UNIVERSITY
OF OSLO

Master's thesis

Search for BSM-physics 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

23rd March 2023

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 great 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]. In spite of the success of the SM, there are still questions we are unable to answer. SM is yet to merge the physics of the micro and macro or even accommedate the particles that make up most of the universe. Most people believe this is not a sign to discard the SM, but instead to expand it!

1.1 The bulding blocks

As early as ancient Greece, humans pondered the nature of the most elementary building blocks of the universe. The Greeks imagined a rope of a given length and a pair of scissors with 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 [2]. The particle that Thomson discovered was 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 *1871-1937* [3]) 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 [4] when we discovered the (at that time thought of as massless) neutrino. Together, the electron and the neutrino form the leptons. We refer to both the leptons and the quarks as fermions.
Upon the evolution of quantum mechanics and 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 experiments, 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 gauge 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 effects, such as the ones allowing us to see objects using our eyes. Similarly, the $W^{\pm}$ 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 neutrons together. Gravity is the only force not included 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 Robert Brout, Francois Englert and Peter Higgs in 1964 and discovered at CERN in 2012. The Higgs boson, sometimes called the God particle is responsible for giving particles mass in a process called spontaneous symmetry breaking of the electroweak theory [5]. Together the fermions and the bosons make up all the particles in the SM.

[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
$1st$	ν_e	< 0.001	0
$2nd$	μ	105.66	-1
$2nd$	ν_μ	< 0.17	0
$3rd$	τ	1776.8	-1
$3rd$	ν_τ	< 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 particles with half-integer spin[2], $\pm 1/2$. A lepton can be either charged or neutral. For reasons that are yet to be known, the leptons come in 3 generations. Each generation contains a pair of charged and neutral lepton. The first generation contains the electron, e^- and the electron-neutrino, ν_e. The second contains the muon, μ and the muon-neutrino, ν_μ. And the third generation contain the tau, τ^- and ν_τ. 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 often neglect the τ 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 10^9eV. 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 [6], 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.[3] 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 is found, along with the respective mass and electric 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.' [7]

The poem above was written by James Joyce (*1882-1941*) in 1939, and was the motivation for Gell-Mann (*1929-2019*) when naming the inner particles of hadrons, quarks. Quarks were introduced to explain some strong-force properties of hadrons. We can categorize quarks as being either positively charged or negatively charged. All down-type quarks have a negative charge equal to 1/3 that of the electron (e) and all positive quarks have a positive charge equal to 2/3 that of the electron (+e). 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 positive and one negative 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 more energy is need to create them.
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. QCD is what allows quarks to change color. It predicts asymptotic freedom when quarks are free at short distances, also known as color

[2]Spin is a quantum number which predicts the effect of an applied electromagnetic field.
[3]The lightest neutrino μ_e, is found to have an upper bound of 1eV.

confinement. Briefly explained, color confinement results in quarks never existing in isolation but always in a quark-antiquark pair, mesons or in three quark state, 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 of hadrons.

1.2 The Forces

Why do the nuclei of atoms stick together? Why do the electrons revolve around said nuclei? And why can neutrons decay to photons and vice versa? These phenomena are all described through the interactions of leptons and quarks. But how do we describe the interactions? In the previous section I briefly mentioned that the interactions of the leptons and quarks (or fermions) are mediated by the gauge bosons. The explanation of force as a mediation of bosons is the cornerstone of the SM and is explained through the introduction of Quantum Field Theory (QFT). QFT is a precise mathematical framework which relies on the properties of local symmetries (Gauge symmetries) and field theory. Given the scope of this thesis an introduction to QFT will not be given, yet certain attributes of the forces themselves are of interest.

1.2.1 Electromagnetism

Electromagnetism is one of the two macroscopic forces[4]. This means that most people have directly experienced it and therefore have built an intuition for it. If you place two oppositely charged objects close enough, they attract. The closer they are, the more they attract. Electric charge is the property that causes particles to experience electromagnetic forces. This is due to the boson responsible for mediating it, the photon (γ). The photon is massless, and only couples to particles with an electric charge, which means only particles with a non-zero charge can interact through the electromagnetic force.

1.2.2 The Strong Force

The strong force is, alongside the weak force a microscopic force, meaning that although we never experience it directly. The strong force holds nuclei together and is (as the name suggests) the strongest force. Similarly to how electric charge plays a role in electromagnetism, the strong force has *color*. Not to be mistaken with the spectrum of frequencies of light, color in the SM is known as the charges associated with the strong force. Where there is one electric charge, color is instead a collection of three conserved charges, 'r', 'b' and 'g'. Like the photon, the gluons carry color and anticolor and only couple to colorful[5] particles. Due to the three color charges, there exists a collection of independent color states, 8 which corresponds to the number of gluons. The only particles with color are the quarks, antiquarks and gluons, which explain why only they experience the strong force.

1.2.3 The Weak Force

The weak force is the weakest of all the forces of the SM. This is due to the bosons coupling weakly to fermions, and the fact that the $W^{\pm}$ and Z are very heavy. The charged weak force is the only force which allows for flavor change[6]. They interact only with left-handed particles. The charge associated with the weak force is the weak isospin. All left-handed fermions and right handed-handed antifermions have non-zero isospin (1/2), meaning the W and Z bosons interact with both the leptons and the quarks.

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.' [8]

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 was

[4]The other being gravity.
[5]Meaning particles with a non-zero color charge.
[6]Flavor is a term used to differentiate the fermions, i.e. the six leptons (electron, muon, electron neutrino etc.) and six quarks (top, bottom, charm etc.). A change in flavor therefor means to transition from one of this flavor to another through the weak charge bosons. For example $\mu_e \rightarrow e^- W^+$

followed by a long period of advancements in the field of physics by the likes of Max Planck (*1858–1947*) and Albert Einstein (*1879–1955*). Less than half a decade after Lord Kelvin uttered the famous words, 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 present some of the problems we hope to tackle in the future.

- SM in its current form cannot incorporate *gravity*, as it does not have sufficient symmetry. The hope has been to integrate gravity into SM through the discovery of a gravity-carrying particle, the gravtion [9]. So far, no-such particle is found.
- *Dark matter* and *dark energy* make up more than 90% of the mass in the observable universe, but is found to lie beyond the SM [10].
- 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; what is the origin of the neutrino mass and is Charge-Parity (CP) violated in the neutrino sector?

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 exciting) 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 ν_e, ν_μ and ν_τ. Flavor mixing is in 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 Brout-Englert-Higgs (BEH) field. For particles to gain mass they need to have a right- and left-handed[7] particle. So far, no right-handed neutrinos have been observed, due to them being sterile[8] in the SM. 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 as far as mass is concerned. 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.

1.3.3 Super Symmetry and the Neutralino

A supersymmetric theory has for many years been an interesting candidate for a new physics beyond the standard model. Supersymmetry (SUSY) aims to alter the standard model such that mass and force (or fermions and bosons) is treated equally. SUSY suggests that each SM particle has an additional Superpartner (SP) which we call a sparticle. The sparticles all differ with half a spin from the original SM particle. Because SUSY is a broken symmetry, sparticles are expected to be much higher than the corresponding SM masses, often in the range of 100- 1000GeV. The difference in spin means that the SP of a fermion is a scalar boson and the SP of a boson is a fermion. SUSY is a candidate to fix many problems in physics, some of which are; the hierarchy problem, fixing the mass of the higgs and possibly the mystery of dark matter. For a more thorough explanation of SUSY and its application, the reader is referred to Refs.[11]

[7] For a massles particle NB FIX .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.
[8] Non-interacting.

1.4 Proton-Proton collisions at the LHC

1.4.1 An Introduction to Particle Accelerators and Detecors

With a circumference of 27 km, the *particle accelerator* Large Hadron Collider (LHC) is the largest piece of scientific equipment ever built. It consists of two separate large tubes aligned with powerful magnets. The magnets are used to accelerate bunches[9] of hadrons (specifically protons) and lead. One set of bunches are accelerated in one of the tubes, and another set is accelerated in the other direction inside the other tube. The bunches are accelerated up to close (v=0.99999991c) the speed of light before they collide at a rate of once every 25 nanosecond. The released energy from the collisions enable the creations of new particles.
To measure the collisions, we use *particle detectors*. In this thesis I will be using data collected by the ATLAS (A Toroidal LHC Apparatus) detector. The ATLAS-detector is the largest general-purpose detector at the LHC and was first active in 2009. The detector consists of several ever-increasing cylindrical layers around the point of collision. In figure 1.1, taken from the ATLAS collaboration [12] the cross section of the detector along with the paths of different particles is visualized. The inside of a detector can be summarized in the following points, listed from innermost to outermost layer:

- *Inner Detector*: The inner detector consists of three layers, Pixel detector, Semi-Conductor Tracker and Transition Radiation Tracker. Its purpose is to measure electromagnetic interactions between the particles produced in the collision and the material in the layers. The measurements are made at discrete points and can be used to infer the momentum of the particles. An electromagnetic field is applied to the inner detector to curve the paths of the particles, so to measure properties of the particles like spin and charge.
- *Calorimeters*: The ATLAS detector has two calorimeters, the *Electromagnetic calorimeter* and the *Hadronic calorimeter*. The Electromagnetic (EM) calorimeter is the first layer of the two and measures the energy of particles interacting through the EM field. The hadronic calorimeter is designed to measure the energy of hadrons (i.e. protons, neutrons, mesons etc.).
- *Muon Spectrometer*: The muon spectrometer is included to detect muons, which would otherwise not be measured by the previous layers. Similarly to the inner detector, the muon spectrometer applies a EM field to curve the path of the particles which allows it to measure several properties.

1.4.2 The Kinematics

The kinematic variables of a particle collision are crucial in any High Energy Physics (HEP) analysis and are easily explained using simple geometry. In figure 1.2 I have drawn a simple axis to illustrate the kinematics of a particle in both the longitudinal (left) and transverse (right) plane. In a three-dimensional axis where the particles colliding travel along the z-axis, the zy-axis defines the longitudinal plane. The angle between the z-axis and the direction of the momentum of one of the particles, define the polar angle of said particle, θ. The xy-axis define the transverse plane and the angle between the x-axis and the direction of the transverse momentum define the azimuthal angle, ϕ. The transverse momentum and the azimuthal angel are both used in this analysis and are used to create further features. Instead of the polar angle, a preferred feature is the pseudorapidity, η. The pseudorapidity is defined as

$$\eta = -ln\left[tan\left(\frac{\theta}{2}\right)\right] \tag{1.1}$$

The pseudorapidity is a preferred feautre to the polar angle because differences in η are lorentz invariant under boosts along the longitudial axis. A large range of features are created using the variables described above. In the appendix I have added a summary of all the features used in this analysis (see table 2). One of said features is the distance between the particle in the $\eta - \phi$-axis, ΔR. We define ΔR as

$$\Delta R = \sqrt{(\Delta\eta)^2 + (\Delta\phi)^2}. \tag{1.2}$$

1.5 The Background Channels

We define background as anything that is not of interest and that mimics the signals of new physics considered. As will be further explained in later sections, we will explicitly demand a 3-lepton final-state in all collisions

[9] Small pockets of particles

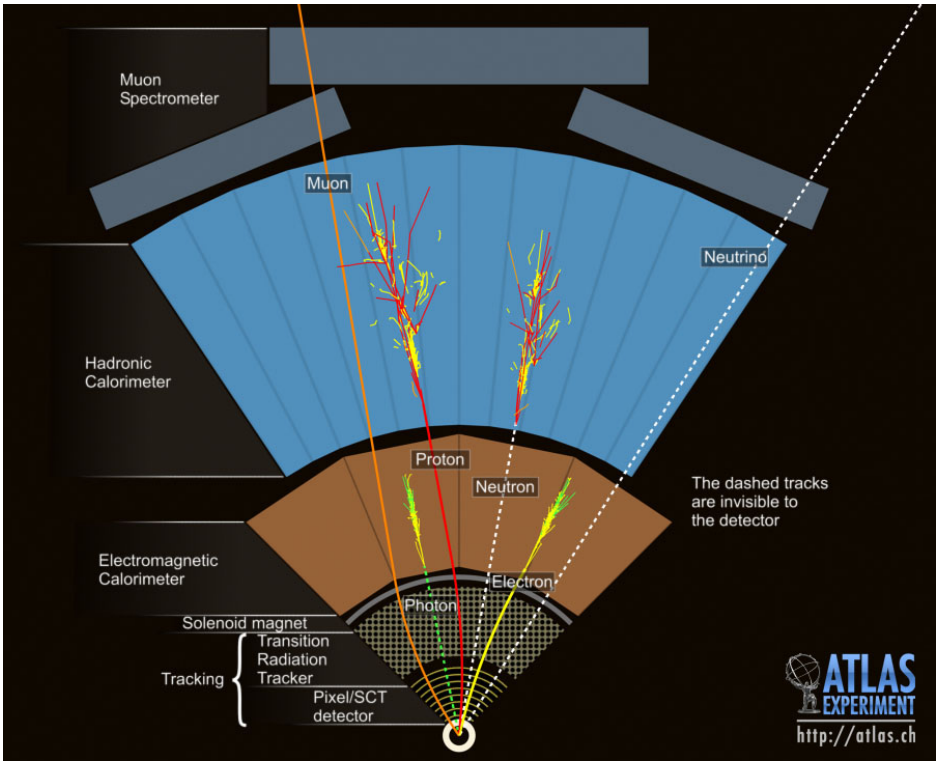


Figure 1.1: Event Cross Section in a computer generated image of the ATLAS detector [12].

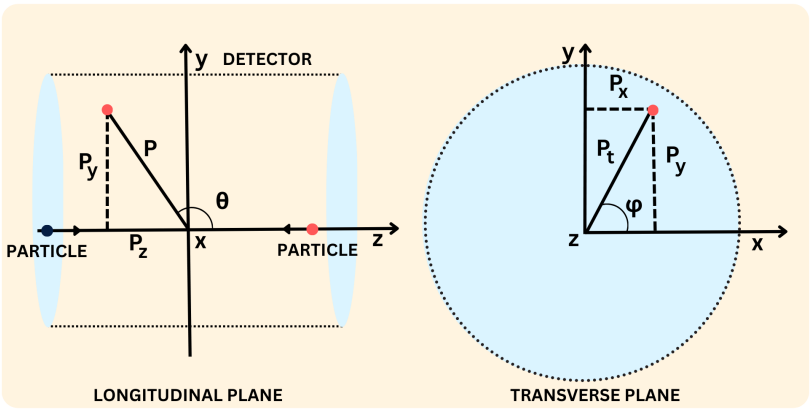


Figure 1.2: An illustration of the general kinematics in a particle-collision, inspired by the figure in the paper by Gramstad [13]. The illustration shows both the longitudial and transverse plane.

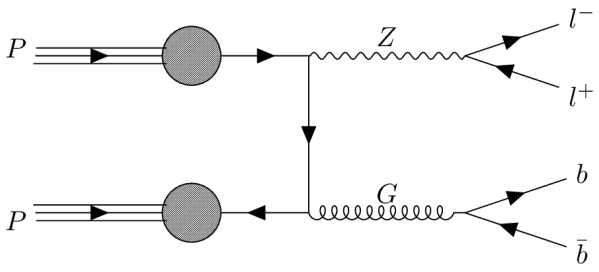


Figure 1.3: The Feynman diagram of the Z-jets channel.

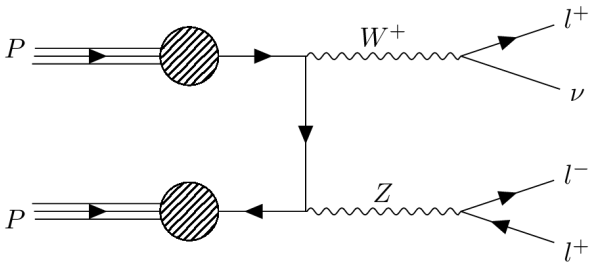


Figure 1.4: The Feynman diagram of the diboson WZ-channel.

considered in the analysis. This will remove a lot of background, but not all. Due to the imperfect nature of the reconstruction of events, a demand for 3-lepton final-state will not be without errors. This leaves room for more variation in the background than one might expect. In this section I will cover the channels[10] which will be of importance during the analysis. I will also discuss which background channels are the hardest to reduce in a potential new physics signal region, also called the irreducible background. These are channels that exhibit similar trends/distribution in the features. Note that the sections bellow are listed by contribution to the 3-lepton final-state data (from most to least).

1.5.1 Z-jets

The Z-jets channel is the largest contribution in all the data. The channel consists of all events resulting in a Z-boson alongside jets. In the cases the Z-boson decays into two leptons, the additional jet acts as a fake lepton and the channel classifies as a 3-lepton final state. In figure 1.3 I have written the Feynman diagram of an example of such a channel. The figure shows a quark-antiquark leading to a Z-boson and gluon. The Z boson decays into two leptons (normally e^-e^+ or $\mu^-\mu^+$) and the gluon hadronises as a jet of hadrons which may obtain a b-hadron leading to a fake lepton.

1.5.2 Diboson (lll)

Dibson channels are defined as channels resulting in two bosons. In the case of (lll), the dibosons decay into a total of three leptons. In figure 1.4 I have drawn the Feynman diagram of an example of such a channel. The figure shows a W- and Z-boson production through a quark-antiquark pair. third quark. The W-boson decays into a lepton with missing energy[11] and the Z-boson decays into a pair of leptons.

1.5.3 $t\bar{t}$

The $t\bar{t}$ channel is defined as a proton-proton collision resulting in a pair of top quark-antiquark. In figure 1.5 I have drawn a Feynman diagram of an example of such a channel. The figure shows gluon-gluon fusion producing a pair of top quarks. The top-antitop pair decay into a bottom-quark and a W boson. The channel constitutes a background when both W bosons decay into a charged and a neutral lepton and one of the b-quarks leads to a fake lepton.

[10] By channels, we refer to the physical diversity which could lead to a specific final-state. Often this refers to the different particles which mediates the process from initial- to final-state.

[11] Neutrinos very rarely interact with anything, making them almost impossible to detect. We therefore refer to neutrinos as missing energy.

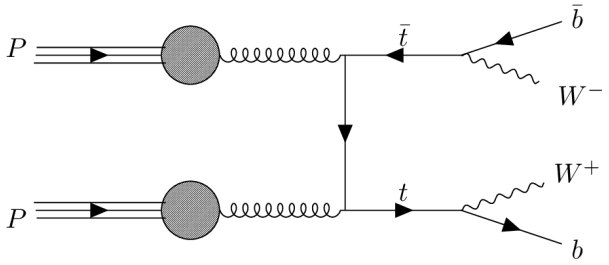


Figure 1.5: The Feynman diagram of the $t\bar{t}$-channel.

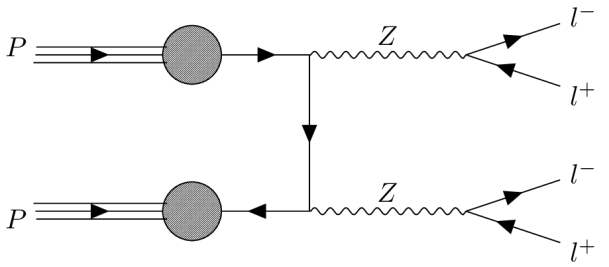


Figure 1.6: The Feynman diagram of the ZZ-channel.

1.5.4 Diboson (llll)

In the case of diboson (llll), the channel refers to events resulting in two Z-bosons which decay into four leptons. In figure 1.6 I have drawn a Feynman diagram of an example of such a diagram. The figure shows a quark-antiquark pair annihilating into two Z-bosons. The two Z-bosons decay into two pairs of leptons. This process constitutes a backgorund when one of the leptons is not reconstructed in the detector.

1.5.5 Top Others

1.5.6 Single Others

1.5.7 Diboson (ll)

1.5.8 Others

1.6 The Signal

1.6.1 Heavy Neutral Lepton

1.6.2 Neutralino's

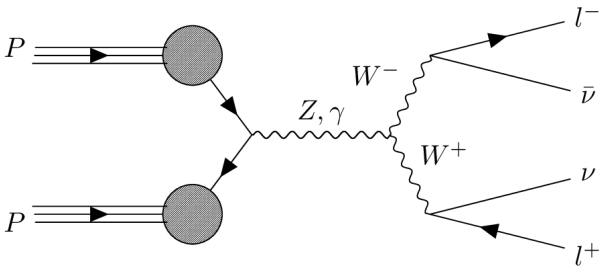


Figure 1.7: The Feynman diagram of the WW-channel.

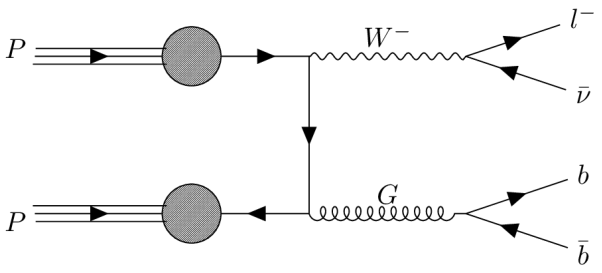


Figure 1.8: The Feynman diagram of the diboson W-jets channel.

