UNIVERSITY
OF OSLO

Master's thesis

Model independent search for Dark Matter using Machine Learning

In final states with dileptons and Missing Transverse Energy with the ATLAS detector at the LHC

Ruben Guevara

Physics: Nuclear and Particle Physics
60 ECTS study points

Department of Physics
Faculty of Mathematics and Natural Sciences

Spring 2023

Ruben Guevara

Model independent search for Dark Matter using Machine Learning

In final states with dileptons and Missing Transverse Energy with the ATLAS detector at the LHC

Supervisors:
Professor Farid Ould-Saada
Dr. Eirik Gramstad

Acknowledgements

Thank you everybody<3<3
Most importantly I want to thank the coffee machine at the section for High Energy Particle Phyiscs for the incredible support in times of need, as the coffee was free.

Abstract

Something something

Contents

List of Figures

List of Tables

List of Acronyms

SM Standard Model

DM Dark Matter

LHC Large Hadron Collider

MET Missing Transverse Energy

ML Machine Learning

NN Neural Network

BDT Bosted Decision Tree

Part I

The theory behind modern particle physics / Background

Chapter 1

The Standard Model of Particle Physics

1.1 Quantum Electrodynamics

1.2 Quantum Chromodynamics

1.3 Electroweak theory and the Brout-Englert-Higgs Mechanism

1.4 Adding it all up

The standard model of particle physics is the combination of three gauge groups. The group explaining electromagnetism $U(1)$, the group describing the weak force $SU(2)_L$ and the group describing the strong force $SU(3)_C$. When combibing all these groups we get spontaneous symmmetry breaking resulting in the Brout-Englert-Higgs Mechanism. The whole lagrangian is of the form

$$U(1)_Y \otimes SU(2)_L \otimes SU(3)_C \Rightarrow$$

$$\mathcal{L}_{SM} = -\frac{1}{4}F_{\mu\nu}F^{\mu\nu} + i\overline{\Psi}\not{D}\Psi + \psi_i y_{ij}\psi_j\phi + h.c. + |D_\mu\phi|^2 - V(\phi) \tag{1.1}$$

where

$$V(\phi) = -\mu^2\phi^*\phi + \frac{\lambda}{2}(\phi^*\phi)^2$$

is the Higgs potential.

All of this is great at explaining what we know so far

Chapter 2

Dark Matter

2.1 Observations of existence

Proof? Here [1, 2]

2.1.1 Cosmology

2.1.2 WIMP

2.2 Beyond Standard Model candidates

2.2.1 New gauge boson

Z' baby [3, 4, 5]

2.2.2 Supersimmetree

Chapter 3

Production, Detection and Analysis

Now that we have established the necessary theoretical groundwork of particle physics in Chapter 1 and 2, it's time to explore how this knowledge can be applied. This leads us to ask important questions such as, how can we measure what we have learned? How do we put it into practice? Most importantly, how can we use this understanding to uncover new discoveries?

will be added later

To answer these questions, we have divided this chapter into three sections, each of which will focus on a different aspect of experimental particle physics. The first section will delve into particle production, followed by an examination of particle detection with the ATLAS detector at the LHC, and finally, we will explore the intricacies of data analysis in particle physics.

By exploring these areas, we hope to provide a comprehensive understanding of the theoretical underpinnings of particle physics, while also highlighting the practical applications of this knowledge.

I am struggling a bit to motivate the different parts of the thesis! So if you have any extra things I can add here feel free to do so!

3.1 Particle production

As we have already seen the shape of the SM we are now ready to dive into the subject of how we can produce the particles that we wish to detect. In this chapter we will start from the basic kinematics of particles and then move to more complex variables that will be of use when analysing data from detectors. The material for the first section is based on Thomson book Modern Particle Physics [6], Jacksons "Kinematics" [7] and Vadlas PhD. thesis [8].

3.1.1 Particle kinematics

Is this comment unnecesary?

