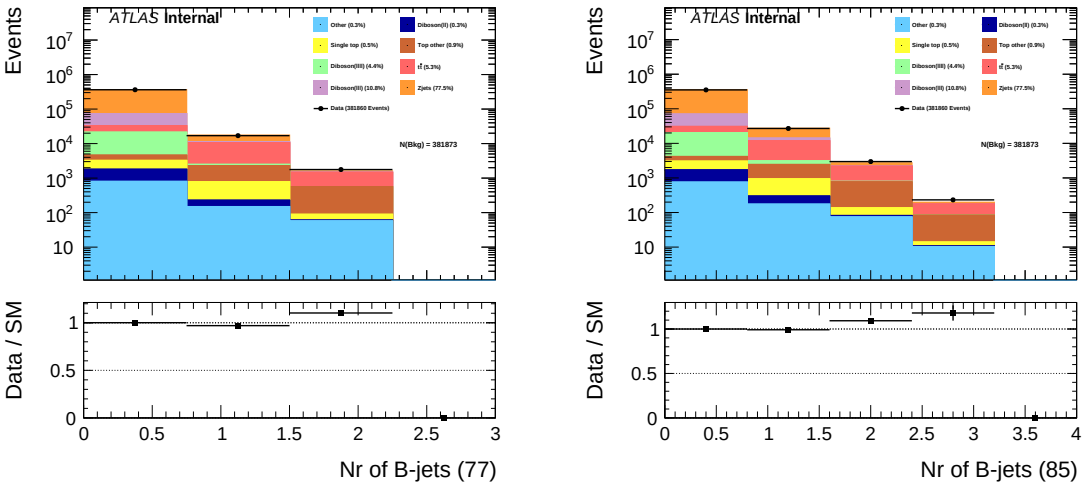

(f)

**Figure 3.5:** The event distribution for for each channel over the energy 3.5a and azimuthal angel 3.5b for the transverse momentum. The distribution of the invariant mass of the three leptons 3.5c and the OSSF pair 3.5d. The distribution over the significance of the missing transverse enegy 3.5e and the sum of $P_t$ 3.5f.



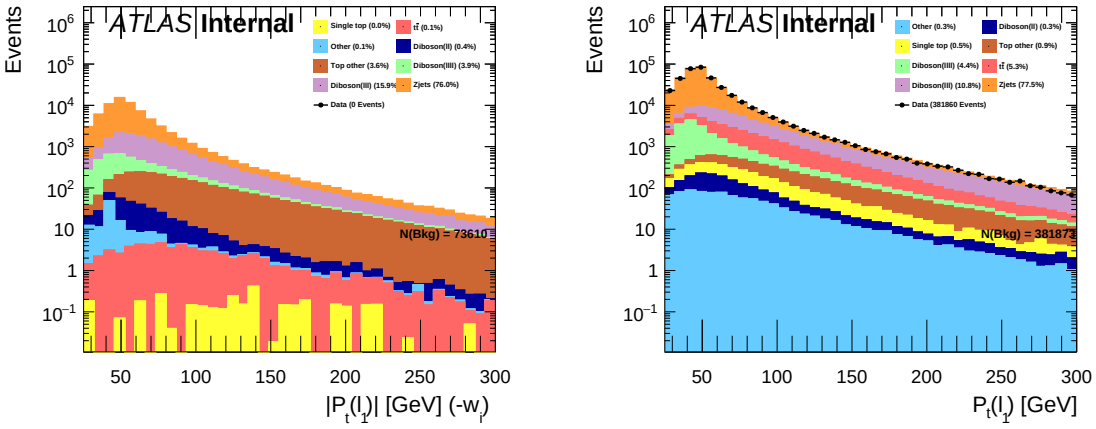

(a) (b) (c) (d) (e) (f)

**Figure 3.6:** The event distribution for for each channel over the sum of $P_t$ for the SS pair 3.6a and the sum over all three leptons added with $E_t^{miss}$ 3.6b. The distribution over $\Delta R$ 3.6c and the flavor combination of the three leptons 3.6d. The distribution of number of jets 3.6e and the mass of the leading di-jet pair 3.6f.