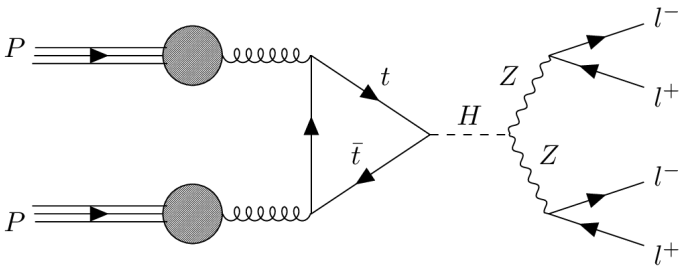


Figure 1.9: The Feynman diagram of the Higgs-channel.

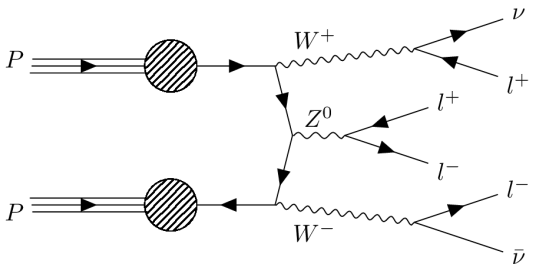


Figure 1.10: The Feynman diagram of the Triboson-channel.

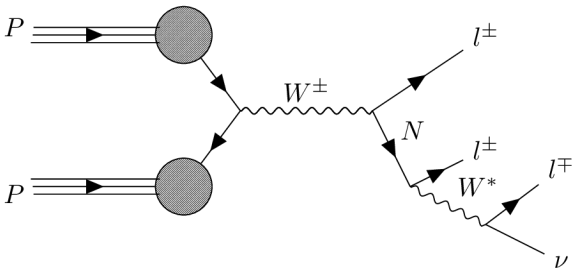


Figure 1.11: The Feynman diagram of the signal-channel.

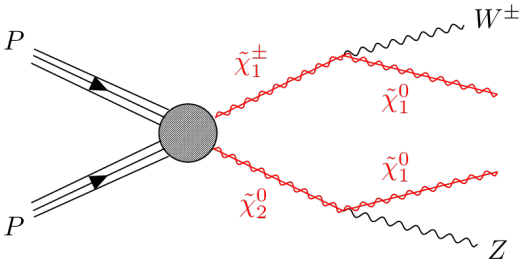


Figure 1.12: The Feynman diagram of the signal-channel.

Chapter 2

Introduction to Machine Learning and Data Analysis

Machine Learning (ML) is rapidly becoming an overwhelming presence 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 [14], separating jets from gluons [15] 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 the use of supervised Deep Neural Networks (DNN). DNN's are famous for their versatility which is partly due to their diversity in architecture and application. In later years Boosted Decision Trees (BDT) have become more and more popular, especially after the release of XGBoost in 2014. The performance of XGBoost matches that of the DNN in many cases, as well as having the advantage of being both stable and easy to use. In this section I will discuss many of the key concepts of ML as well as go into detail of specific models applied in this search.

2.1 Phenomenology

ML models 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 aims to be dynamic and self improving. ML utilizes data 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* are provided along with the data which allows a ML model to learn how to map a set of inputs to a target. In this thesis I will use the notation of $X = [\mathbf{x_0}, \mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_N}]$ to refer to a dataset, $T = [t_0, t_1, t_2, ..., t_N]$ to the corresponding target and $Y = [y_0, y_1, y_2, ..., y_N]$ to the output of the model where N is a number of points. The datasets contain a set of vectors with length equal to the number of features. The target and output could similarly contain vectors, but will in this thesis be restricted to scalar values. Generally, 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 the model can apply any attained knowledge from the study of a set of inputs and targets to predict the target of a new dataset. 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 (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[12]. 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

[12] More on this in later sections.

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 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 at 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 because $-\boldsymbol{\nabla}_\theta(g)$ 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 hyperparameter decided by the user and is called the learning rate, η. 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 η can drastically affect 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 spending computation time beyond reason.

2.2.2 Adam

2.2.3 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

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

2.3 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 Data Handling

2.4.1 Scaling

The range of values for the different data 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* [16]. 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 data set, X as

$$\mathcal{S}(X) = \frac{X - \boldsymbol{\mu}_x}{\boldsymbol{\sigma}_x}, \tag{2.3}$$

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

2.4.2 Principal Component Analysis

Principal Component Analysis (PCA) is a popular dimensionality reduction technique used in many analyses. The goal of PCA is to reduce a high-dimensional dataset while at the same time conserve as much of the variance in the data as possible. The motivation behind dimensionality reduction is rooted in two (main) reasons. The first being noise reduction. Some features could not only be non-contributing during training, but could even introduce noise. The second reason is lack of convergence. In a large data set with many features and different classifications (in our case channels), a ML could struggle to identify the most important trends. By reducing the dimensionality in the data, the hope is that this would be easier.
In simple terms, PCA finds the direction in the feature space along which the data has the largest variance. It does this in 4 steps:

- **First**: Center the data around 0 by subtracting the mean from each feature.
- **Second**: Calculate the covariance matrix to find the covariance of each feature pair.
- **Third**: Calculate the eigenvalue and eigenvectors of the covariance matrix.
- **Fourth**: Order the eigenvector by size of the eigenvalues to define the directions in the feature space with the largest variance.
- **Fifth**: Cast the data along these directions to form a new data set with the new features ranked from largest to lowest variance.
- **Sixth**: Remove the features with the least variance according to some threshold defined by the user.

In figures 2.1 and 2.2 I have plotted the distribution of 10, 100 and 500 samples for the features with the most and second most variance (2.1), and least and second least variance (2.2) after applying PCA to our data set. In this PCA I am yet to remove features, so all four features are taken from a data set where all the variance is still conserved. With this in mind, the different scales on the y- and x-axis (10^0 for figure 2.1 and 10^{-7} for figure 2.2) show why we are able to justify removing the features. In figure 2.2 we observe that all channels exhibit the same trends and are practically identical. In other words, these features would not contribute in training a model to distinguish the different channels.

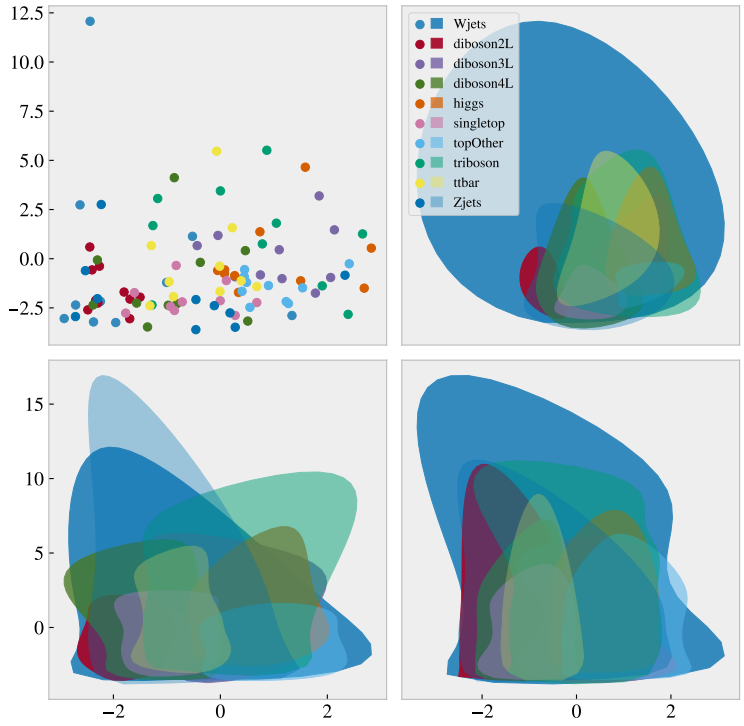


Figure 2.1: The distribution of the two PCA-features containing most variation for (left to right, up to down) 10, 10, 100 and 500 samples from each channel. Each sample filling the requirement with being less than one standard deviation from the mean of both features, respectively.

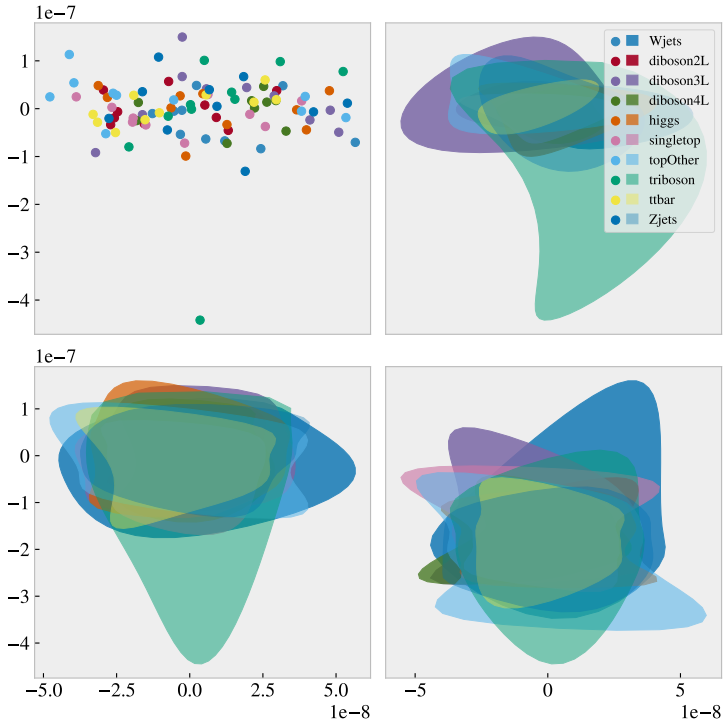


Figure 2.2: The distribution of the two PCA-features containing the least amount of variation for (left to right, up to down) 10, 10, 100 and 500 samples from each channel. Each sample filling the requirement with being less than one standard deviation from the mean of both features, respectively.

2.5 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 too 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.

2.5.1 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.

2.5.2 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.

2.6 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 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.

2.6.1 General structure

There are often drawn comparisons between the structure of the neural network, and the way the human mind operates, 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.

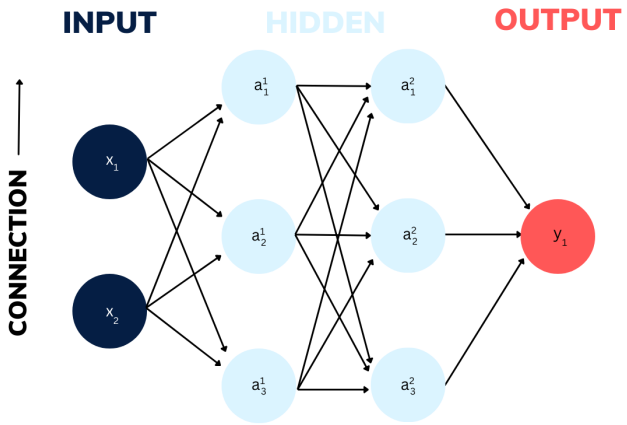


Figure 2.3: An illustration of a NN with two hidden layers.

The neural network functions by passing information in between the different layers through 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 2.3. 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 to a prediction. In figure 2.3 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 one node has on the other.
In a traditional FFNN the information is passed linearly (in figure 2.3, from left to right) in a process we call *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.

2.6.2 Feeding Forward

With the structure described in the previous section, a trained model, $\mathcal{F}$ produces a prediction, Y for a data set, X 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 $l \in \{2, ..., L\}$[13] 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 2.3), 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 activated value of 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, σ^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

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

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

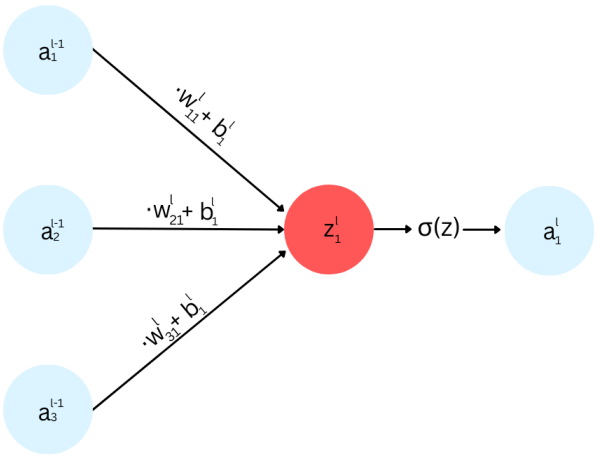


Figure 2.4: An illustration information pass for one layer to another.

A more detailed illustration of the information pass from one layer to a node in the next is displayed in figure 2.4. In figure 2.4, we see all steps described in the process; *(1)* all nodes in l-1 are summed with a corresponding weight, *(2)* the sum is shifted 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 σ^l. This method is used for information pass between all layers, except 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 of 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.

2.6.3 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. When a neural network produces a prediction, Y the error in the prediction is defined by the cost function as $\mathcal{C}(Y, T)$, where T is the target. To minimize $\mathcal{C}$, the backward 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 δ_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 the expression for δ_k^l

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

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 δ_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.

2.6.4 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*

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

where α is scalar.

,

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

2.6.5 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.

Dropout

The dropout-layer functions by assigning a predetermined probability to each neuron in a given layer. Based on the probability, a number of neurons (dependent on the probability) is dropped in each forward pass. 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.
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-adaptation. 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 (see subsection 2.5.2).

Channel-Out

Dropout-layers are not the only layers to dynamically change the architecture of a network. In this thesis I additionally explored the lesser known, *Channel-Out*-layer as found in the paper by Srivastava et al. [17]. Channel-out, similarly to dropout creates an ensemble of networks by removing a set of nodes for each round of training. Contrary to dropout, channel-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 in each respective unit. The goal of channel-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 channel-out layer creates something resembling a stacking ensemble (see subsection 2.5.2).

Stochastic-Channel-Out

As I will describe in further detail in sections to come, I choose to implement channel-out layer myself. A consequence of this was that I was able to experiment. In section 2.6.5, I described the issue of complex co-adaptation. I also described a possible solution to the issue, the dropout layer. Although dropout and channel-out function rather similarly, they do not deal with this issue similarly. Where the dropout layer will force dormant nodes to contribute in the training by allowing for dominant nodes to be dropped, the channel-out would instead allow for dominant nodes to take further control inside their respective unit.
To remedy this issue, I proposed a new layer, the Stochastic-Channel-Out (SCO). SCO functions similarly to channel-out, with the exeption of the units. Where the channel-out utilizes static local units[14] throughout training, the SCO instead utilizes dynamic units which changes for each data point. In other words, for each data point, each node would potentially have a new group of nodes to compare with when finding the larges activation. The goal of the SCO is to capitalize on the randomness of the dropout layer, and the trend specific paths' aspect from the channel-out layer.
In the same manner as for channel-out, the redimensionalization is active during prediction and training a like for the SCO. This means that also the SCO creates an ensemble resembling a bagging ensemble.

Maxout

Additionally, to the channel-out I will be applying a second layer proposed by the paper by Srivastava et al. [17], the *Maxout* layer. The maxout layer functions very similarly to the channel-out with a small difference

[14] I.e. the nodes are placed in local units in the beginning of training and held constant throughout.

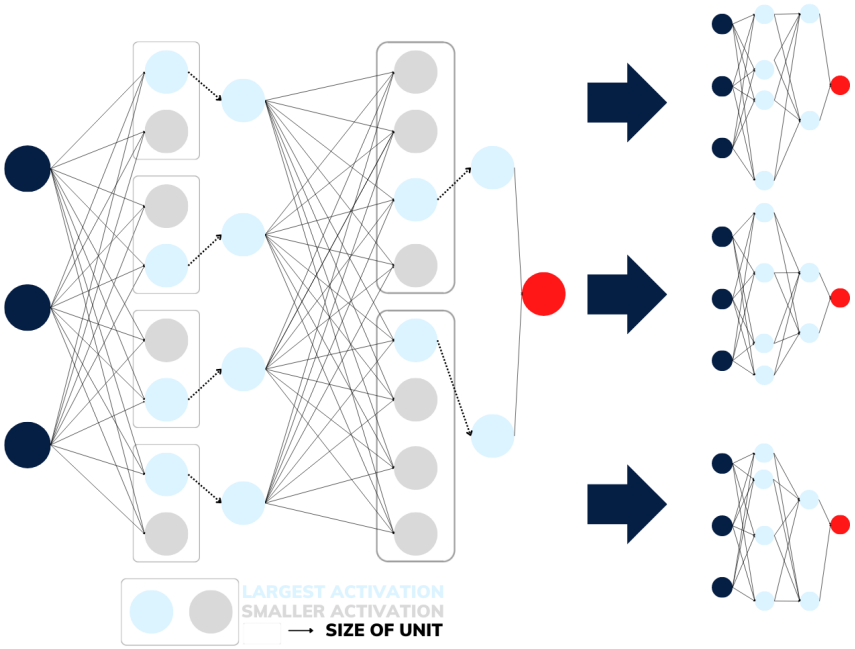


Figure 2.5: An illustration of a Neural network with two hidden layers, each with 8 neurons. The first hidden layer has a maxout activation layer with 4 units, the second has 2. The figure also illustrates the resulting ensemble of smaller neural networks as a consequence of the maxout activation layers.

in the dropping of nodes. In the dropout layer, nodes are dropped by neglecting all contributions from said node, or in other words by multiplying the activation by all dropped nodes by zero. This is replicated in both the channel-out layer and the SCO layer. In doing so, all nodes, activated and dropped alike, possesses their own set of weights and biases. This is not the case for the maxout layer. In the maxout layer, each unit possesses just one set of weights and biases. For each forward pass in the network, the largest activated node will contribute to the nodes in the following layer using the same weights and biases as the rest of the nodes in the units. However, not that the nodes only share parameters connected to the layer in front, not behind. This allows for the maxout layer to have fewer parameters to tune, while at the same time building trend specific pathways similar to channel-out and SCO.

In figure 2.5 I have illustrated a maxout network. The figure shows a NN with two hidden layers with 8 nodes each. In the first hidden layer the maxout layer creates 4 units, each containing two nodes and in the second layer the maxout creates two units 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 2.5, I have illustrated how the different configuration of paths through the network creates an ensemble of networks with their own architecture. Note in the figure how, as described in the previous section, each node possesses its own connections to the previous layer, while sharing the connections with the rest of the unit, to the next. In figure 2.6 I have drawn a separate illustration to show how this differs from both channel-out and SCO in this regard. Figure 2.6 illustrates how all three layers redimensionalize the network, how channelout and SCO differ from maxout in regard to the relationship between units and parameters and finally how channel-out differs from SCO in choice of units[15]

2.6.6 Parametrized Neural Network

In this thesis I will be studying Beyond Standard Model (BSM)-simulations with a diversity in choice of free parameters. This means that my signal is in it of itself diverse. The free parameters studied in this thesis are the mass of new particles. The mass of a particle in a collision can greatly alter the feature space spanned by a channel. This means that a ML-model could potentially struggle to tune according to all trends in the signal. The obvious solution to this problem is simply to implement one model per mass combination, or choice of mass and simply reusing the background. Although this approach is very popular, I have decided not to use it for (mainly) three reasons:

- This approach has been carefully studied for many years, and leaves little room to further explore.

[15] In figure 2.6 this is highlighted by the different colors surrounding the nodes in SCO, where each color represents a unit.

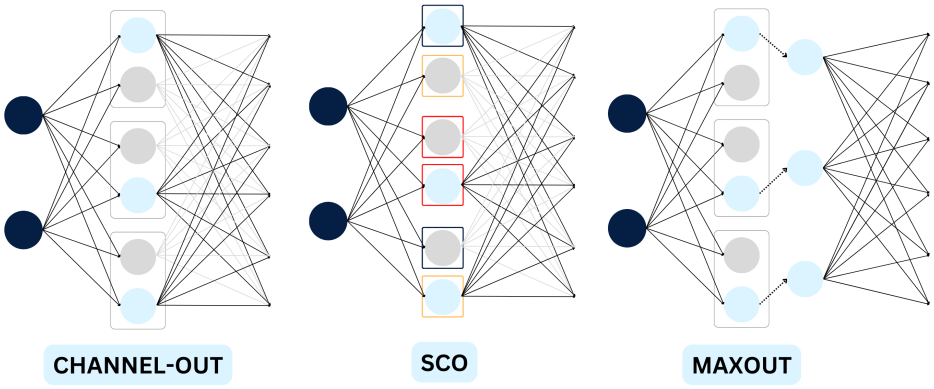


Figure 2.6: An illustration of three different layers, channelout, SCO and maxout. The figure shows how each layer redimensionalize the network and how channelout and SCO differ from maxout in regard to the relationship between units and parameters.

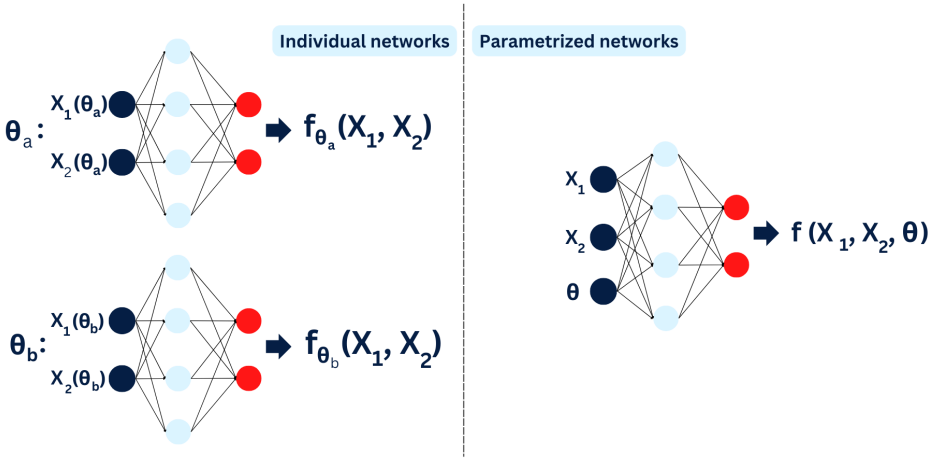


Figure 2.7: An illustration of a comparison between the parameter individualistic network approach and the PNN.