As proved by Einstein, everything in spacetime can be described by four-vectors. For the purposes of particle physics, where we are mainly interested in the motion of particles, we will look at the four-momentum. Instead of using general variables as Einstein did, we will describe the particles using the four-momentum in terms of the geometry of the decetors, that means we will use the polar angle, θ, and the azimuthal angle, ϕ, such that we have

$$p^\mu = (E, p_x, p_y, p_z) \xrightarrow{Lab} (E, p_T \cos\phi, p_T \sin\phi, |\mathbf{p}| \cos\theta) \tag{3.1}$$

where p_T is the *transverse momentum* expressed as

$$p_T \equiv \sqrt{p_x^2 + p_y^2} = |\mathbf{p}| \sin\theta \tag{3.2}$$

The energy and momentum can be expressed in relativisic quantities, $E = \gamma\beta$ and $\mathbf{p} = \gamma m\boldsymbol{\beta}$, where $\gamma = 1/\sqrt{1-\beta^2}$ and $\boldsymbol{\beta} = \mathbf{v}/c$[1] where m is the mass of the particle and c is the speed of light in vacuum. By contracting[2] two four-momentum we get the important Lorentz invariant square of the *invariant mass*

$$m^2 = p_\mu p^\mu = E^2 - |\mathbf{p}|^2$$

which can be generalized for a system containing n particles as

$$m^2 = p_\mu p^\mu = \left(\sum_{i=1}^{n} E_i\right)^2 - \left(\sum_{i=1}^{n} \mathbf{p}_i\right)^2 \tag{3.3}$$

As this thesis will focus on a dilepton (and missing transverse energy) final state, which is of the type $2 \to 2$ $(+MET)$ then the invariant mass of the two leptons in the final state will be of interest, we will denote this as m_{ll}. From this we can also get another interesting variable, the *transverse energy*. This follows diretly from the same equation

$$E_T = \sqrt{m^2 + p_T^2} \tag{3.4}$$

The invariant mass is what we measure in the final state only. But as we are going to use data[3] from the LHC, from which the initial state is controlled, it will be of interest to see what the total energy and momentum of the two protons are. The term for this is called the *centre-of-mass* energy, $\sqrt{s}$, where s is defined by the same formula in Eq. (3.3), with the difference beeing that we look at the initial particles. For this thesis we will look at data and simmulations of Run II from the LHC, which had $\sqrt{s} = 13$ TeV.

do I need sources for the neutrino claim?

As this thesis aims to search for DM, which we know does not interact with matter in the same way as neutrinos , meaning it leaves no signal in detectors. As we know both the centre-of-mass energy, $\sqrt{s}$, and the invariant mass of all particles in the final state, Eq. (3.3). Then the presence of the non-interacting particles can often be inferred from the presence of *missing transverse energy*[4] (MET), which is defined by

$$E_T^{miss} = \mathbf{p}_{miss} \equiv -\sum_i \mathbf{p}_{T,i} \tag{3.5}$$

where the sum extends over the measured momenta of all the observed particles in an event. From this formula, if all particles produced in the collision have been detected, then this sum should be zero. Meaning that signifiant MET is therefore indicative of the presence of an undetected particle.

[1] As this is a particle physics thesis I will convert to Natural Units where we set $c = 1$
[2] Using the particle physicists convention of the minkowsky metric tensor $\eta_{\mu\nu}$, (+, -, -, -)
[3] And mostly simmulations mimicking the ATLAS detector
[4] Also called *missing momentum*

Another useful kinematic variable is the *hadronic activity* which is the scalar sum of the transverse momentum of all jets in an event, defined as

$$H_T = \sum_{i \in \{jets\}} ||\mathbf{p}_{T,i}|| \tag{3.6}$$

this gives a measurement of the hadronic energy scale of an event. Another handy trick comes from the reailization that the centre-of-mass frame is between the hadrons, where the total momentum is given as a function of the energy of the hadron. This means that the final state particles are boosted along the beam axis. With this realization we can now introduce a Lorentz invariant[5] kinematic property known as the *rapidity, y* used to express the lepton angles

$$y \equiv \frac{1}{2} \ln \left(\frac{E + p_z}{E - p_z} \right) \tag{3.7}$$

We can use that $p_Z = E \cos\theta$ in the high-energy limit as the mass is negligible. In this limit we can use the *pseudorapidity*, η, defined by

$$\eta \equiv -\ln \left(\tan \frac{\theta}{2} \right) \tag{3.8}$$

The pseudorapidity is an interesting variable as it can tell us how close to the beam the final state particles are, where the higher $|\eta|$ means closer to the beam. This variable can also be negative, meaning backwards scattering. From this we can define a new variable which will come handy with particle identification, which is called the *R-cone*, that defines a cirlce in (η, ϕ)-space surrounding the object of interest. It is defined as