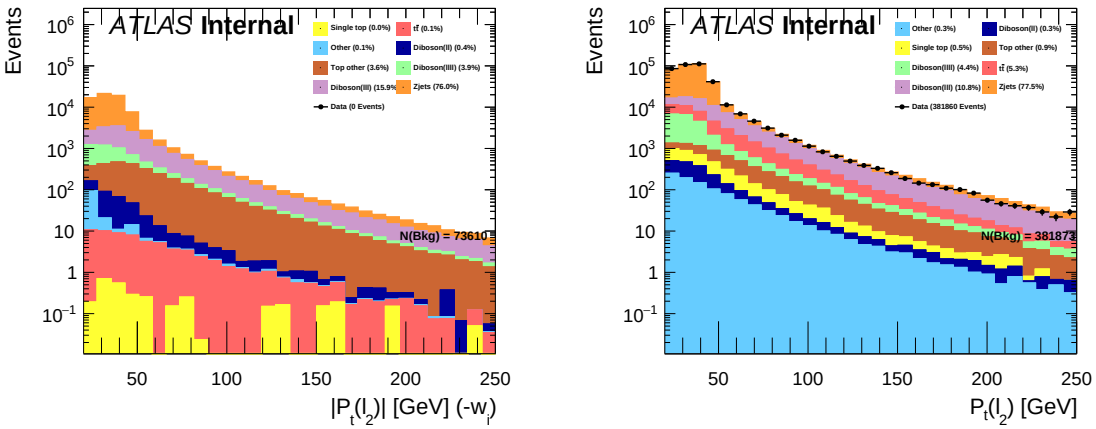



(a) (b)

**Figure 3.7:** The event distribution for for each channel over the number of b-jets with 77% 3.7a and 85% 3.7b efficency.

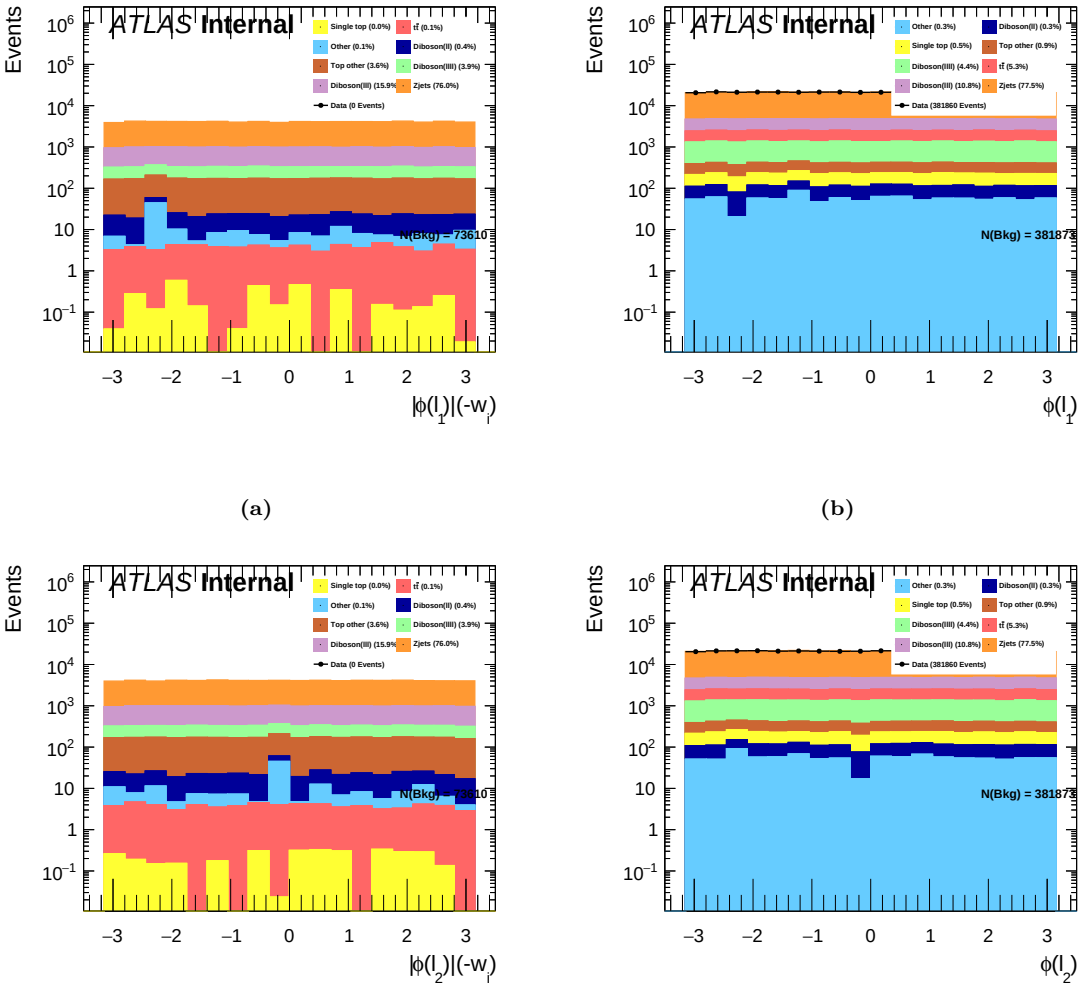



(a) (b)