- Some signals overlap in the feature space. This means that by neglecting signal with relative similar mass, you could be neglecting statistics which could help tune to your original signal.
- By including two signals with relatively similar masses, some[16] suggest that the model would be able to interpolate between the masses and cover a larger search area.

When the data set is dependent on a set of free parameters, $\theta = [\theta_0, \theta_1, \theta_2, ..., \theta_N]$, we can write $X(\theta)$. For a single neural network trained on a set of parameters, we define the model as $\mathcal{F}(X(\theta))$. In the case we build one model for each choice of parameter, we get a set of models defined as $\{\mathcal{F}_j(X(\theta_j))$, $\mathcal{F}_k(X(\theta_k))$, ...,$\mathcal{F}_l(X(\theta_l))\}$, where j, k and l are indices which correspond to individual choices of parameters respectively. In the paper by Baldi et al. [18], they propose a separate solution to the problem than the one proposed above, the Parametrized Neural Network (PNN). By simply including the parameters as additional features, we can write $\mathcal{F}(X, \theta)$. In doing so, we can utilize all the statistics at the same time, while also aiding the NN in its effort to recognize all individual trends for all signal. In practice, it is doing this by adding a shift to the total output from the initial layer according to the discrete distribution of the parameters. The hope is that each discrete shift will motivate a separate individualistic tuning for each choice of parameter. For the SM background, adding a parameter representing the mass of a BSM particle is meaningless. Therefore, the background is randomly assigned values according to the same distribution used in the signal data. In doing so, one is essentially assigning each set of signal its own portion of background data.

[16] See the paper by Baldi et al. [18].

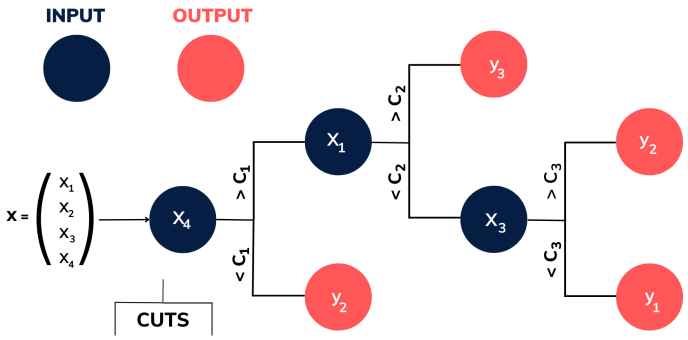


Figure 2.8: An illustration of a simple DT, mapping a 4 dimensional input to one of three values in the target space.

2.7 Decision Trees and Gradient Boosting

2.7.1 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 solving a large range of complex problems. In this section I will cover the use of DT as applied to a supervised classification problem.
The goal of decision trees are to create a flowchart-like tree structure from input to output. Similarly to the traditional cut-and-count method used in particle physics, a DT places *rectangular-cuts* on a data set. It does so to create a collection of thresholds $\mathcal{C} = \{c_1, c_2, ..., c_{N_c}\}$, which when applied to a new data set and target, $\tilde{X}$ and $\tilde{T}$ will sort $\tilde{\mathbf{x}}_i$ to a $\tilde{y}_i$.
In figure 2.8, I have illustrated a simple DT classifying a 4-dimensional input data to one of three classifications. As is visualized in figure 2.8, the DT applies a set of thresholds on the data to find the route applicable to a target. For each applied cut, the data split into what we call *branches*. Each branch represents a subset of the data. The final subset, after a sufficient amount of cuts ends in what we call *leaves*. The leaves are the label we assign the subset of data contained in this branch.
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.

2.7.2 Gradient Boosting in Decision Trees

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 collection of m number of weak classifiers, estimators. A prediction for the model on a given data-points, x_i is defined as $\mathcal{F}_m(\mathbf{x}_i)$, 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, $\mathcal{C}(\mathcal{F}_m(\mathbf{x}_i), y_i)$. In other words we define the new estimator as:

$$\tilde{\mathcal{C}}(\mathcal{F}_m(\mathbf{x}_i), \mathbf{y}_i) = h_m(\mathbf{x}_i), \tag{2.11}$$

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

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

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

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

The XGBoost [19] 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. For more detailed explanation on the XGBoost framework, the reader is refered to Ref.[20]

2.8 Machine Learning Applied to a BSM Search

So far I have presented the goal of my analysis (to discover new physics) as well as the tools (ML) I will be using to achieve it. In this section I will discuss how ML is applied to the problem as well as how it compares to traditional methods.

2.8.1 The Traditional Approach

As discussed, I have been presented with two sets of data; the measured collision data from ATLAS and the simulated Monte Carlo (MC) data. The origin of the latter data set will not be covered in great detail in this thesis, but it is worth mentioning that the simulations are based on SM theory. In other words, by comparing the measured collision data with the SM simulations, we are essentially comparing what is predicted by the SM to what is measured by experiment. If the two differ in ways not explained by simulation error or other non physics related errors, one could interpret the deviation as a new physics.
To summarize, a search for new physics, is a search for deviation between simulation and experiment. At first thought, this might seem like a simple task. And if said deviation was large, it would be easy. In reality, any new physics which predicts large contributions in the data currently measured by ATLAS has been excluded a long time ago. Today, any promising extension of the SM predicts to contribute rarely in any detector. As will be presented in later section, some theories that will be searched for in this thesis, only contribute to a total of 6 events in a data set consisting of more than 3800000 events ($< 0.002\%$). Not only would such a deviation be incredibly hard to detect, but it would be close to impossible to determine if such a deviation is rooted in new physics or simply noise.
The traditional approach to this problem, is to study the data in physics motivated regions. For example, in section 1.6 I presented the Feynman diagrams of the signals I searched for in this thesis. As mentioned, this type of final state is expected to exhibit large amounts of missing transverse energy[17]. The traditional approach is to neglect all the data with small amounts of missing energy, and only consider events of interest. By applying these kinds of constraints on the data, you are creating a region where you expect to find as much of the signal as possible and as little of the background. After applying a sufficient amount of demands (or cuts), you would count the remaining data in your *search region* and check for deviation. This approach is called Cut-and-Count (CC).

2.8.2 The Machine Learning Approach

For a signal which greatly differs from the SM, the CC method could be sufficient. But, in cases where the signal is similar to background, it becomes harder to create effective[18] cuts. In figure 2.9, I have drawn an illustration of two different feature distributions both displaying the distribution for a hypothetical signal and background. Additionally, I have drawn a threshold in both distributions which represents the cut made to create a search region. In the first figure (left) the distributions of signal and background are relatively

[17] Due to the neutralions being both heavy and neutral.
[18] By effective cuts I mean cuts which removes large amounts of the background while at the same time preserves as much of the signal as possible.

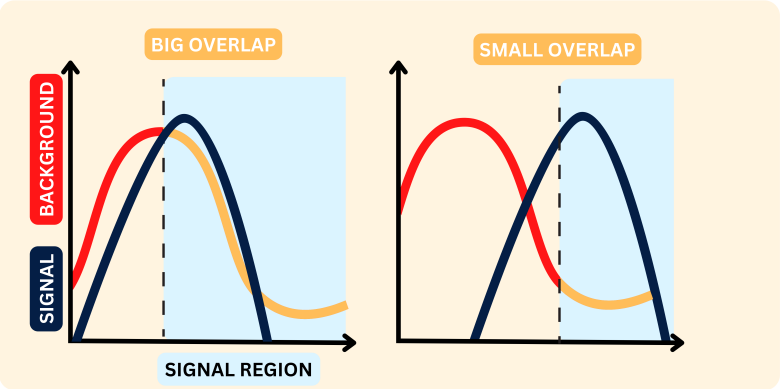


Figure 2.9: An illustration of a traditional cut-and-count approach as well as the motivation for creating a ML variable.

similar, and the signal region contains large amounts of background. In the second (right), the distributions of signal and background are greatly different, and this variable would allow for a much more effective signal region.
The goal of introducing ML, is to create a new feature in the data set where background and signal exhibit as little overlap as possible. This is very similar to how we introduce physics motivated high-level features like $H_t(ll)$ or $M_l ll$, but instead of using an analytical function grounded in physics, we apply the output of a trained ML model. Then, once the ML-variable is created, we apply a cut (similar to CC) which will define an effective search region.
How we create the ML-variable can vary depending on the type of ML. In the case of unsupervised ML, one aims to perform a type anomaly detection. One unsupervised approach is to overfit on the background, then hope that said overfitting is reflected when a new process is introduced. An effective unsupervised approach would be incredibly beneficial, as it would be totally model independent. In other words, the same model could be applied to all signal. As mentioned in earlier sections (see section 2.1), the focus of this thesis will not be on unsupervised ML, but supervised. The largest difference between the two is the introduction of an additional data set, the signal. By simulating and training on what we expect the new physics to look like, we are able to achieve a much more effective output, but the output would tailered to the signal it's been exposed to and not much else[19].

2.8.3 Analogy

2.9 Model assessment

2.9.1 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. In figure 2.10 I have plotted an illustration of a ROC curve. The curve is plotted on an x-, y-axis where the x-axis represents false-positive rates and the y-axis represents true-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.

[19] Except for overlapping signals or models which are able to interpolate between regions.

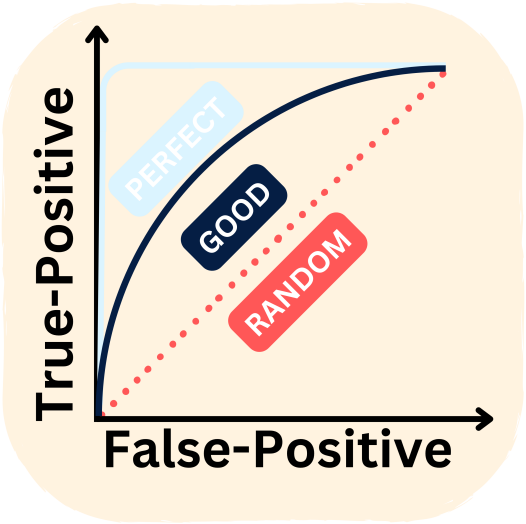


Figure 2.10: An illustration of a ROC curve.

2.9.2 Sensitivity

AUC is a very good measure of how separated the distribution for signal and background are for a feature. In the search for new physics, features with large separation between signal and background are incredibly helpful, but not the end goal. The final result in the search for new physics is the discovery of a significant deviation between the SM and experiment. We measure the deviation between the measured collision data and the simulated SM data in units of significance, Z.
For a given signal region with n_{obs} events of measured collision data and b simulated SM data, we define the significance of a deviation as

$$Z = \sqrt{2\left[n_{\text{obs}}\ \ln\frac{n_{\text{obs}}}{b} - n_{\text{obs}} + b\right]} \text{ or } Z = \sqrt{2\left[(s+b)\ln\left(1+\frac{s}{b}\right) - s\right]}, \tag{2.14}$$

where we have defined the signal, s as $s = n_{obs} - b$. In the case of large statistics ($b >> s$), we can write Z as

$$Z = \frac{n_{obs} - b}{\sqrt{b}} = \frac{s}{\sqrt{b}}. \tag{2.15}$$

By studying equation 2.15 we observe that the significance resembels the ration for signal and background. The larger the Z, the more significant the deviation. In physics, one often deems the results a discovery if Z is larger than 5.
In the case we are not intrested in discovering physics, but simple to measure the sensitivity of a model, we can also use significance. In this case we do not have n_{obs}. Instead we simply define s as the number of events for the simulated signal inside the signal region. This number would be spesific for each mass combination and will be a measure for how sensitive the model is for that exact combination.

Chapter 3

Implementation of the Analysis

3.1 Tools and Data

Every year technology for generating and measuring particle collisions is improving. 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 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 MC-sumulations and real collision data.