$$\Delta R = \sqrt{(\Delta\eta)^2 + (\Delta\phi)^2} \tag{3.9}$$

Another interesting variable is the *transverse mass*, defined as

$$m_T^2 = m^2 + p_T^2 \tag{3.10}$$

where m^2 is the invraiant mass defined in Eq. (3.3). What is interesting with this variable is that it is the equivalent of the invariant mass equation, that takes into account invisible particles! We can take this further by looking at a supersymmetric version of the transverse mass, which calculates a transverse mass for two leptons by distributing the total p_T^{miss} among the two systems, and minimizing the maximum of the two transverse masses by varying the distribution of the $p_T^{miss}-$vector in terms of the size of p_T. This is called the *stransverse mass* and is defined by

$$m_{T2}^2(\chi) = \min_{\not{\mathbf{q}}_T^{(1)} + \not{\mathbf{q}}_T^{(2)} = \not{\mathbf{p}}_T} \left[\max \left\{ m_T^2 \left(\mathbf{p}_T^{\ell_1}, \not{\mathbf{q}}_T^{(1)}; \chi \right), m_T^2 \left(\mathbf{p}_T^{\ell_2}, \not{\mathbf{q}}_T^{(2)}; \chi \right) \right\} \right] \tag{3.11}$$

where $\not{\mathbf{q}}_T$ are "dummy 2-vectors", χ is a free parameter used to "guess" the mass of the invisible particle, and $m_T^2(\mathbf{p}_T, \mathbf{q}_T)$ is an application of Eq. (3.10) using two particles:

$$m_T^2(\mathbf{p}_T, \mathbf{q}_T) = 2(p_T q_T - \mathbf{p}_T \cdot \mathbf{q}_T)$$

For a more detailed explanation and interpretation of the stransverse mass I refer the reader to the paper by Barr et.al. [9]. Even though the stransverse mass was made with neutralinos in mind, it can still be used to calculate SM processes. For example, if we want to reduce WW background events, we can first recall that each boson can decay as $W \to l + \nu_l$ with the W mass as an endpoint. Meaning that we can use m_{T2} to reduce the WW events in a dilepton final state by requiring that $m_{T2} > m_W$.

[5] Under boosts along the beam axis

3.1.2 Proton-proton collisions

With all the kinematics out of the way the question of how the particles are produces still remains. The answer to that is protons. The way we produce elementary particles to study is by colliding two protons togheter. The reason as to why this works is because protons are also made of elementary particles, two *up* and one *down* to be specific. Because of this it is not hard to realize that the Feynman rules aquired from the SM (Chapter 1) also apply here. In this subsection we will study how different effects of $pp-$collisions affect the cross section, and therefore the expected number of events to occur, from a kinematical point of view.

Parton Distribution Functions

Altough the proton is made up for two up quarks and one down quark, called the *valnce quarks*. The proton aslo constits of gluons and other quarks, called partons. These partons become imporant in deep innelastic scatterings, where the proton breaks apart due the high energies in the collisions. As we accelerate the protons before colloding them, we also accelerate the quarks and gluons inside of it, each of the quarks carry a momemntum fraction x, called the Bjorken x. We can then calcluate the invariant mass of two colliding partons q_1 and q_2, with the momentum fraction x_1 and x_2, from the protons momentum p_1 and p_2 respectively as

$$m^2 = x_1 x_2 s$$

where s is the centre-of-mass energy squared of the $pp-$system. The valence quarks in the proton do not only interact with the other valence quarks in the other proton, but they might also emit gluons which decay into $q\bar{q}-$pairs, making a "sea" of gluons and quarks around the valence quarks. The momentum of the partons inside of the proton are dependent on the momentum transfer Q^2 and is represented by an experimentally determined momentum distribution, known as the *parton distribution function* (PDF) [10] $f(x, Q^2)$. **Question: Should I say more about PDFs than this? I feel like it is not relevant for my thesis, but is still good to know.** In other words, the PDFs give the probability of a parton to collide with the momentum fraction x. The shape and form the PDFs play an imporant role of estimating the processes that occur after the proton collisions, and therefore are crucial when simulating events using Monte Carlo [11]. If we take as an example the process $pp \to l^+l^- + X$ where X denotes any hadrons formed by the remaining quarks. Figure 3.1 showcases the process.