(c) (d)

**Figure 3.8:** The absolute value of the $P_t$ for events with stricly negative weights ($l_1$ 3.8a, $l_2$ 3.8c) and all events ($l_1$ 3.8b, $l_2$ 3.8d).




(a) (b)

(c) (d)

**Figure 3.9:** The absolute value of the $\phi$ for events with strictly negative weights ($l_1$ 3.9a, $l_2$ 3.9c) and all events ($l_1$ 3.9b, $l_2$ 3.9d).

**Listing 3.3:** Python-file for calling dataframe and calculating $M_{lu}$.

### Channel-out

TensorFlow have already implemented a

```
def channel_out(inputs, num_units = 200, axis=None, training=None):
    shape = inputs.get_shape().as_list()
    if shape[0] is None:
        shape[0] = -1
    if axis is None:  # Assume that channel is the last dimension
        axis = -1
    num_channels = shape[axis]

    if num_channels % num_units:
        raise ValueError('number of features({}) is not '
                         'a multiple of num_units({})'.format(num_channels,
                             num_units))
    shape[axis] = num_units
    shape += [num_channels // num_units]
    grouped = tf.reshape(inputs, shape)
    top_vals = tf.reduce_max(grouped, -1, keepdims=True)
    isMax = tf.reshape(tf.greater_equal(grouped, top_vals), [shape[0],
        num_channels])
    output = tf.multiply(tf.cast(isMax,tf.float32), inputs)
    return output
```

**Listing 3.4:** Python-file for calling dataframe and calculating $M_{lu}$.

# Appendices

# Appendix A

# Acronyms

**API** Application Programming Interface

**AUC** Area Under the Curve

**CNN** Convolutional Neural Network

**CP** Charge-Parity

**DNN** Deep Neural Networks

**DT** Decision Trees

**FFNN** Feed-Forward Neural Network

**HPC** High Performance Computing

**LHC** Large Hadron Collider

**MC** Monte Carlo

**ML** Machine Learning

**MSE** Mean Squared Error

**NN** Neural Network

**OSSF** Opposite Sign Same Flavour

**QCD** Quantum Chromo Dynamics

**QED** Quantum Electro Dynamics

**RNN** Recursive Neural Network

**ROC** Receiver Operating Characteristic

**SM** Standard Model

**SR** Signal region

**SS** Same Sign

# Bibliography


[1] J. Joyce, *Finnegans Wake*. Penguin Books, New York, 1999.

[2] W. Thomson, *Lord kelvin addresed the british association for the advancement of science*, 1900.

[3] D. Guest, J. Collado, P. Baldi, S.-C. Hsu, G. Urban and D. Whiteson, *Jet flavor classification in high-energy physics with deep neural networks*, *Physical Review D* **94** (dec, 2016) .

[4] J. Pumplin, *How to tell quark jets from gluon jets*, *Phys. Rev. D* **44** (Oct, 1991) 2025–2032.

[5] Scikit-learn developers, "Pandas."

[6] The XGBoost Contributors, "Xgboost."

[7] The Root team, "Root."

[8] The Root team, "Rdataframe."

[9] The Pandas team and voluntary contributers, "Pandas."

[10] The Root team, "Tlorentzvector."

[11] M. Franchini, K. H. Mankinen, G. Carratta, F. Scutti, A. Gorisek, E. Lytken et al., *Search for type-III seesaw heavy leptons in dilepton final states in $pp$ collisions at $\sqrt{s}$ = 13 TeV with the ATLAS detector*, .

[12] M. Aaboud, , G. Aad, B. Abbott, D. C. Abbott, O. Abdinov et al., *Electron reconstruction and identification in the ATLAS experiment using the 2015 and 2016 LHC proton–proton collision data at $$\sqrt{s} = 13$$ $$\text{TeV}$$*, *The European Physical Journal C* **79** (aug, 2019) .