3.1.1 Monte Carlo Data

3.1.2 ROOT, RDataframe and Pandas

ROOT [21] 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, ordering of histograms and statistical analysis tools.
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 [22]. 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 the stransverse mass (M_{T2}), invariant mass of 3 leptons (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 [23]. 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 the full ML pipeline (data-handling, training, validating ets.) is completed, the data is transformed back to RDataFrame, to take advantage of the 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 in the 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 transformed 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 functions that are used in the analysis and are not already native to ROOT. In the case of computing the trilepton invariant mass, the $C++$-function is created like shown in listing 3.1.
In listing 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 $TLorentzVector$ [24] 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, η, ϕ and M. Then, through the $TLorentzVector$ class we can simply add all three vector together, and extract the invariant mass of the 3 leptons.

```
// 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 listing 3.2, 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 helperfunctions 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 select '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.

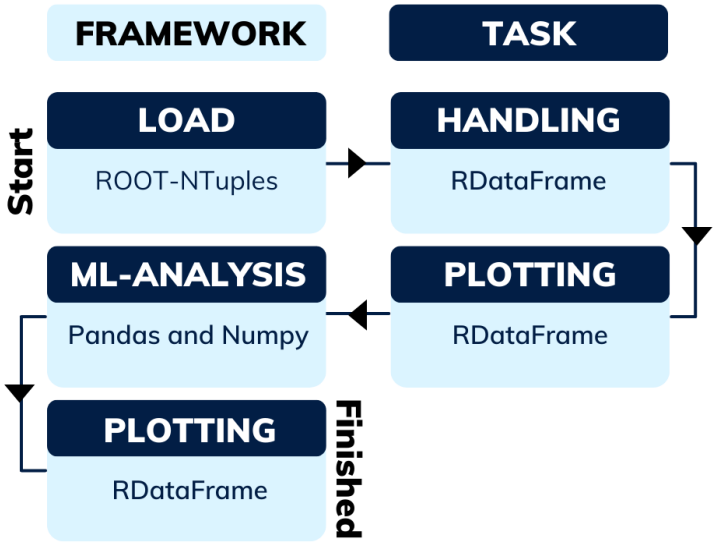


Figure 3.1: A visual summary of the workflow and framework use for the computational analysis.

3.1.4 The Signal

As I have mentioned in previous sections I will be searching for a large range of signals, all related to the same physics, but with different set of parameters. In figure 3.2 I have drawn a grid showing the different mass combinations searched for in this analysis. The figure shows the data set containing masses ranging from $M_{\tilde{\chi}_1} \in [0, 400]$ and $M_{\tilde{\chi}_2} \in [100, 800]$. In the beginning of my analysis I only received a subset of the full signal set. In the figure I have marked the original data set[20] with a white label in the top right corner of the box.
Inn the first data set I received (which will be referred to as the *original* data set.) there are a total of 30 different mass combinations in the range from $M_{\tilde{\chi}_1} \in [0, 400]$ and $M_{\tilde{\chi}_2} \in [400, 800]$.

3.2 Features

The choice of which features to study and which to neglect is 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 researchers can exploit when creating ours signal search 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 missing transverse energy, 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[21]. Therefore, preselection 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 requirements were taken from the ATLAS article from 2022 [25]. Given the

[20] I.e. the first data set I received.
[21] By including large amounts of irrelevant data, you risk the ML-model tuning unnecessarily to remove easily reducible background, which could compromise performance on irreducible background.

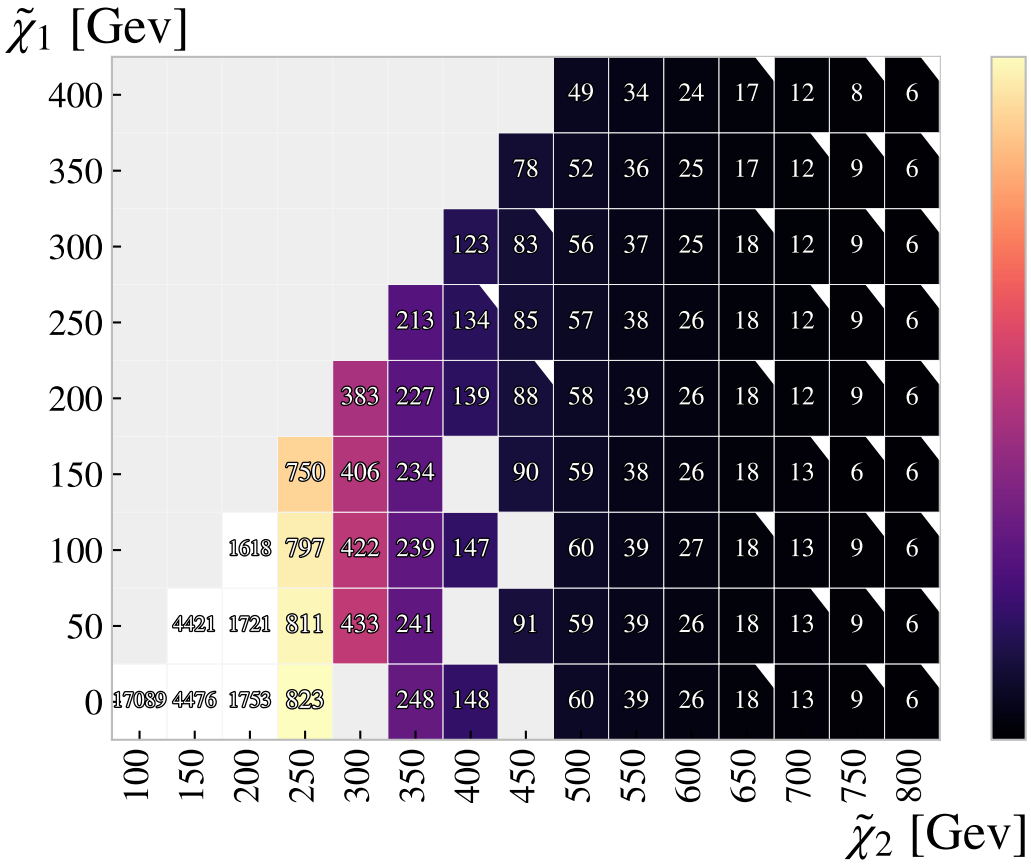


Figure 3.2: A depiction of all mass combinations in the full signal data set.

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.

definitions we demand that each event contains exactly three signal (3.2) and three baseline leptons (3.1), 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 [26]. 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 from 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 isolation criteria for a cone around the lepton is used to suppress QCD-background events and reduce fake leptons. 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 transverse momentum, P_t. The latter case can often be caused by poor reconstruction or miss identification. These are issues we aim to solve by checking different triggers. Given our data set is composed of different data sets spread over several years, different triggers are used.

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.

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.

3.2.2 Lepton Variables

Now we will have a look at lepton variables 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 η and the azimuth angle ϕ. 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 lepton was calculated and included along with the energy E_t^{miss} and azimuth angle ϕ^{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 specific searches such as this, one can allow one self to add physics motivated higher-level features. The higher level features used in this thesis were inspired by [25] (ATLAS 2022).
Firstly I added different mass variables, namely m_{lll} and $m_{ll}(OSSF)$ (Opposite Sign Same Flavour (OSSF)). The first being the trilepton invariant mass and the latter being the dilepton invariant mass of the pair with 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 momenta. 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

Now we can have a look at the jet-features. Given the final-state of interest should be independent of jets, there are not many features added with jet information. But, given the risk of miss identification 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 efficiency of a multivariate analysis used to separate jet-flavors. The efficiencies used for b-tagging 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, m_{jj}.

3.2.4 Validation

As mentioned in previous sections, the comparison between SM MC and real data is a crucial part of the analysis, and we must therefore insure an adequate reconstruction of the real data before we begin the analysis. This is not only true for the low-level features, but for all features used in the analysis to create a new ML variable. Therefore, we will in this section compare both sets of data for all features included in the analysis.
In the figures displayed from 3.3 to 3.8, I have drawn the event distribution for all the features discussed in section 3.2.2 and 3.2.3. In table 3.4 and 3.5 I have created an overview of the figures displayed for lepton specific and event specific features respectively. The error bars in each bin are defined by default set to ($\#events\ per\ bin$). Under each figure I plot the ratio of the observed data divided by the MC for each bin. By studying the ratio-plots for each figure we observe that the ratio for all bins for all features are between 1.2 and 0.8. Most bins even lying closer to 1. Simply put we observe that the MC seems to adequately

Feature	l_1	l_2	l_3
p_t	3.3*a*	3.3*b*	3.3*c*
η	3.3*d*	3.3*e*	3.3*f*
ϕ	3.4*a*	3.4*b*	3.4*c*
M_T	3.4*d*	3.4*e*	3.4*f*
$Charge$	3.5*a*	3.5*b*	3.5*c*
$Flavor$	3.5*d*	3.5*e*	3.5*f*

Table 3.4: Refrences to figures for all lepton spesific feature distributions.

Feature	$Refrence$
E_T^{miss}	3.6*a*
$\phi(miss)$	3.6*b*
M_{lll}	3.6*c*
$M_{ll}(OSSF)$	3.6*d*
Sig E_T^{miss}	3.5*a*
$H_t(lll)$	3.6*f*
$H_t(SS)$	3.7*a*
$H_t(lll) + E_T^{miss}$	3.7*b*
ΔR	3.7*c*
Flavor combo	3.7*d*
Nr of signal Jets	3.7*e*
M_{jj}	3.7*f*
Nr of B-jets(77)	3.8*a*
Nr of B-jets(85)	3.8*b*

Table 3.5: Refrences to figures for all event spesific feature distribution.

imitate the trends of the observed data for all features used in the analysis.
Apart from the excellent agreement between observed and simulated data, we can note a couple of other expected but interesting trends. The first being the size of each channel. $Z-jets$ is by far the largest channel followed by the $Diboson(lll)$. Although $Z-jets$ is the largest channel, by comparing the Feynman-diagrams of each channel (section 1.5), $Diboson(lll)$ should be the hardest background to separate due to the similarities in the final-states of the signal. The second point of interest, are the Z-mass (91GeV/c) peaks for some mass related features (3.6c, 3.6d). This just reiterates the fact of the large contributions made by the boson-dominant channels in the data set.

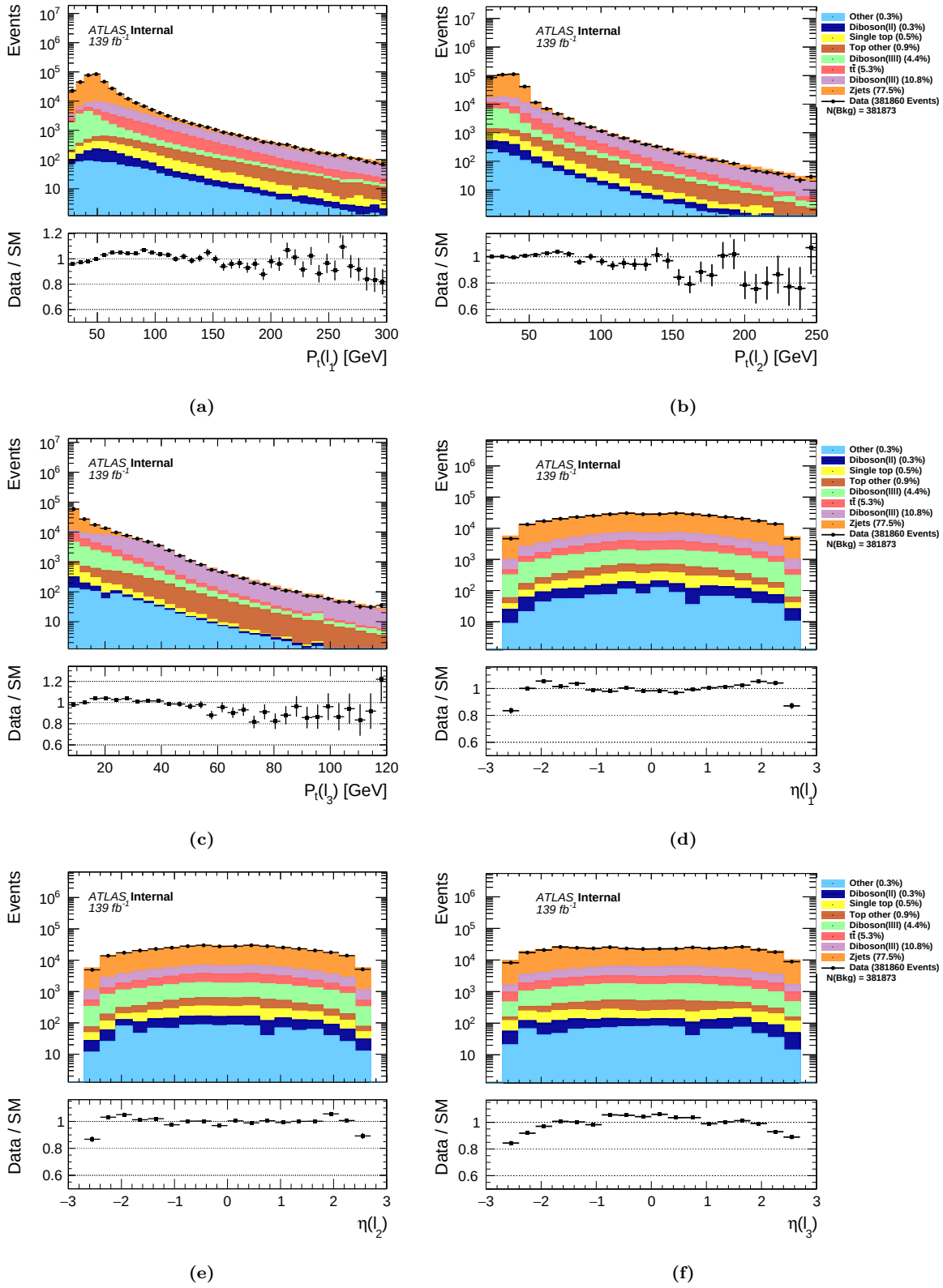


Figure 3.3: MC and real data comparison and event distribution for each channel over P_t for the first 3.3a, second 3.3b and third 3.3c lepton. Similarly, the distribution over η for the 3.3d, second 3.3e and third 3.3f lepton.

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 and training a NN, negative weights are not a problem. But, when using the weights in the training of the XGBoost classifier, the algorithm does not work. EXPLAIN WHY.
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 3.9a I plotted the absolute value P_t distribution of events with negative weights and compare it to all the events in figure 3.9b. Similarly, I have plotted the same values for the second lepton in figures 3.9b and 3.9d 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 3.10a, 3.10b, 3.10c and 3.10d respectively but for ϕ. The same observations are made for both P_t and ϕ. This means we can be justified in treating the negative weight distribution uniformly 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 take the absolute value [22] and normalize all the weights, 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 Machine Learning Models

3.3.1 Model selection

In this analysis I have chosen to compare 4 different ML-models, dens-Neural Network (NN), Parametrized Neural Network (PNN), Local-Winner-Takes-All (LWTA) and XGBoost. The first three methods are all types of NN. I have deliberately chosen to focus on NN given that there is far more freedom in the design of the architecture of a NN, than compared to a XGboost. Additionally, this was motivated by the selfish reason being that I found the networks more interesting to study and dissect. Therefore, most of the analysis, comparisons and discussion is focused on the networks, while XGBoost is included as a loose benchmark.
The choice of the three network architectures is motivated in wanting to compare a simple deep network, a network ensemble and a PNN. I would assume that the optimal architecture would be a network which combines elements from each, but for the purpose of discussion and research I have chosen to keep them somewhat separate.

3.3.2 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[23], 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 (see subsection 2.6.5) does, but with minor differences. Additionally, I wanted the freedom to experiment with the implementation of the layer. The implementation of both max-out and channel-out (see subsection **??**) was done by creating a custom activation function which is called inside a dense-layer.
In the code listing, 3 I have included the code implementation of the activation function used to create

[22] Thereby removing all negative weights.
[23] The developers of TensorFlow

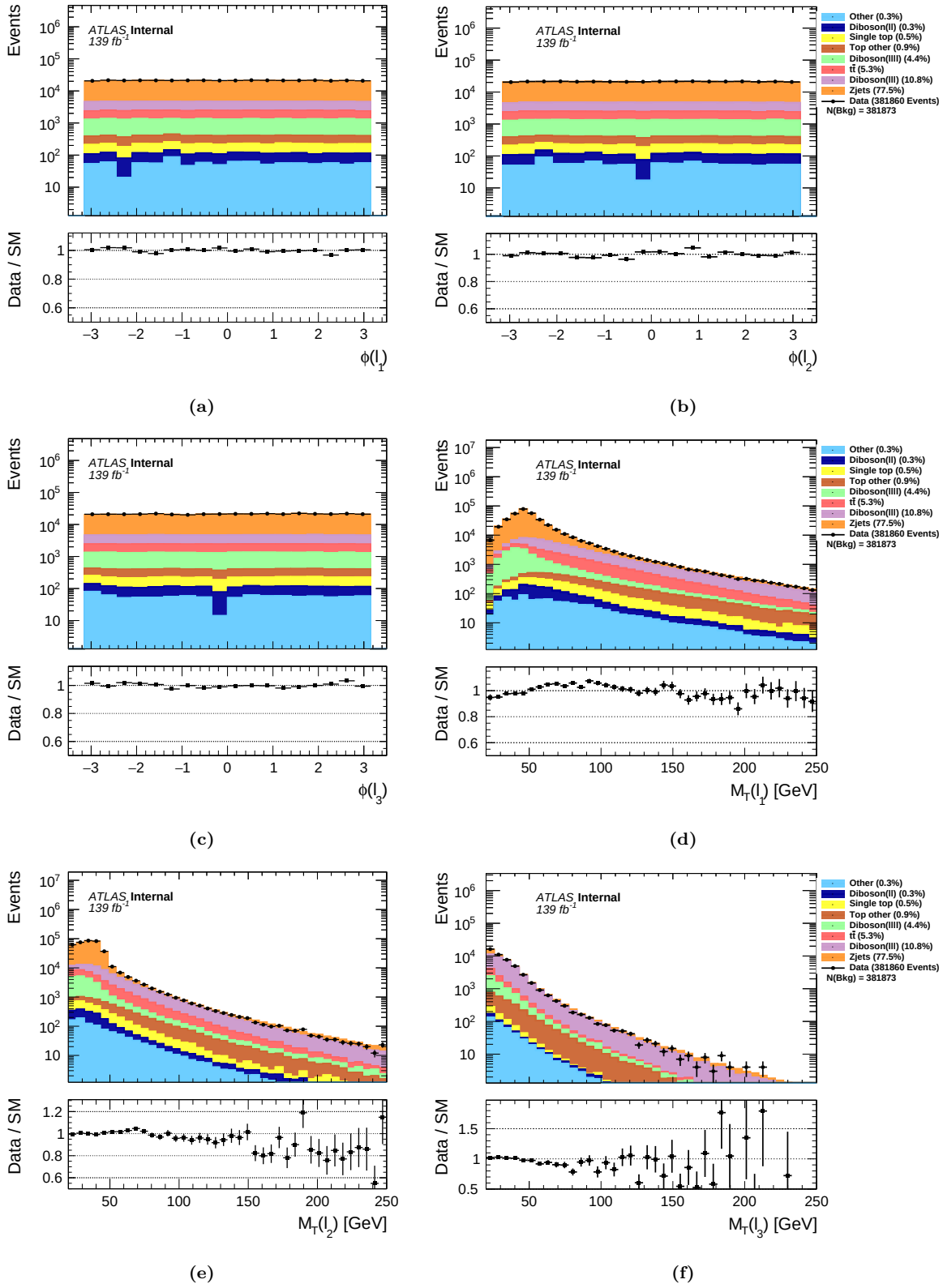
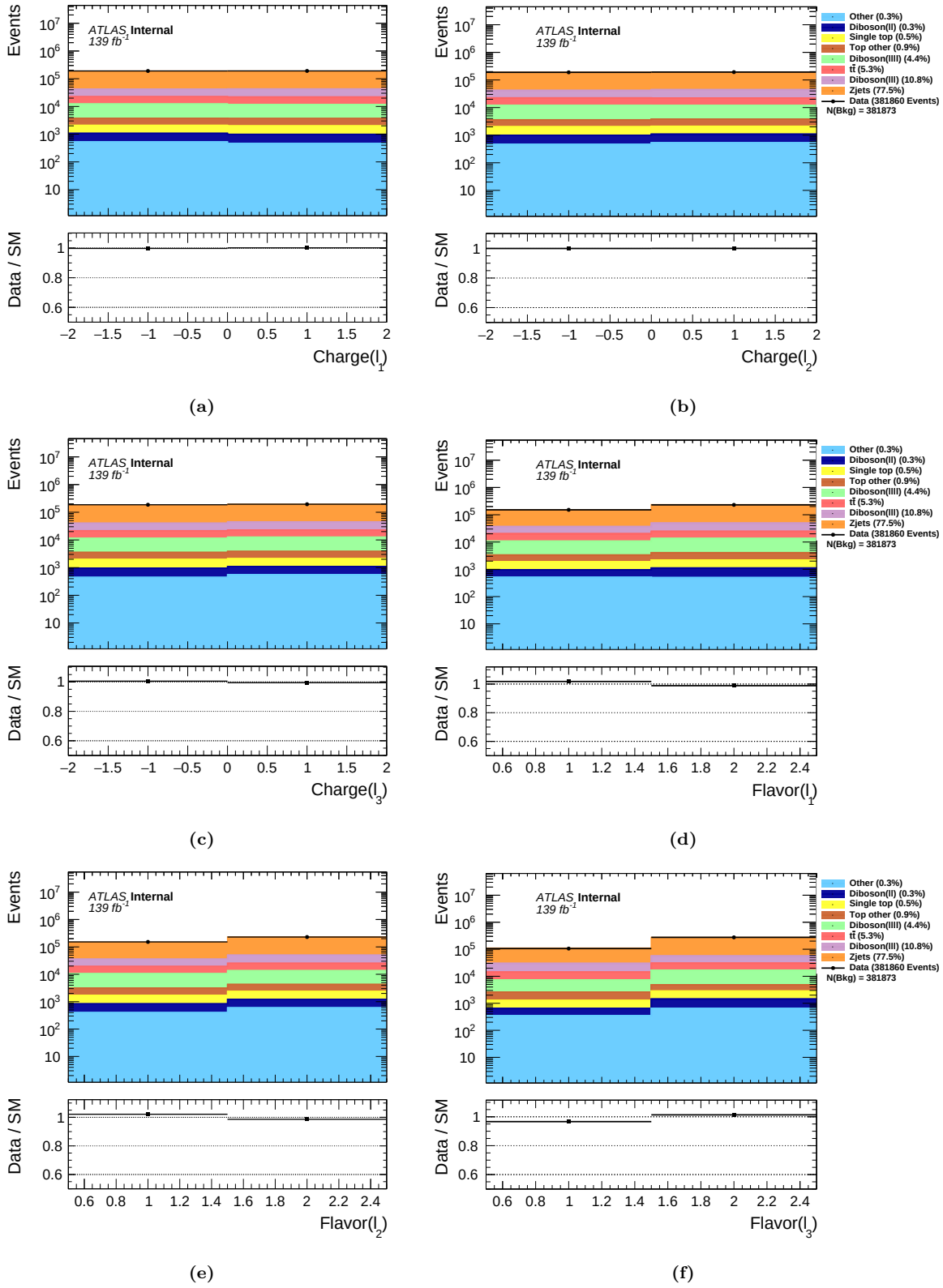


Figure 3.4: MC and real data comparison and event distribution for each channel over ϕ for the first 3.4a, second 3.4b and third 3.4c lepton. Similarly, the distribution over m_t for the first 3.4d, second 3.4e and third 3.4f lepton.



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

Figure 3.5: MC and real data comparison and event distribution for each channel over the charge for the first 3.5a , second 3.5b and third 3.5c lepton. Similarly, the distribution over the flavor for the first 3.5d, second 3.5e and third 3.5f lepton

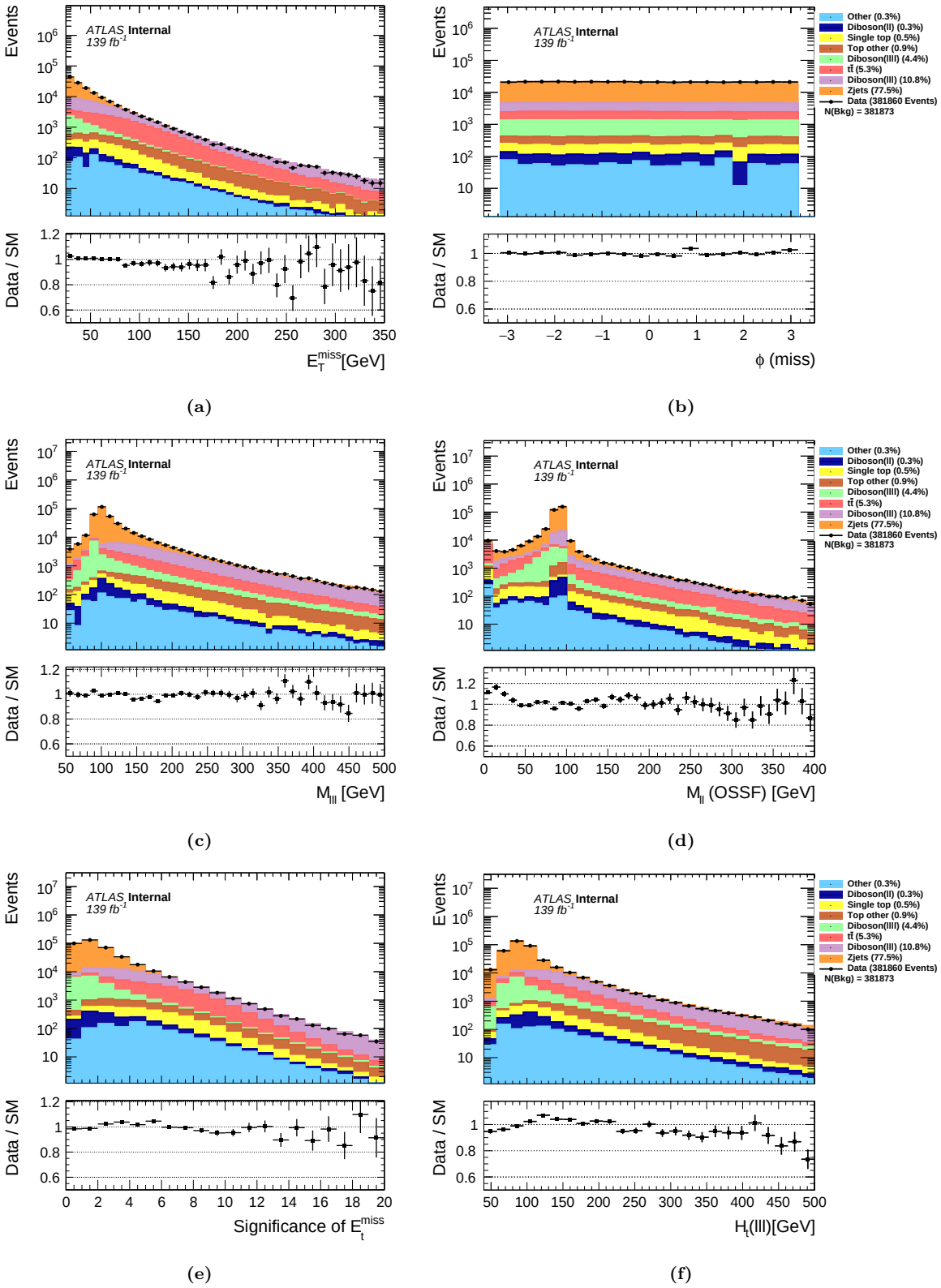


Figure 3.6: MC and real data comparison and event distribution for for each channel over the energy 3.6a and azimuthal angel 3.6b for the transverse momentum. The distribution of the invariant mass of the three leptons 3.6c and the OSSF pair 3.6d. The distribution over the significance of the missing transverse enegy 3.6e and the sum of P_t 3.6f.

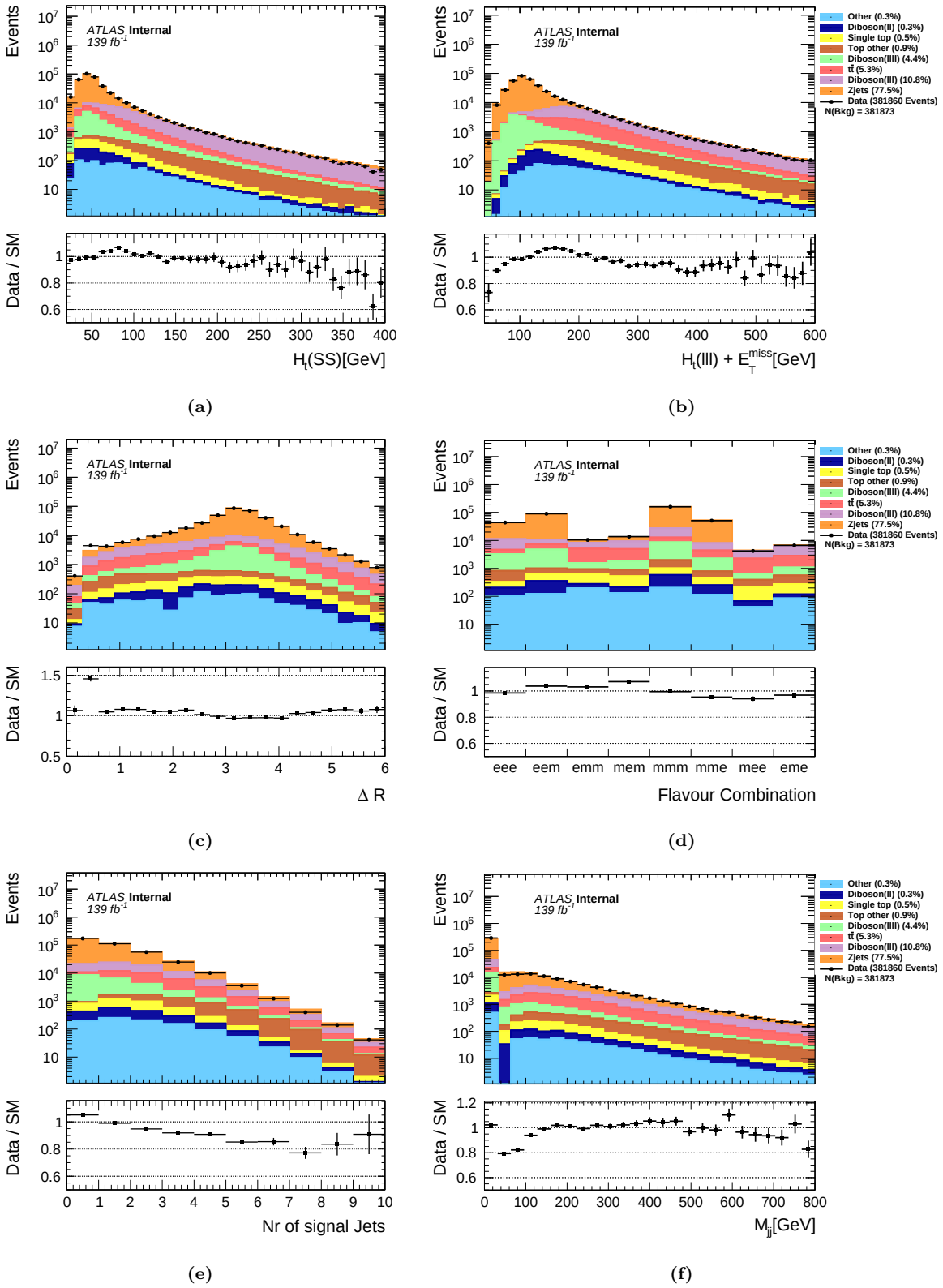
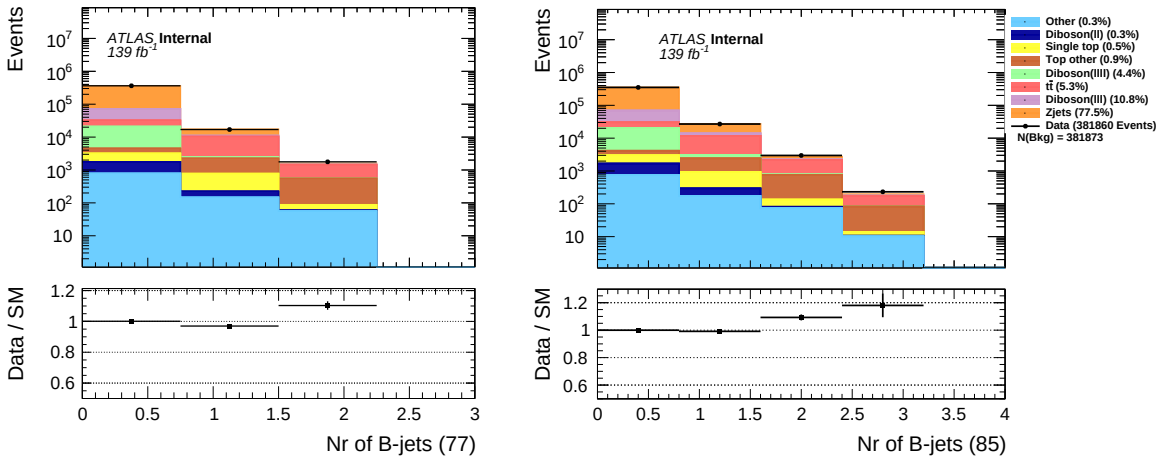
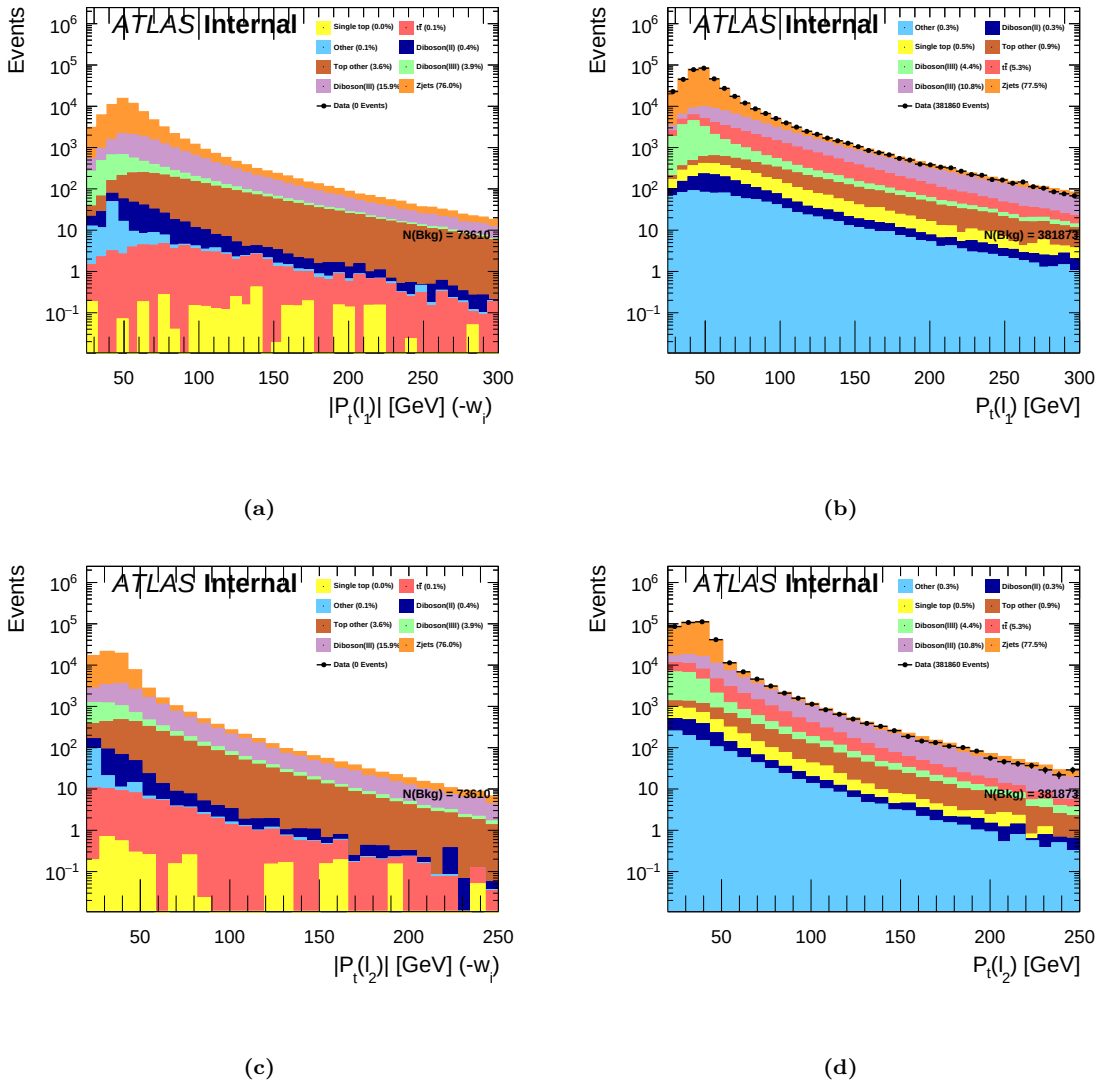


Figure 3.7: MC and real data comparison and event distribution for each channel over the sum of P_t for the SS pair 3.7a and the sum over all three leptons added with E_t^{miss} 3.7b. The distribution over ΔR 3.7c and the flavor combination of the three leptons 3.7d. The distribution of number of jets 3.7e and the mass of the leading di-jet pair 3.7f.



(a) (b)

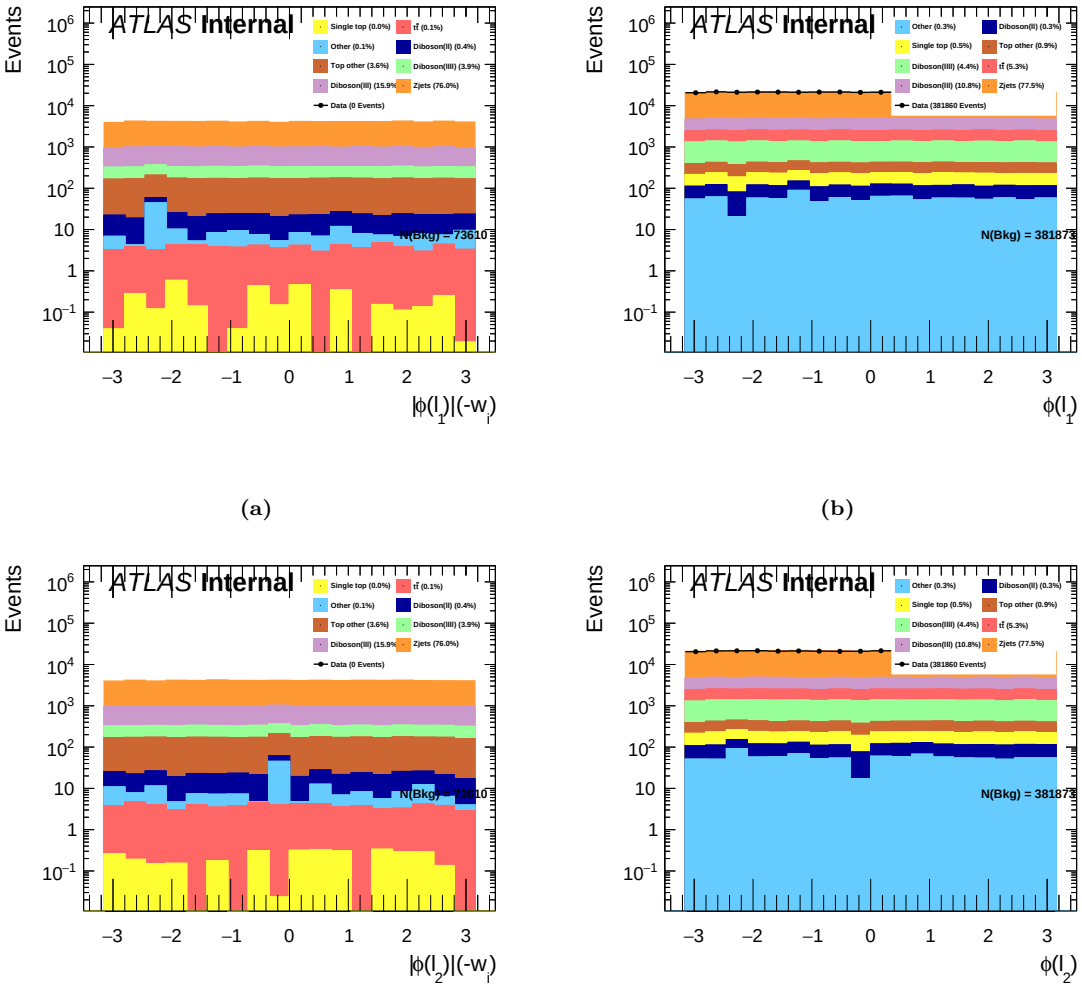
Figure 3.8: MC and real data comparison and event distribution for each channel over the number of b-jets with 77% 3.8a and 85% 3.8b efficiency.



(a) (b)

(c) (d)

Figure 3.9: The absolute value of the P_t 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).



(a) **(b)**

(c) **(d)**

Figure 3.10: The absolute value of the ϕ for events with strictly negative weights (l_1 3.10a, l_2 3.10c) and all events (l_1 3.10b, l_2 3.10d).

the max-out layer. The listing shows how the function takes the input from the previous layer, defines the new shape of what will become the output, groups the nodes in the size defined by number of units, then returns an output which includes only the largest activated node from each unit. By using this function in a dense-layer, said layer will act as a max-out layer.

Algorithm 1 The pseudo code for implementing the Maxout layer in TensorFlow