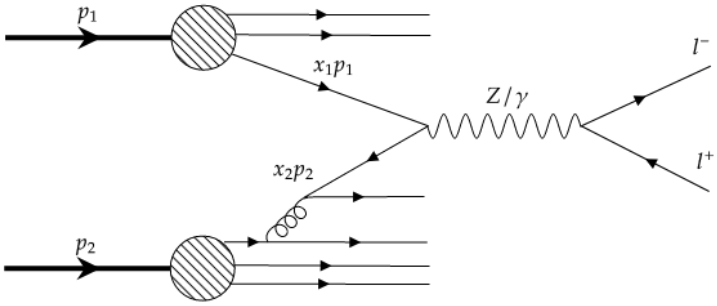


Figure 3.1: Feynmann diagram depicting the $pp \to l^+l^- + X$ process where X denotes any hadrons formed by the remaining quarks not taking part in the $q\bar{q}-$collisionc carrying x_1 and x_2 momentum from the protons p_1 and p_2 respectively

The cross section for the process is

$$\sigma\left(p(p_1)p(p_2) \to l^+l^- + X\right) =$$

$$\int_0^1 \int_0^1 \sum_f f_f(x_1) f_{\bar{f}}(x_2) \cdot \sigma\left(q_f(x_1p_1) + \bar{q}_f(x_2p_2) \to l^+l^-\right) dx_2 dx_1 \tag{3.12}$$

where $\sigma\left(q_f(x_1p_1) + \bar{q}_f(x_2p_2) \to l^+l^-\right)$ is calculated using the rules explained in Chapter 1, and f are the PDFs. As we can see this can drastically change the cross section, and with that the probability, for an event we want to study.

Breit-Wigner resonane

Another important aspect when looking at particle collisions is the *Breit-Wigner resonance*. All unstable particles have a decay rate, Γ, (given as inverse of the lifetime τ) which is present at the wavefunction, $\Psi \propto e^{-i(m-i\Gamma/2)}$. This decay rate also becomes apparent when the unstable particle is the propagator of an event we are studying

$$\sigma \propto \frac{1}{(s-m^2)^2 - m^2\Gamma^2} \tag{3.13}$$

From this we can see that as the square of the centre-of-mass energy s, approaches the unstable particles mass m, there will be a resonance at the invariant mass of the unstable particle, this is called the Breit-Wigner resonance. It is because of resonances like this that we can identify particles such as the Z boson. This resonance will also be present when studying some of the BSM DM models in this thesis.
should I cite the original paper or Z discovery?

Expected events

The most important value we need to know when studying $pp-$collisions is the number of interactions taking place, this is defined as

$$N = \sigma \int \mathcal{L}(t)dt, \qquad \text{where} \quad \mathcal{L} = f\frac{n_1 n_2}{\sigma_x \sigma_y} \tag{3.14}$$

where σ is the cross section of the interaction as expressed in Eq ... while also taking into account the effects from Eq. (3.12) and Eq. (3.13). The last three symbols come from accelerator kinematics where $\sigma_{x,y}$ denotes the beam size, f is the frequency of bunch crossings, and $n_{1,2}$ is the number of particles in bunches.

ADD THIS IN CHAPTER 1!

3.2 The ATLAS detector

We have so far on this chapter discussed how particles are produced. But we so far not yet explained how we actually detect them, arguably the most important matter in the field of high energy particle physics! This section of the chapter is just about that, and we will explain how the detection happens in A Toroidal LHC ApparatuS, or more commonly know as the ATLAS detector. Figure 3.2 showcases the detector and its size. The information of this section is largely based on the original ATLAS Collaboration introduction paper presented to CERN [12]. (and Knuts thesis [8] that described it in a more undertandable way).

from my understanding, correct me if wrong!

The ATLAS detector is a general multi-purpose[6] detector located at the LHC and covers nearly the entire solid angle around the collision point, as described with the $(\eta, \phi)-$coordinates in Section 3.1.1. The ATLAS detector consists of four main subdetectors; (i) an inner tracking detector (ID), an (ii) electromgetic calorimeter (ECAL), and a (iii) hadronic calorimeter (HCAL), and lastly (iv) a muon spectrometer (MS). Figure 3.3 visualises the four $(i) - (iv)$ main sub-detectors, along with how the different particle types interact with each layer. A brief description of each layer is explained in this section.