```
def MaxOut(Input):
    % Pass input through weight kernel and adding bias terms
    Input ← Input × Weights
    Input ← Input + Bias

    % Reshape input into units
    Input ← Reshape(Input, (Nr Units, Size of Units))

    % Reduce input to the largest activation in each unit
    Output ← Max(Input)

    % Reshape to size equal the number of units.
    Output ← Reshape(Input, (Nr Units))
    return Input
```

Channel-out

For the activation function defined to create the channel-out layer, I again grouped the nodes similarly as I did for max-out. Instead of returning the largest value from each unit, I used TensorFlow's function, $tf.greater\_equal$ to create a tensor with booleans. The booleans are chosen by comparing each unit to the largest value in that unit. By multiplying this tensor to the original input, I am left with the desired result of a tensor containing the largest activated nodes along with the rest whom are now set to zero.

Algorithm 2 The pseudo code for implementing the channel-out layer in TensorFlow

```
def Channel-out(Input):
    % Pass input through weight kernel and adding bias terms.
    Input ← Input × Weights
    Input ← Input + Bias

    % Reshape input into units
    Input ← Reshape(Input, (Nr Units, Size of Units))

    % Reduce input to the largest activation in each unit.
    Output ← Max(Input)

    % Set original activation where activation is largest and 0 where it's not.
    Output ← Where(Input == Output, Input, 0)

    % Reshape to original size.
    Output ← Reshape(Output, (Input's Original Shape))
    return Output
```

SCO

In section 2.6.5 I described how SCO is an extension of channel-out. The only difference between the two, is that SCO utilizes dynamic units instead of static. Therefore, when creating the SCO layer, the only change needed was grouping the nodes differently for each prediction. This was implemented by

3.3.3 Model Architecture

When choosing a network architecture, there are several ways to proceed. One way is to apply a grid search. A grid search is simply defining a grid of parameters to test, then running through all combinations and

Algorithm 3 The pseudo code for implementing the SCO layer in TensorFlow

```
def SCO(Input):
    % Pass input through weight kernel and adding bias terms.
    Input ← Input × Weights
    Input ← Input + Bias

    % Shuffle all the values
    InputShuflle ← Shuffle(Input)

    % Reshape input into units
    InputShuflle ← Reshape(InputShuflle, (Nr Units, Size of Units))

    % Reduce input to the largest activation in each unit.
    OutputShuffle ← Max(InputShuflle)

    % Set 1 where activation is largest and 0 where not.
    OutputShuffle ← Where(InputShuflle == OutputShuffle, 1, 0)

    % Un-shuffle all the values
    Output ← UnShuffle(Input)

    % Reshape to original size.
    Output ← Reshape(Output, (Input's Original Shape))

    % Multiply input with output to set all input that are not the largest, to zero.
    Input ← Input × Output

    return Input
```

choosing the highest performer. With a sufficient amount of tests, a grid search should converge towards an optimal architecture. Grid search is very common and there exists a large range of very complex varieties [27]. For my analysis I chose not to perform a grid search, for several reasons. The first being interpretability. Understanding a NN is already hard, allowing for complex and unique architectures would only add another layer of mysticism. The second is the size of the data set. The larger the data set, the larger the amount of data would be needed to adequately perform tests for each combination of parameters. Not only is this time-consuming, but trying to mediate this issue could lead to poor performance. The third and most important reason is that I wanted to experiment with the architectures. By manually tuning the parameters, I was able to achieve a far better understanding of the final architecture.

Dense Neural Network and PNN

The purpose of the simple dense NN, is to compare the more complex networks to what is usually considered a traditional NN. All layers are dense layers, meaning that all nodes in the previous layer are connected to the current layer, and likewise the current layer is connected to the next. The general structure of the network is summarized in figure 3.11, with the network label NN[24]. The figure shows a DNN with three hidden layers, all with 600 nodes each. All hidden layers utiliz the *LeakyReLU* activation (see section 2.6.4) with an $\alpha = 0.01$. The architecture is designed to perform deep-training, and will train on a training set where all mass combinations are included[25].
The PNN architecture is like the name suggests, included to represent the model proposed by the article by Baldi et al. [18]. The architecture is illustrated in figure 3.11, with the label PNN. The figure shows a practically identical structure to the dense-NN, with the only difference being in the input-layer. As was discussed in section 2.6.6, the PNN includes the new physics signal free parameters[26] alongside the features in the input layer.

[24] Note that the dense network will henceforth be referred to as NN
[25] Contrary to training one network for each mass combination.
[26] In our case, the masses of the BSM-particles.

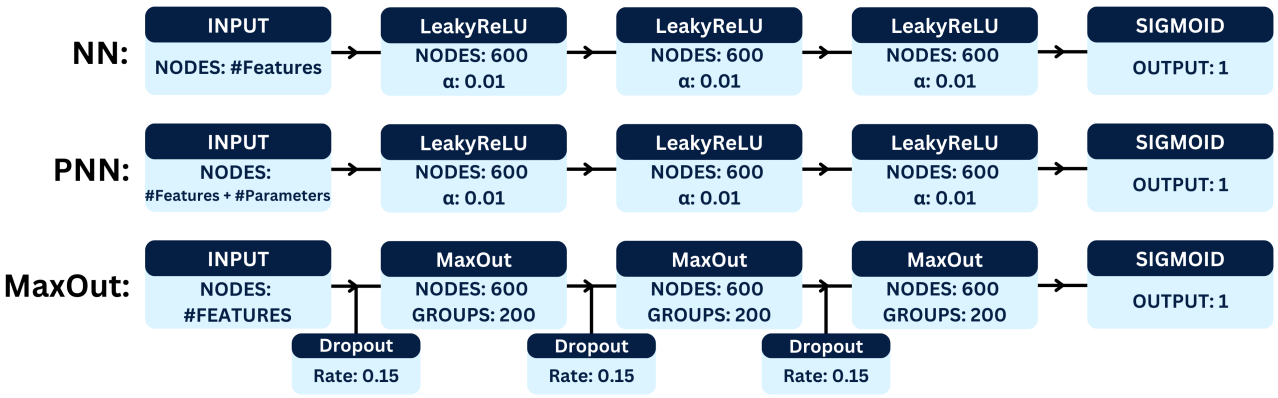


Figure 3.11: A visual summary of the workflow and framework use for the computational analysis.

MaxOut, Channel-Out and SCO

The ensemble methods are slightly than both the dense NN and PNN in terms of architecture. To limit the complexity for comparison reasons I choose to build an identical architecture for maxout, channel-Out and SCO, with the only difference being which of the three layers is used. In figure 3.11 I have illustrated the MaxOut architecture, with the label MaxOut. The figure shows a network with 6 hidden layers, 3 MaxOut and 3 dropout. The network alternates between drop out and MaxOut, starting with dropout and finishing with MaxOut. The MaxOut layers have 600 nodes each which reduce down to the 200 nodes with the largest activation in their respective groups. Each dropout layer has a dropout rate of 0.15. Channel-out and SCO have the same architecture to maxout, but replacing maxout with the two respectively.

XGBoost

The main motivation to apply XGBoost is simply to benchmark my analysis, and therefore not a lot of effort has been put into the design of the architecture. Therefore, the default parameters[27] of XGBoost has been used. The main parameters of the model are summarized as the following:

- η (learning-rate) = 0.3
- Max depth = 6
- Maximum number of trees = 100

3.4 Model Training and Validation

3.4.1 Training and Validating data

When building a ML-model, the usual approach is to divide your data into three sets; training, validation and testing. The training set is used to tune the internal parameters of the model, i.e. the weights and biases of a NN or the cuts of a DT. The validation set is used to tune the hyperparameters of the model, i.e. the architecture of the NN or the maximum depth of the DT etc. The test set should only be used when the model is finished, and is used to benchmark the models' performance. In our case, the performance we are interested in, is the performance on the full MC-bakgroundset and its comparison to the measured collision data.[28]. Therefore, in this analysis only two sets of data will be used, training and validation where both are sampled from the simulated MC data set. In theory, one could even just use one data set (training) including all the data, but the second was added as a precaution to reduce overfitting when applied to the measured collision data.
The overarching strategy is summarized in the following points:

- Shuffle the data set.
- Split the data set in two, training (80%) and validation (20%)

[27] See https://xgboost.readthedocs.io/en/stable/parameter.html for a complete overview of default parameters.
[28] See section INSERT THE APPROPRIATE SECTION.

- Scale the two data set such that the sum of the weights of the background is equal to the sum of the weights of the signal in each data set.
- Scale both data sets using the Standard Scalar approach (see section 2.4.1) using the parameters of the training set on both sets.

The first step ensures an equal distribution of processes in both data sets. The $80 - 20\%$ is a popular practice in ML (see [28]) and was chosen here for convenience. The third step is done to ensure an equal prioritization for background classification and signal classification. In other words, if there was a large imbalance between signal and background, the model would be motivated to tune more towards one trend than another. The final step is motivated in the section regarding data handling 2.4.1.

3.4.2 Training strategy

To best compare the different ML-models, I decide to apply the same training strategy to all (including the BDT). The strategy is simply to train the model with the training set, then apply an early-stoppage algorithm (see section 2.5.1) with a cut-off based on the performance on the validation set. For every epoch, the model makes a prediction on the validation data set, and logs the results. If more than 10 epochs go by without improving on the epoch with the best result, training stops and the weights of the best epoch are reset. This strategy was repeated for the BDT, but logging for each new additional tree instead of epoch. The performance from each epoch was measured in AUC (see section 2.9.1), where the AUC was calculated with the same weighting as in training (50% signal and 50% background). The distribution of signal vs background is important when studying AUC, as it will greatly affect the value. For example, a classifier which predicts all data to be background is a poor classifier. But, the larger the amount of background relative to signal, the higher the AUC would be measured for such a classifier.

Chapter 4

Results & Discussion

In this chapter I will present the results from my analysis. First, I will present and discuss my results by comparing different ML- models.

4.1 Benchmarking the Analysis with Boosted Trees

Boosted trees have been an essential part of HEP analysis for many years (see [29] and [30]). The XGBoost framework has likewise been used as the benchmark for all ML analysis, often chosen for its performance in sensitivity and extreme HPC attributes. Another reason for its popularity is the incredibly simple API, allowing to create and predict a model in two lines of code. Additionally, its incredible boosting capabilities mean that it is little effected by variation in its structure. This leads many to the conclusion that the default parameters (see section 3.3.3) are often the best.
I choose to perform a sensitivity analysis for the original signal data set using a XGBoost model. The results will act as a benchmark when performing further testing with NN-variants. In figure 4.1 I have presented a grid displaying the achieved significance (see section 2.9.2) for each mass combination in the original signal data set. In the figure we can observe that the XGBoost model performs better for smaller masses. This results can be somewhat counterintuitive, due to signal with smaller masses having a larger resemblance to background than signal with large masses. The explanation is simple that there is much more of the smaller mass signal than there are larger mass signals. By studying figure 3.2 we know that there are a total of 134 events with ($\tilde{\chi}_1 = 200$, $\tilde{\chi}_2 = 400$GeV) compared to 6 events with ($\tilde{\chi}_1 = 400$, $\tilde{\chi}_2 = 800$GeV). Not only does this mean that the model will have had more small mass signal to train on than large mass, but also that a potential signal region would have to keep far more of the large mass signal to achieve a high significance.
To further investigate the performance of the XGBoost model, I have drawn the distribution of the output for the entire background data set as well as for signal with 4 different mass combinations. The result is found in figure 4.2. The figure shows the output of the XGBoost model with the full output range 4.2a and the output ranging from 0.9-1.00 (4.2b). From the figure we observe a clear separation between signal and background, where most of the signal is given high values (> 0.9) and most of the background is given small values (< 0.1). To further support the effect of statistics in the signal, we can take note of the amount of signal in the higher range of the output (0.975-1). Although the model is able to achieve a much higher effectiveness (is able to preserve more of the signal) for higher masses, there is still at least 4 times as much signal with ($\tilde{\chi}_1 = 200$, $\tilde{\chi}_2 = 400$GeV) (the lightest mass combination in the figure) then there are of any other.

4.2 Dense Neural Networks

4.2.1 Deep vs Shallow

4.2.2 Parameter Specific Networks and Interpolation

4.3 Ensemble methods

4.3.1 Visualizing Sparse Pathways

As mentioned in section 3.3.2, the MaxOut and ChannelOut layers applied in this analysis were created by me using the TensorFlow API's. As such, I found it imperative to make sure that the layers worked as

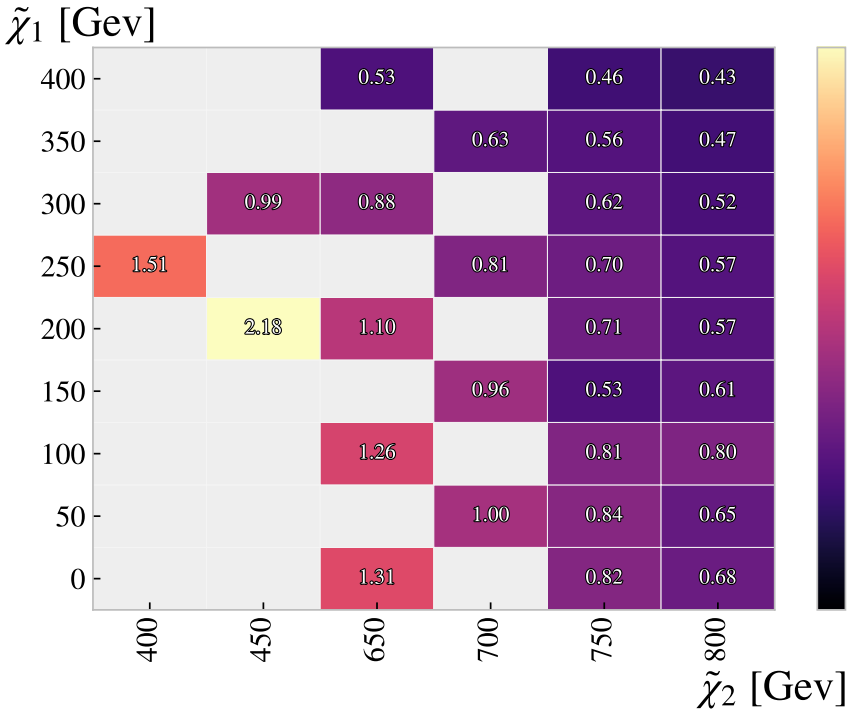
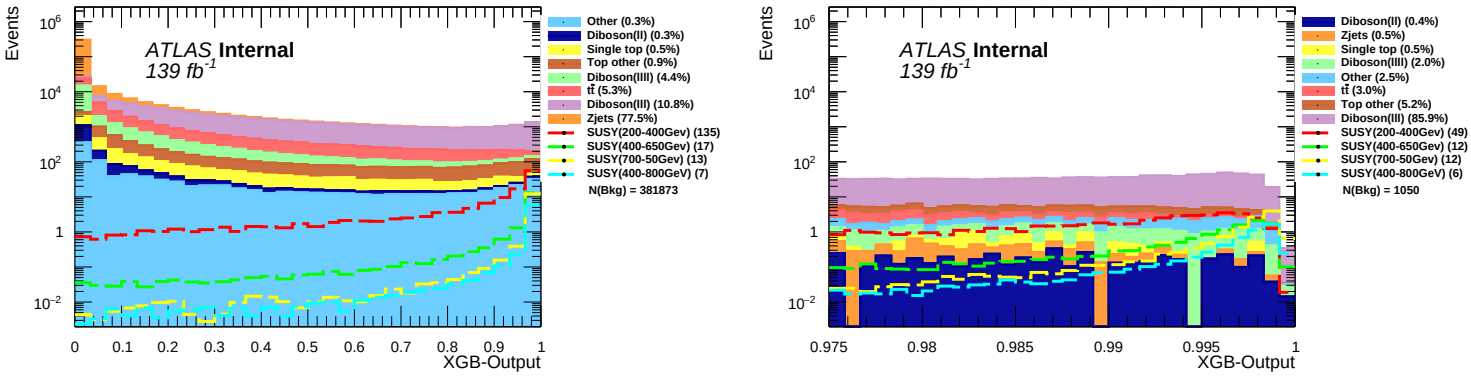


Figure 4.1: A grid displaying the achieved significance on the original signal set, using the signal region created by the *XGBoost* network.



(a) **(b)**

Figure 4.2: The output distribution from a trained XGBoost model for the background and signals with 4 different mass combinations: ($\tilde{\chi}_1 = 200$, $\tilde{\chi}_2 = 400$GeV), ($\tilde{\chi}_1 = 400$, $\tilde{\chi}_2 = 650$GeV), ($\tilde{\chi}_1 = 700$, $\tilde{\chi}_2 = 50$GeV) and ($\tilde{\chi}_1 = 400$, $\tilde{\chi}_2 = 800$GeV). The figure includes the full output range (4.2a) and the output ranging from 0.9-1.00 (4.2b).

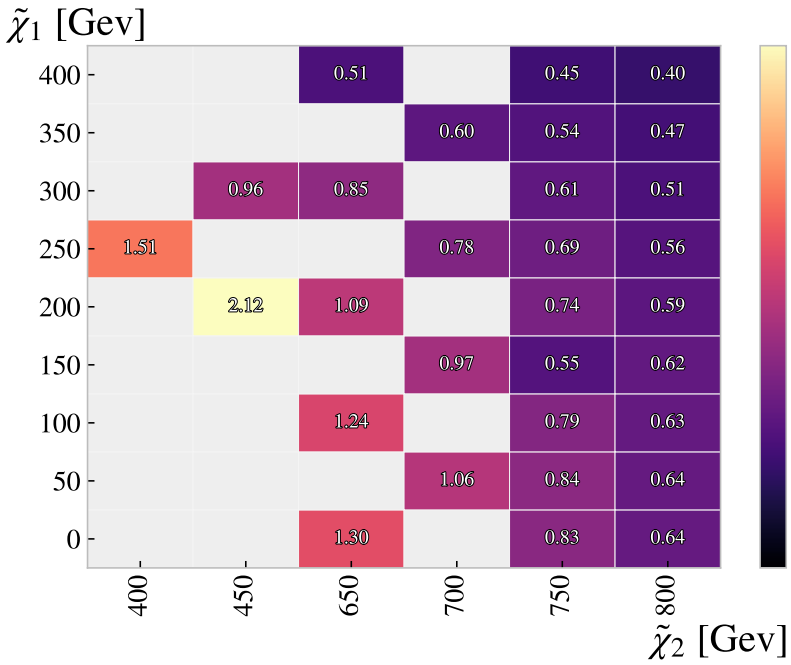


Figure 4.3: A grid displaying the achieved significance on the original signal set, using the signal region created by the shallow NN network.

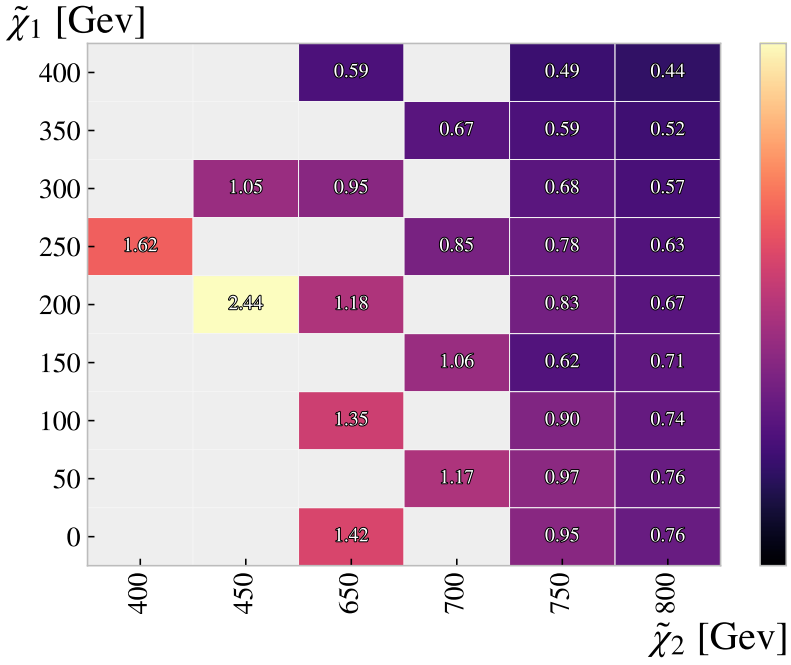
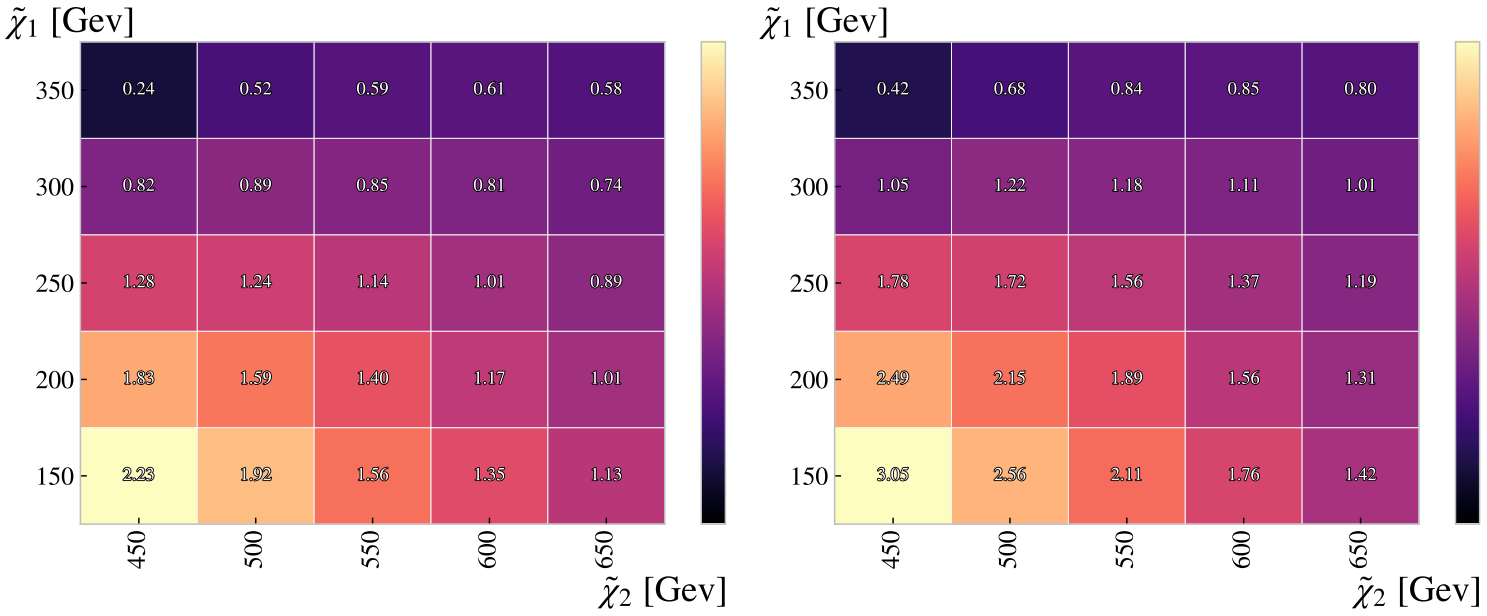