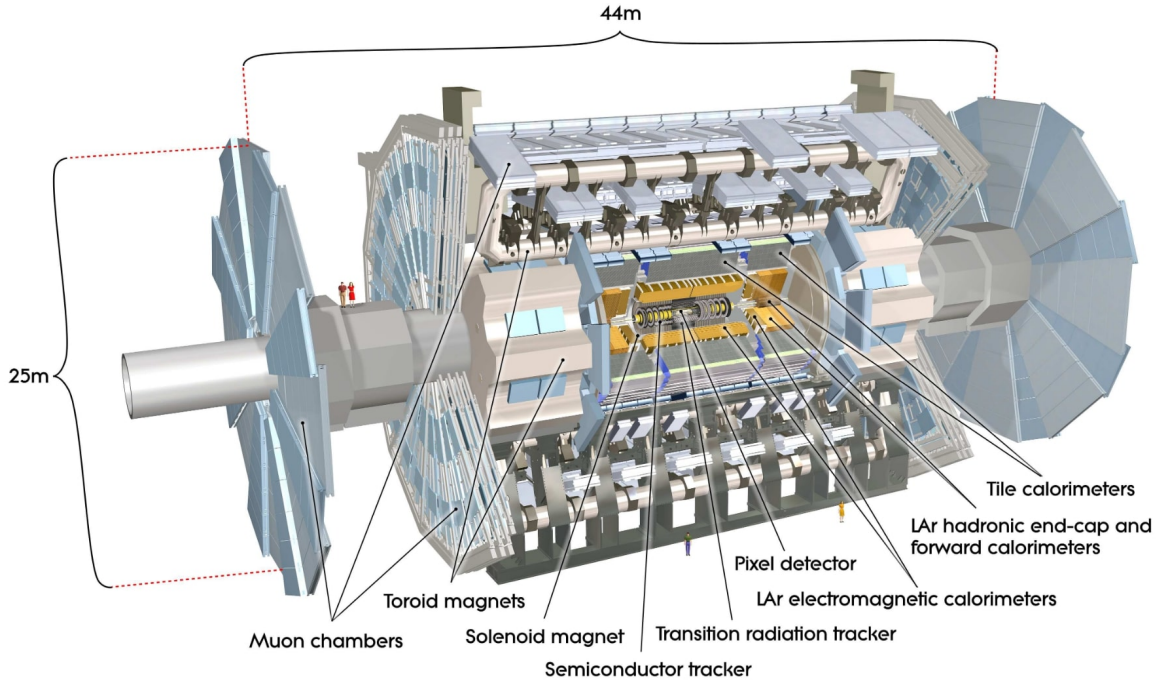


Figure 3.2: Cut-away view of the ATLAS detector, image taken from Ref. [12]

3.2.1 Inner detector

The inner-detector (ID) system is immersed in a 2T axial magnetic field and provides charged-particle tracking in the range $|\eta| < 2.5$. The ID provides the first measurements of the momentum and identification of electrically charged particles, as these can be determined by the curvature of their reconstructed tracks. The ID is made of three independent systems; the pixel detector, the semiconductor tracker (SCT) and the transition radiation tracker (TRT).

The pixel detector is made up of 80 million sillicon pixel sensors, each of size $50 \times 400 \mu\text{m}^2$, and spread over multiple layers. Outside the pixel layers are the SCTs, which consist of sillicon microstrips trackers, also placed on multiple layers. The SCT covers the pseudorapidity region $|\eta| < 2.5$. Furthest away from the interaction point lies the TRTs, they consist of 4mm in diameter straw tubes, which enables track-following up to $|\eta| = 2.0$

[6] Probing $pp-$ and $AA-$ (heavy ions) collisions.