Figure 4.4: A grid displaying the achieved significance on the original signal set, using the signal region created by the deep NN network.



(a) **(b)**

Figure 4.5

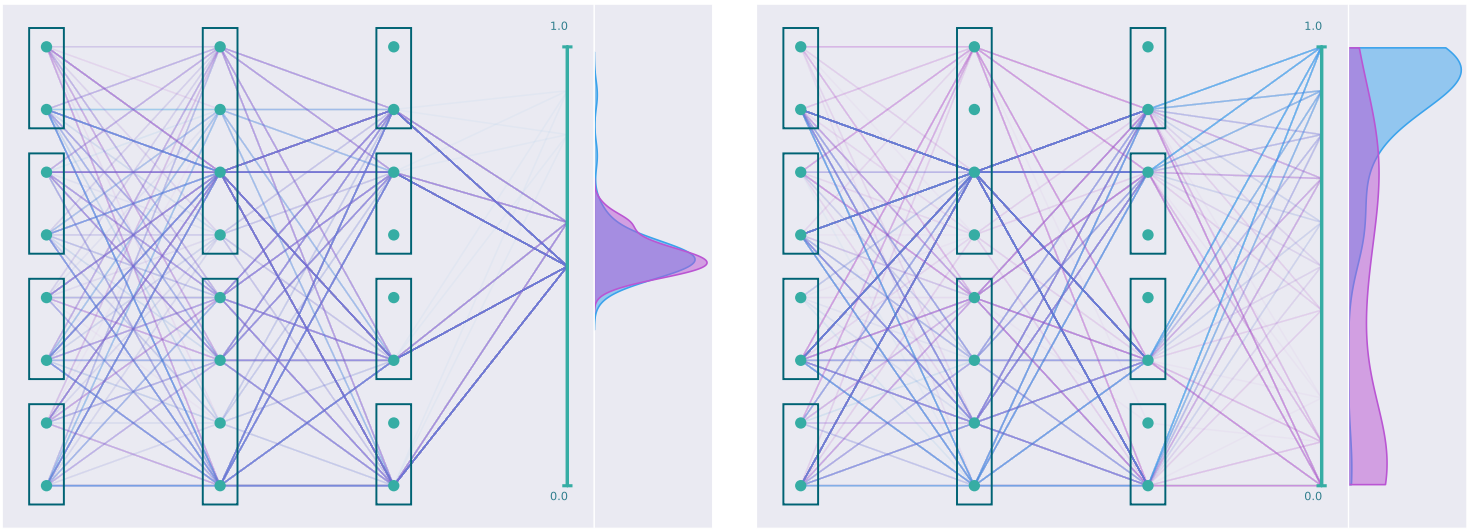
intended. To do this I created a small network with three layers, with 8 nodes each, all applying MaxOut layers with 4, 2 and 4 groups respectively. In this section I will dissect the activations of said network before and after training.
In figure 4.6a, I have plotted the activation of 100 randomly sampled collisions, 50 background and 50 signal for an untrained model. Adjacent, I plotted the resulting distribution of the output. From the figure we observe little to no deviation between the activation from the signal and the background. This is mirrored in the distribution of the output which is centered around the middle of the range. A small exception can be found for larger output values, where we observe a small distribution of signal values. This is due outliers in the signal data set.
In figure 4.6b, I plotted a similar plot as described above, but using a trained model. In this figure the output is far more separated, and we see noticeable differences in the activation of nodes. To highlight the difference in activation, I drew two new figures where the signal (4.7a) and background (4.7b) were drawn individually. In the two figures we notice that there is still a noticeable variation in the activation for both signal and background. This is due to outliers in the data, but also the variation of trend. Regardless, there are still clear trends in activation which lets us know that the model has found specific paths in the network for signal and background separately. Most noticeably, the two group in the middle hidden layer highlight this fact. In the case of the signal, the upper group show a clear favoritism to the second bottom node. For the background this is also partly true, but with far more spread in the other nods. Similarly, in the bottom group (in the same layer), the background shows large activation in the uppermost node, while the signal data does not.

4.3.2 Sensitivity Result

4.4 Parametrized Neural Network

4.4.1 Discriminating Masses

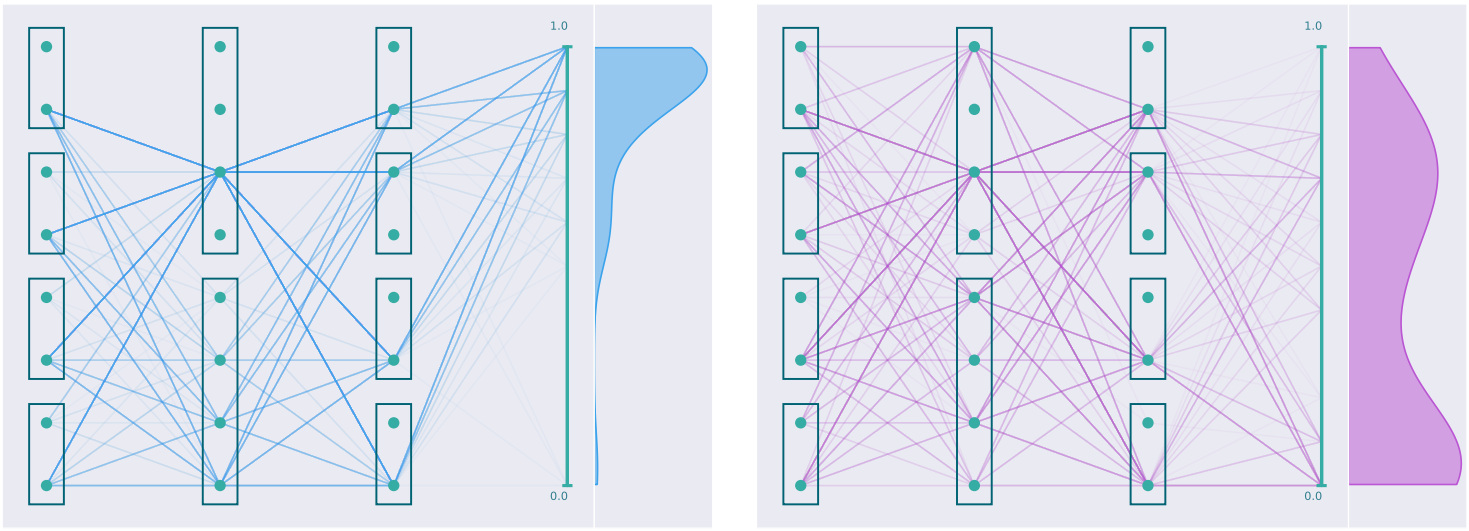
In figure 4.12a I have drawn the distribution of the output from the trained PNN architecture (see section 3.3.3). The model was trained using the FS-MLM data set. In figure 4.12a, 4 signals have been included; ($\tilde{\chi}_1 = 50$, $\tilde{\chi}_2 = 250$GeV), ($\tilde{\chi}_1 = 100$, $\tilde{\chi}_2 = 200$GeV), ($\tilde{\chi}_1 = 200$, $\tilde{\chi}_2 = 300$GeV) and ($\tilde{\chi}_1 = 150$, $\tilde{\chi}_2 = 250$GeV). All data, including both signal and background were given the values of 50 and 250. As was the hope, the PNN performs better on the signal for which the parameter choice was correct ($50 - 250$) and worse on the rest. This means that the PNN model is tuned to each signal combination.



(a) **(b)**

Figure 4.6: A calculated visualization of a 3 layer MaxOut network. Each path represents a data point where all connected nodes were the largest activation in their respective group. The distribution on the far right represent the output distribution. The figure to the left (4.7a) is the result before training and the figure to right (4.7b) is after.

.



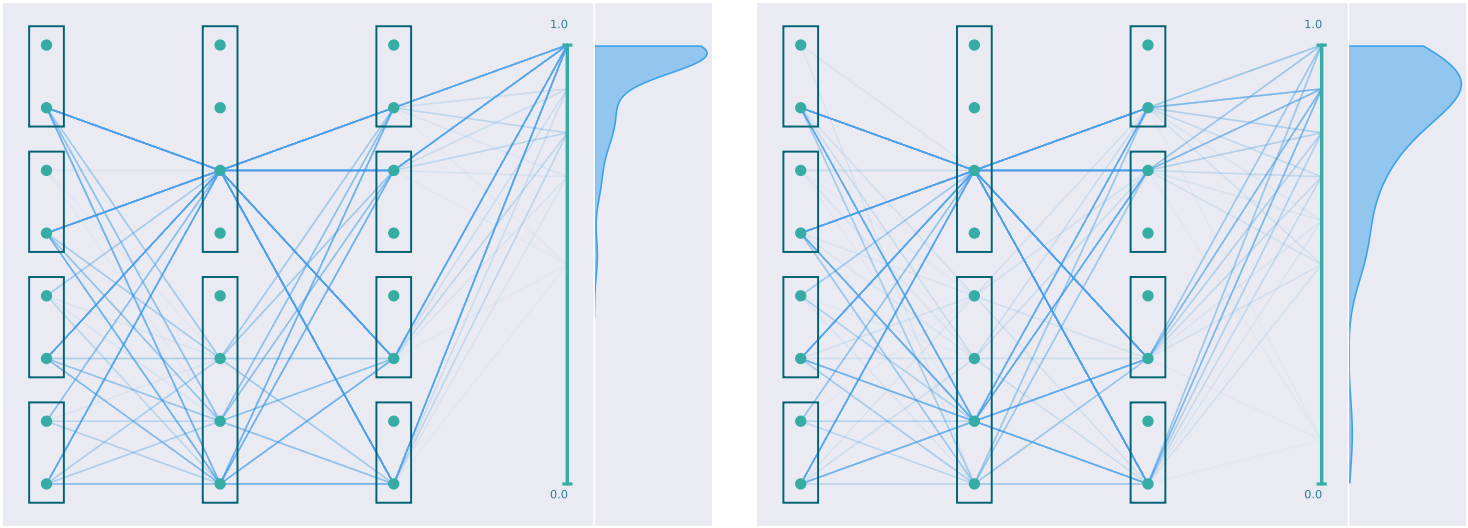
(a) **(b)**

Figure 4.7: A calculated visualization of a 3 layer MaxOut network. Each path represents a data point where all connected nodes were the largest activation in their respective group. The distribution on the far right represent the output distribution and the figure with blue paths 4.7a is the result of signal and the figure pink 4.7b.

.

Label \ Channel	$(50, 250)$	$(100, 200)$	$(150, 300)$	$(200, 300)$
$(50, 250)$	**80.8**%	45.8%	**77.5**%	50.1%
$(200, 300)$	77.3%	**54.6**%	76.3%	**59.0**%

Table 4.1: A listing of the remaining procentage of each mass combination in the output range 0.95-1.00 using the labels ($\tilde{\chi}_1 = 50$, $\tilde{\chi}_2 = 250$GeV) and ($\tilde{\chi}_1 = 200$, $\tilde{\chi}_2 = 300$GeV) respectively.



(a) (b)

Figure 4.8: A calculated visualization of a trained MaxOut network with 3 hidden layers. Each path represents a data point where all connected nodes were the largest activation in their respective group. The figure to the left (4.8a) is a result of signal with ($\tilde{\chi}_1 = 50$, $\tilde{\chi}_2 = 250$GeV) and the right (4.8b) ($\tilde{\chi}_1 = 200$, $\tilde{\chi}_2 = 300$GeV).

(a) (b)

Figure 4.9: A cut-out of the fifth and sixth neuron (counting from the top) in the second hidden layer, activated by the signal with ($\tilde{\chi}_1 = 50$, $\tilde{\chi}_2 = 250$GeV) 4.9a and ($\tilde{\chi}_1 = 200$, $\tilde{\chi}_2 = 300$GeV) 4.9b.

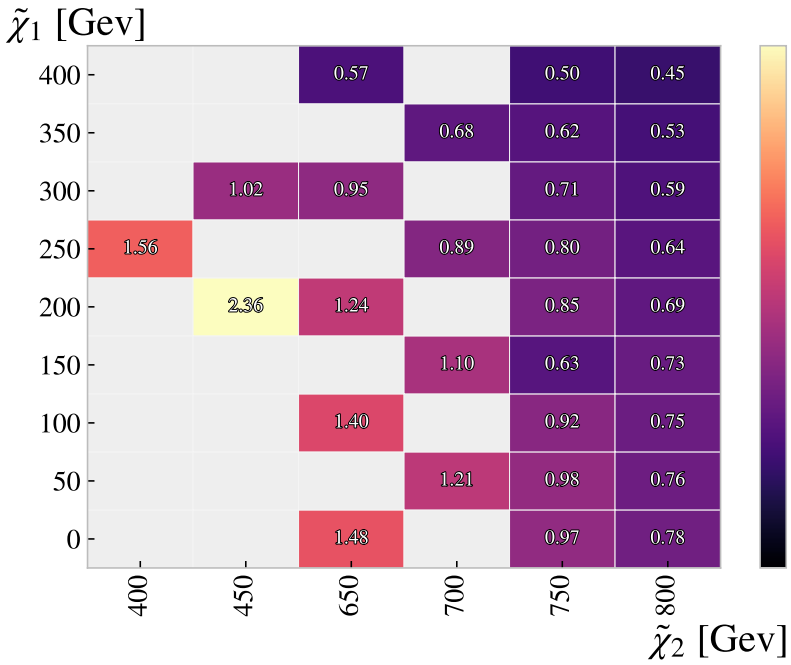


Figure 4.10: A grid displaying the achieved significance on the original signal set, using the signal region created by the *MaxOut* network.

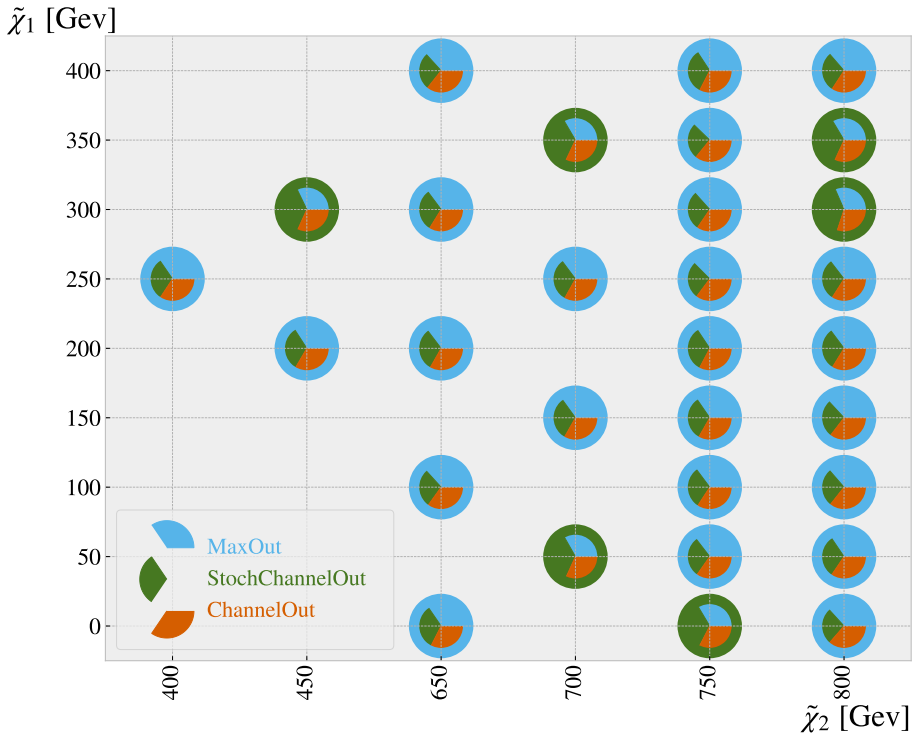
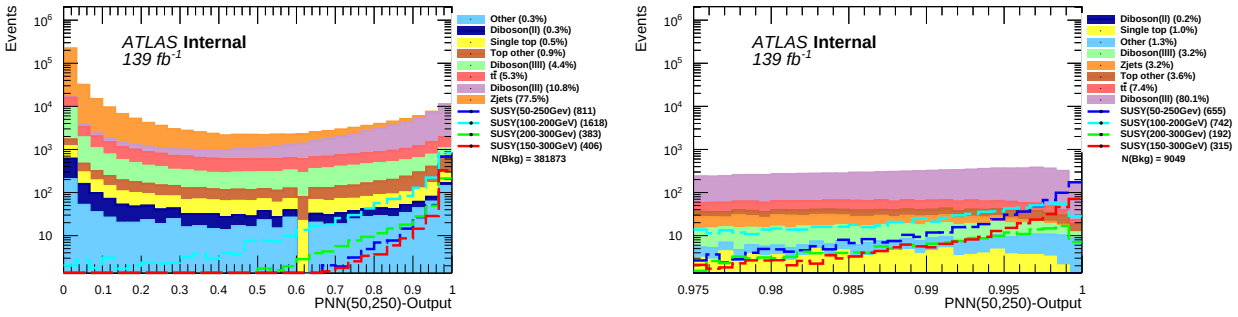
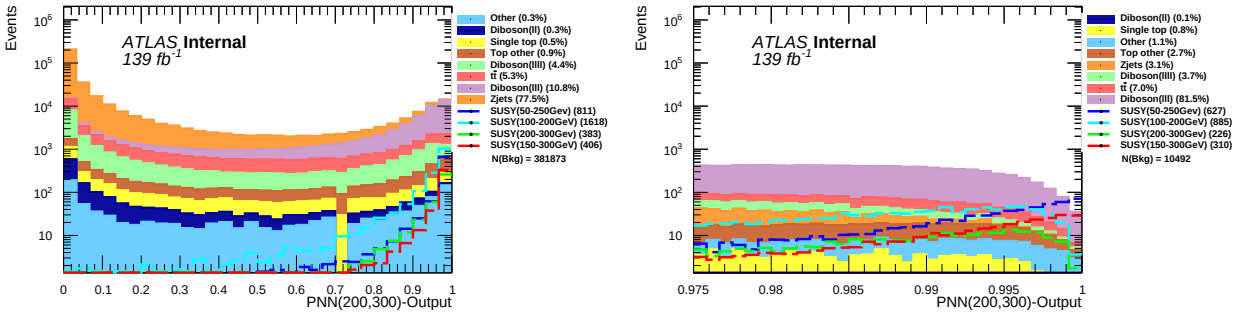


Figure 4.11: A sensitivity comparison between the ensemble methods (MaxOut, SCO, ChannelOut) on the original signal data. The size of the ”pie” represents the relative size of the significance and the color around each point displays the method with the largest sensitivity for the respective combination.



(a) **(b)**

Figure 4.12: The output distribution from a trained PNN model for the background and signals with 4 different mass combinations: ($\tilde{\chi}_1 = 50$, $\tilde{\chi}_2 = 250$GeV), ($\tilde{\chi}_1 = 100$, $\tilde{\chi}_2 = 200$GeV), ($\tilde{\chi}_1 = 200$, $\tilde{\chi}_2 = 300$GeV) and ($\tilde{\chi}_1 = 150$, $\tilde{\chi}_2 = 250$GeV), and where all the data were given the parameter features of ($\tilde{\chi}_1 = 50$, $\tilde{\chi}_2 = 250$GeV). The figure includes the full output range (4.12a) and the output ranging from 0.975-1.00 (4.12b).



(a) **(b)**

Figure 4.13: The output distribution from a trained PNN model for the background and signals with 4 different mass combinations: ($\tilde{\chi}_1 = 50$, $\tilde{\chi}_2 = 250$GeV), ($\tilde{\chi}_1 = 100$, $\tilde{\chi}_2 = 200$GeV), ($\tilde{\chi}_1 = 200$, $\tilde{\chi}_2 = 300$GeV) and ($\tilde{\chi}_1 = 150$, $\tilde{\chi}_2 = 250$GeV), and where all the data were given the parameter features of ($\tilde{\chi}_1 = 200$, $\tilde{\chi}_2 = 300$GeV). The figure includes the full output range (4.12a) and the output ranging from 0.975-1.00 (4.12b).

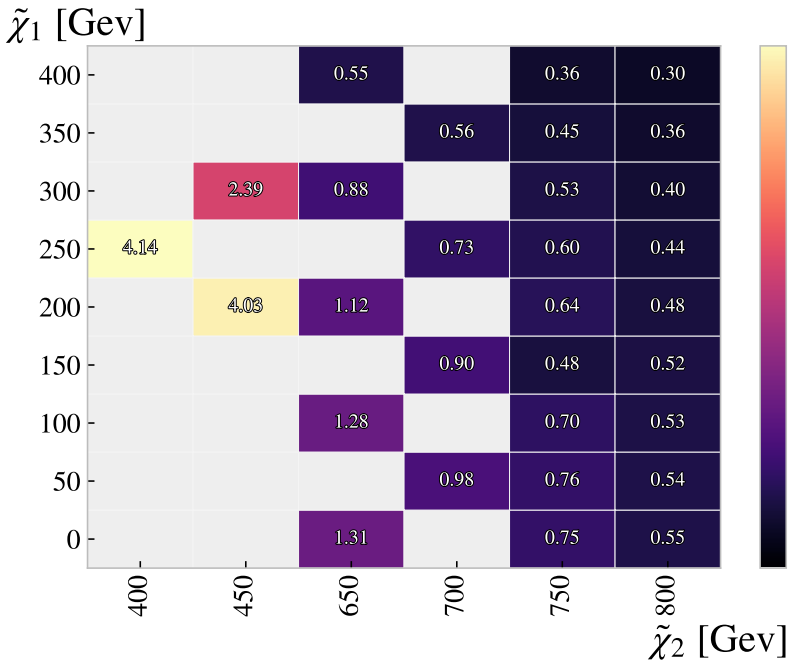
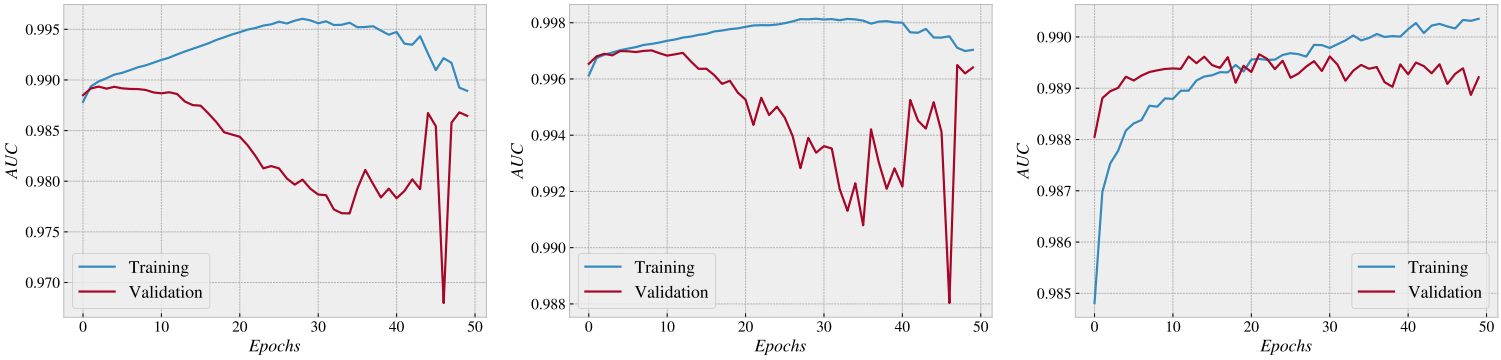


Figure 4.14: A grid displaying the achieved significance on the original signal set, using the signal region created by the PNN network.



(a) **(b)** **(c)**

Figure 4.15

4.4.2 Sensitivity Result

4.5 Comparing Machine Learning Models

4.5.1 Training History and Overfitting

4.6 Increasing Sensitivity through PCA

So far in the analysis, all features have been given an equal weighting in the beginning of training, But, as mentioned in section 2.8, not all features contribute to an effective signal region. In section 2.4.2, I explained how through the use of PCA, we are able to create a new set of features which can be ordered by amount of variance. In this way, we can reduce the dimensionality of the data set, while at the same time preserving most of the variance. Inn this section I will present the results of training on a data set which has gone through such a PCA.
In this analysis I performed a PCA on the data set, and demanded that 99.99% of the variance should be preserved. In doing so, 5 features were removed.

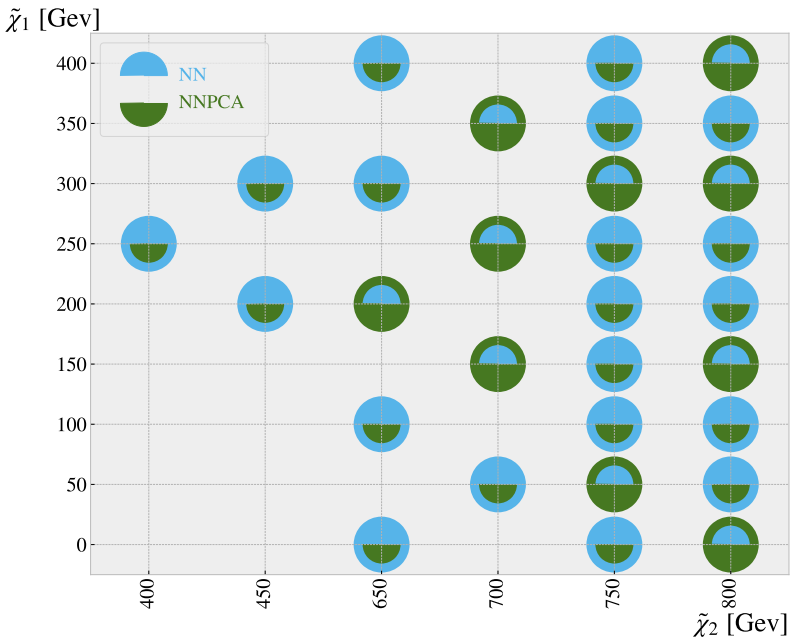


Figure 4.16: A grid displaying the achieved significance on the original signal set, using the signal region created by the NN network. A PCA analysis has been applied to the data being utilized in this result.

NN and PCA

MaxOut Network and PCA

PNN and PCA

4.7 Comparing Models on Full Statistics Signal

4.7.1 Running on Full Statics

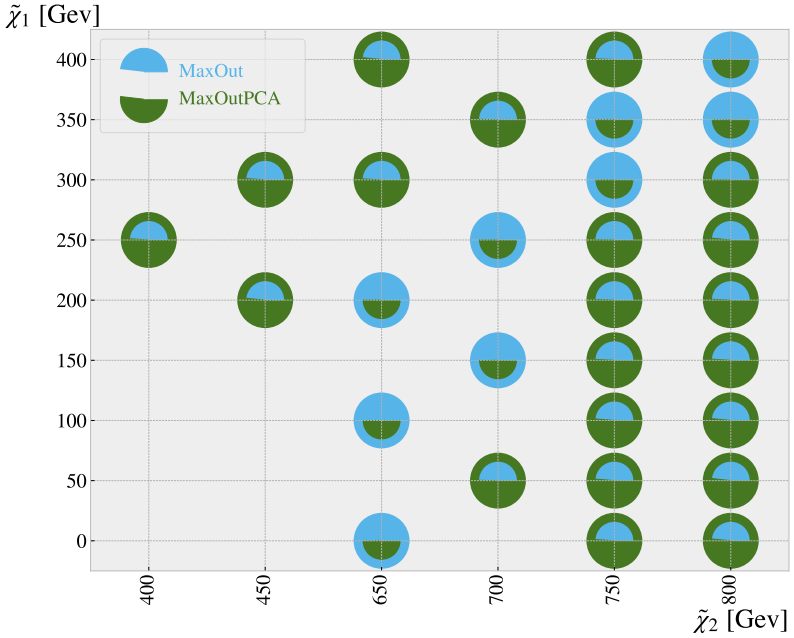


Figure 4.17: A grid displaying the achieved significance on the original signal set, using the signal region created by the NN network. A PCA analysis has been applied to the data being utilized in this result.

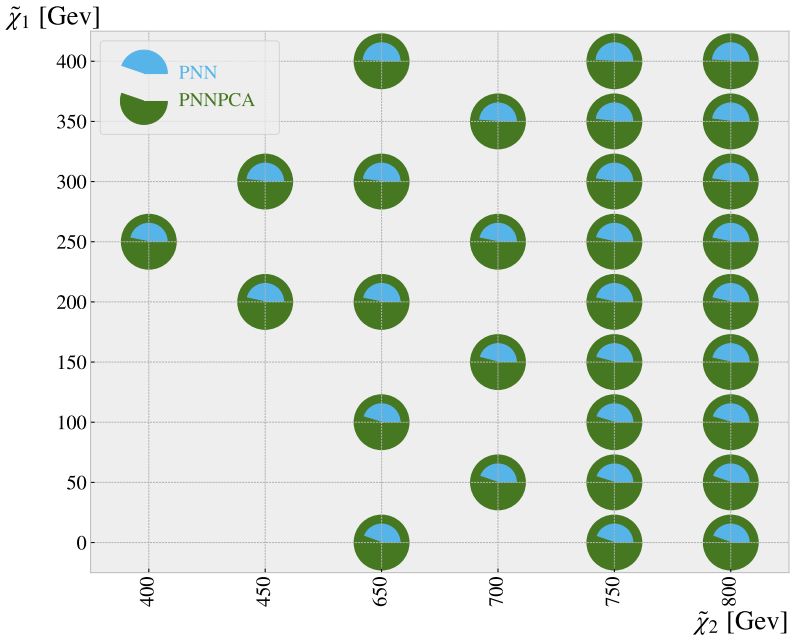


Figure 4.18: A grid displaying the achieved significance on the original signal set, using the signal region created by the NN network. A PCA analysis has been applied to the data being utilized in this result.

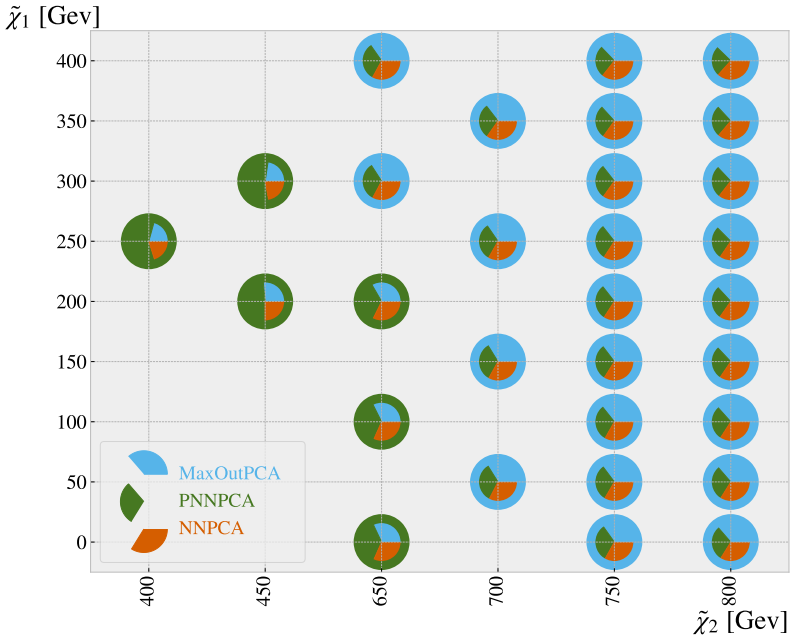


Figure 4.19: A grid displaying the achieved significance on the original signal set, using the signal region created by the NN network. A PCA analysis has been applied to the data being utilized in this result.

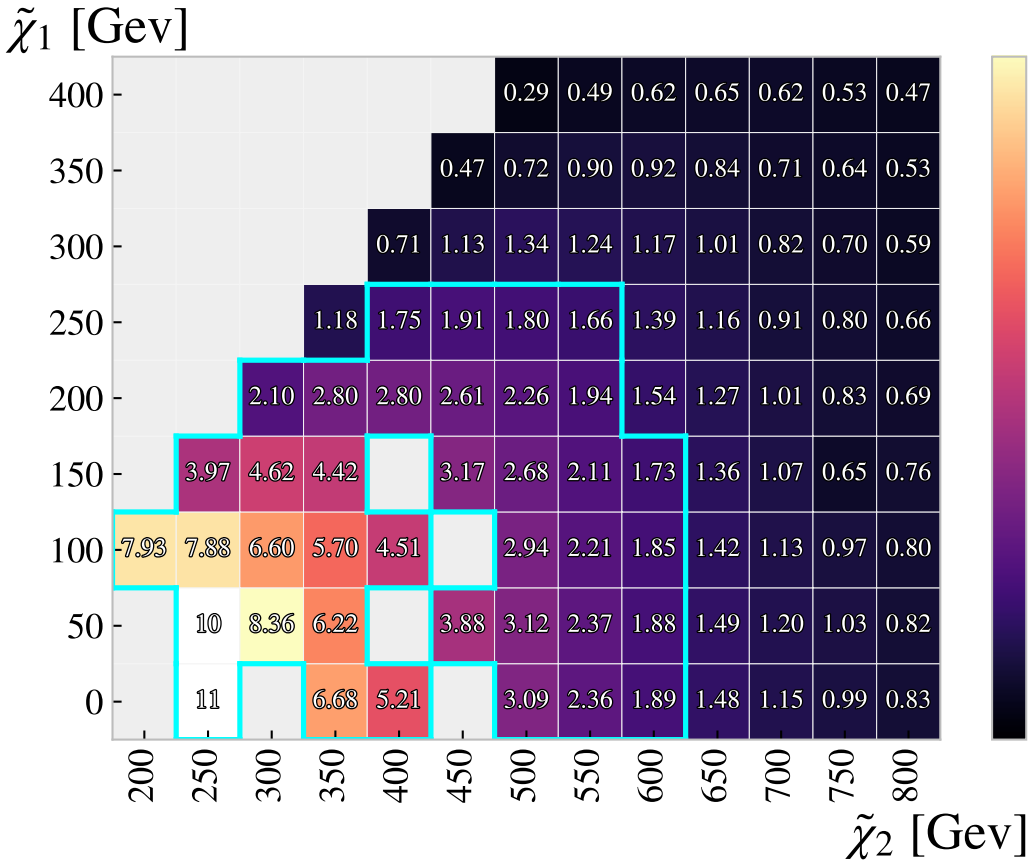


Figure 4.20: A grid displaying the achieved significance on the full statistics signal set, using the signal region created by the *MaxOut* network.

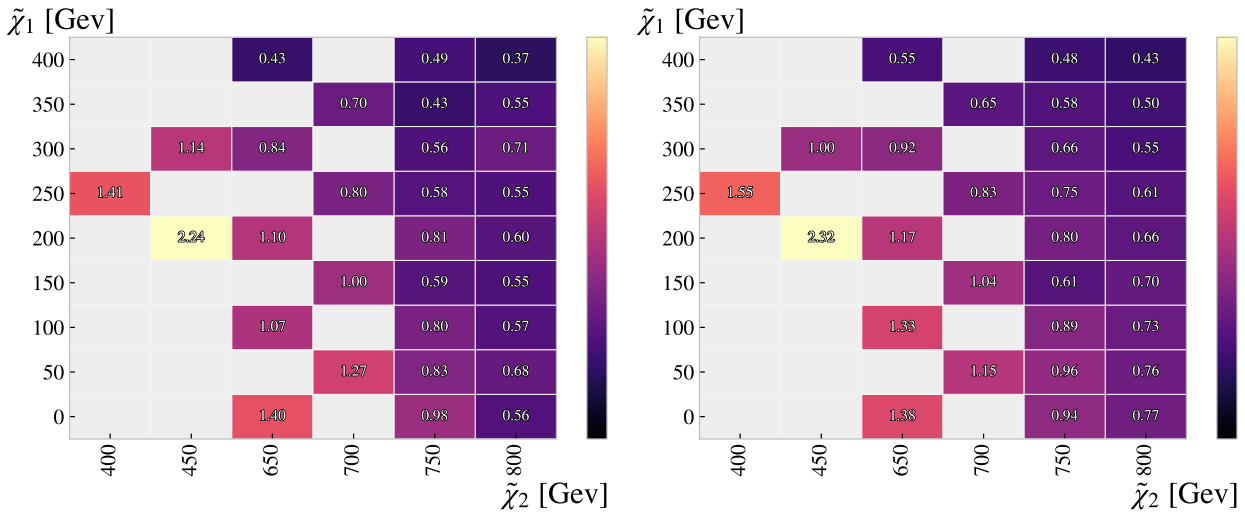
Appendices

Appendix A

A.1 Sensitivity Grids

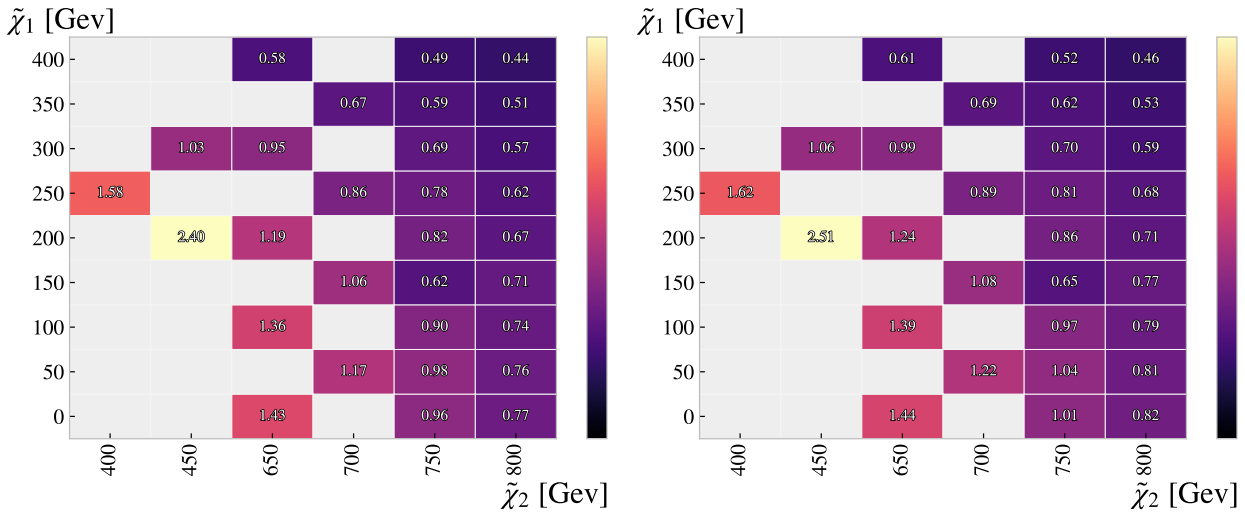
A.1.1 Ensembles

A.1.2 Increasing Sensitivity Through



(a) **(b)**

Figure 21: A grid displaying the achieved significance on the original signal set, using the signal region created by the SCO 21a and a channel-out network 21b. A PCA analysis has been applied to the data being utilized in this result.



(a) **(b)**

Figure 22: A grid displaying the achieved significance on the original signal set, using the signal region created by the NN 22a and a maxout network 22b. A PCA analysis has been applied to the data being utilized in this result.

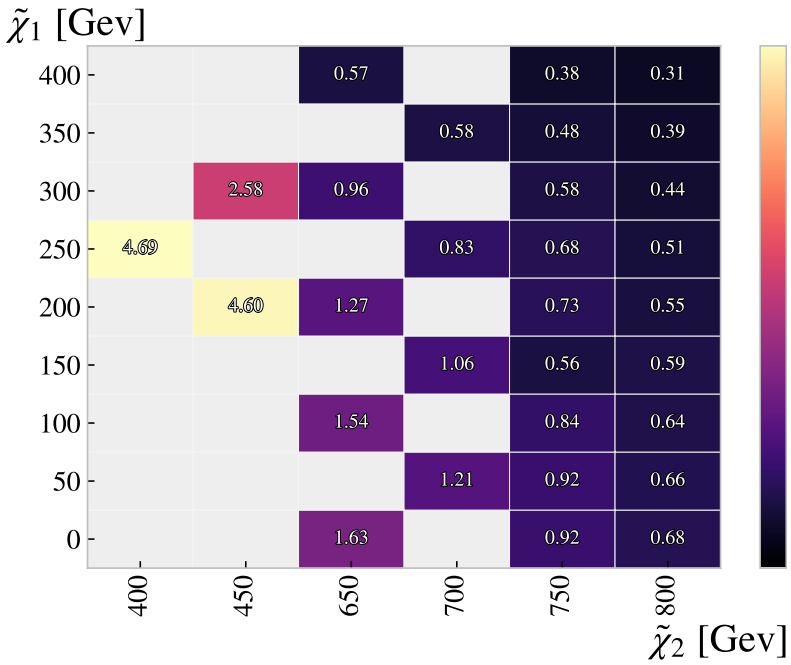


Figure 23: A grid displaying the achieved significance on the original signal set, using the signal region created by the PNN network. A PCA analysis has been applied to the data being utilized in this result.

A.2 The Features

Feature Name	Description
P_t	Transverse momentum
η_t	Pseudo rapidity
ϕ_t	Azimuthal angle
M_T	Transverse mass
$Charge$	EM charge
$Flavour$	Particle type
E_T^{miss}	Transverse missing energy
$\phi(miss)$	Azimuthal angle of the missing energy
M_{lll}	Trilepton mass
$M_{ll}(OSSF)$	Mass of the OSSF pair
Sig E_T^{miss}	Significance of E_T^{miss}
$H_t(lll)$	Sum of P_t for all three leptons
$H_t(SS)$	Sum of P_t for the SS pair
$H_t(lll) + E_T^{miss}$	-
ΔR	Distance defined in the $\eta - \phi$ space
Flavor combo	Combination of flavors for all three leptons
Nr of signal Jets	Nr of jets passing the signal criteria
M_{jj}	Mass of the leading jet pair
Nr of B-jets(77)	-
Nr of B-jets(85)	-

Table 2: A summary and description of all features used in this analysis.

A.3 The implementation of Channel-Out, SCO and Maxout

Channel-out

For the activation function defined to create the channel-out layer, I again grouped the nodes similarly as I did for max-out. Instead of returning the largest value from each unit, I used TensorFlow's function, *tf.greater_equal* to create a tensor with booleans. The booleans are chosen by comparing each unit to the largest value in that unit. By multiplying this tensor to the original input, I am left with the desired result of a tensor containing the largest activated nodes along with the rest whom are now set to zero.

```
def call(self, inputs: tf.Tensor,mask: tf.Tensor = None) -> tf.Tensor:
    # Pass input through weight kernel and adding bias terms.
    inputs = gen_math_ops.MatMul(a=inputs, b=self.kernel)
    inputs = nn_ops.bias_add(inputs, self.bias)

    num_inputs = inputs.shape[0]
    if num_inputs is None:
    num_inputs = -1

    # Reshaping inputs such that they are grouped correctly
    num_competitors = self.units // self.num_groups
    new_shape = [num_inputs, self.num_groups, num_competitors]
    inputs = tf.reshape(inputs, new_shape)

    # Finding maximum activations and setting losers to 0.
    outputs = tf.math.reduce_max(inputs, axis=-1, keepdims=True)
    outputs = tf.where(tf.equal(inputs, outputs), outputs, 0.)
    # Reshaping outputs to original input shape
    outputs = tf.reshape(outputs, [num_inputs, self.units])

    self.counter = outputs
    return outputs
```

Listing 1: Python implementation for the custom activation function used to define the channel-out layer.

```
    def call(self, inputs: tf.Tensor, mask: tf.Tensor = None) -> tf.Tensor:
    inputs = gen_math_ops.MatMul(a=inputs, b=self.kernel)
    inputs = nn_ops.bias_add(inputs, self.bias)
    #tf.print(inputs)
    num_inputs = inputs.shape[0]

    if num_inputs is None:
```

```
    num_inputs = -1

    shuffle_index = tf.random.shuffle(self.index)
    unshuffle_index = tf.tensor_scatter_nd_update(tensor = self.zeros ,
                                                  indices = tf.reshape(shuffle_index
                                                      ,
                                                                       [inputs.shape
                                                                           [1],1]),
                                                  updates = self.index)
    inputs_s = tf.gather(inputs, shuffle_index, axis = 1)


    # Reshaping inputs such that they are grouped correctly
    num_competitors = self.units // self.num_groups
    new_shape = [num_inputs, self.num_groups, num_competitors]
    inputs_s = tf.reshape(inputs_s, new_shape)

    # Finding maximum activations and setting losers to 0.
    outputs = tf.math.reduce_max(inputs_s, axis=-1, keepdims=True)

    outputs = tf.where(tf.equal(inputs_s, outputs), 1.0, 0.)
    # Reshaping outputs to original input shape
    outputs = tf.reshape(outputs, [num_inputs, self.units])

    outputs = tf.gather(outputs, unshuffle_index, axis = 1)
    outputs = tf.multiply(inputs, outputs)
    self.counter = outputs
    return outputs
```

Listing 2: Python implementation for the custom activation function used to define the SCO layer.

```
def call(self, inputs: tf.Tensor) -> tf.Tensor:
    # Passing input through weight kernel and adding bias terms
    inputs = gen_math_ops.MatMul(a=inputs, b=self.kernel)
    inputs = nn_ops.bias_add(inputs, self.bias)

    num_inputs = inputs.shape[0]
    if num_inputs is None:
        num_inputs = -1
    num_competitors = self.units // self.num_groups
    new_shape = [num_inputs, self.num_groups, num_competitors]

    # Reshaping outputs such that they are grouped correctly
    inputs = tf.reshape(inputs, new_shape)
    # Finding maximum activation in each group
    outputs = tf.math.reduce_max(inputs, axis=-1,keepdims=True)

    counter = tf.where(tf.equal(inputs, outputs), outputs, 0.)
    # Reshaping outputs to original input shape
    self.counter = tf.reshape(counter, [num_inputs, self.units])

    return tf.reshape(outputs,[num_inputs, self.num_groups])
```

Listing 3: Python implementation for the custom activation function used to define the max-out layer.

Acronyms

API Application Programming Interface

ATLAS A Toroidal LHC Apparatus

AUC Area Under the Curve

BDT Boosted Decision Trees

BEH Brout-Englert-Higgs

BSM Beyond Standard Model

CC Cut-and-Count

CNN Convolutional Neural Network

CP Charge-Parity

DNN Deep Neural Networks

DT Decision Trees

EM Electromagnetic

FFNN Feed-Forward Neural Network

HEP High Energy Physics

HPC High Performance Computing

LHC Large Hadron Collider

LWTA Local-Winner-Takes-All

MC Monte Carlo

ML Machine Learning

MSE Mean Squared Error

NN Neural Network

OSSF Opposite Sign Same Flavour

PCA Principal Component Analysis

PNN Parametrized Neural Network

QCD Quantum Chromo Dynamics

QED Quantum Electro Dynamics

QFT Quantum Field Theory

RNN Recursive Neural Network

ROC Receiver Operating Characteristic

SCO Stochastic-Channel-Out

SM Standard Model

SP Superpartner

SR Signal region

SS Same Sign

SUSY Supersymmetry

Bibliography

[1] G. Aad, T. Abajyan, B. Abbott, J. Abdallah, S. A. Khalek, A. Abdelalim et al., *Observation of a new particle in the search for the standard model higgs boson with the ATLAS detector at the LHC*, *Physics Letters B* **716** (sep, 2012) 1–29.

[2] J. J. T. M. F.R.S., *Xlviii. the relation between the atom and the charge of electricity carried by it*, *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* **40** (1895) 511–544, [https://doi.org/10.1080/14786449508620801].

[3] P. E. R. F.R.S., *Lxxix. the scattering of α and β particles by matter and the structure of the atom*, *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* **21** (1911) 669–688, [https://doi.org/10.1080/14786440508637080].

[4] F. Reines and C. L. Cowan, *The neutrino*, *Nature* **178** (1956) 446–449.

[5] J. Bernstein, *Spontaneous symmetry breaking, gauge theories, the higgs mechanism and all that*, *Rev. Mod. Phys.* **46** (Jan, 1974) 7–48.

[6] Super-Kamiokande Collaboration collaboration, Y. Fukuda, T. Hayakawa, E. Ichihara, K. Inoue, K. Ishihara, H. Ishino et al., *Evidence for oscillation of atmospheric neutrinos*, *Phys. Rev. Lett.* **81** (Aug, 1998) 1562–1567.

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

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

[9] O. Antipin, D. Atwood and A. Soni, *Search for rs gravitons via wlwl decays*, *Physics Letters B* **666** (2008) 155–161.

[10] E. Oks, *Brief review of recent advances in understanding dark matter and dark energy*, .

[11] D. Chung, L. Everett, G. Kane, S. King, J. Lykken and L.-T. Wang, *The soft supersymmetry-breaking lagrangian: theory and applications*, *Physics Reports* **407** (2005) 1–203.

[12] J. Pequenao, "Event Cross Section in a computer generated image of the ATLAS detector.." 2008.

[13] E. Gramstad, *Searches for supersymmetry in di-lepton final states with the ATLAS detectorat* $\sqrt{s}$ *= 7 TeV*, .

[14] 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) .

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

[16] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel et al., *Scikit-learn: Machine learning in Python*, *Journal of Machine Learning Research* **12** (2011) 2825–2830.

[17] Q. Wang and J. JaJa, "From maxout to channel-out: Encoding information on sparse pathways."

[18] P. Baldi, K. Cranmer, T. Faucett, P. Sadowski and D. Whiteson, *Parameterized machine learning for high-energy physics*, 1601.07913 [hep-ex, physics:hep-ph].

[19] T. Chen and C. Guestrin, *XGBoost: A scalable tree boosting system*, in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, (New York, NY, USA), pp. 785–794, ACM, 2016. DOI.

[20] T. Chen and C. Guestrin, *XGBoost*, in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, aug, 2016. DOI.

[21] R. Brun and F. Rademakers, *Root - an object oriented data analysis framework*, in *AIHENP'96 Workshop, Lausane*, vol. 389, pp. 81–86, 1996.

[22] The Root team, "Rdataframe."

[23] T. pandas development team, *pandas-dev/pandas: Pandas*, Feb., 2020. 10.5281/zenodo.3509134.

[24] The Root team, "Tlorentzvector."

[25] 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*, .

[26] 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 TeV*, *The European Physical Journal C* **79** (aug, 2019) .

[27] P. Liashchynskyi and P. Liashchynskyi, *Grid search, random search, genetic algorithm: A big comparison for nas*, 2019. 10.48550/ARXIV.1912.06059.

[28] A. Gholamy, V. Kreinovich and O. Kosheleva, *Why 70/30 or 80/20 relation between training and testing sets: A pedagogical explanation*, .

[29] ATLAS collaboration, *Performance of the Reconstruction and Identification of Hadronic Tau*

Decays with ATLAS, tech. rep., CERN, Geneva, 2011.

[30] ATLAS collaboration, *Performance of Top Quark and W Boson Tagging in Run 2 with ATLAS*, tech. rep., CERN, Geneva, 2017.