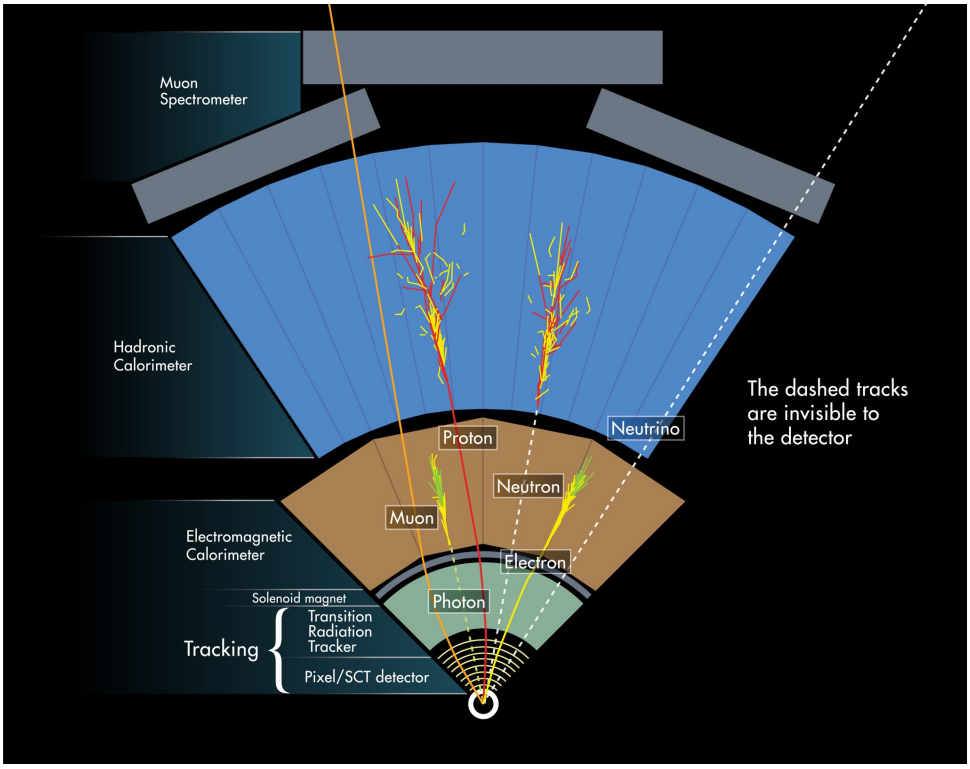


Figure 3.3: Illustration of the ATLAS detector layers, image taken from Ref. [13]

3.2.2 The calorimeters

The ATLAS detector has two types of calorimeters, the *electromagnetic calorimeter* (ECAL) and the *hadronic calorimeter* (HCAL), both designed to fully stop certain types of particles. Both calorimeters covers a pseudorapidity range $|\eta| < 4.9$. The ECAL is immediately surroinding the inner detector and is divided into a barrel part ($|\eta| < 1.475$) and two end-cap components ($1.375 < |\eta| < 3.2$). The ECAL consists of absorbing lead plates, with liquid Argon (LAr) in between. The thickness of the calorimeter is made to fully measure the shower of photons and electron/positrons. The muons will only lose a small fraction of their energy as they have longer interaction lenghts with lead.

The HCAL is immediately surroinding the ECAL on all sides and consists of two types of detectors. In the barrel ($|\eta| < 1.0$) and the extended barrel regions ($0.8 < |\eta| < 1.7$), the HCAL is made of steel plates with plastic scintillator tiles as active material. While on the end-cap regions ($1.5 < |\eta| < 3.2$) there are hadronic LAr detectors, with absorbing copper plates as active material; in the forward region ($3.1 < |\eta| < 4.9$) a combination of copper and tungsted plates are used as active material. The active materials are chosen to maximze the ineraction cross-section with hadrons, such as neutrons, protons and pions. The depth of the HCALs is also designed to fully stop the particles and their showers in order to measure their total energies. Hadrons are efficiently stopped at the HCALs, meaning that only muons and invisible particles, such as neutrinos and potentially dark matter, leave the HCAL.

3.2.3 Muon spectrometer

The outermost layer of the ATLAS detector is the *muon spectometer* (MS), dedicated to the measurement of the muons momenta. The MS, similar to the ID, consists of multiple layers of detector material, and is immersed in a strong magnetic field to bend the trajectories of the charged muons. The MS is made of four different types of detector component: (*i*) Monitored Drift Tubes (MDTs) on the barrel, (*ii*) Cathode Strip Chambers (CSCs) dealing with the events closer to the beam line in the end cap, (*iii*) Resistive Plate Chambers (RPCs) in the barrel and (*iv*) Thin Gap Chambers (TGCs) in the end caps. The MDTs and CSCs are used for tracking while the RPCs and TGCs are used for tiggers. The tracking is provided for pseudorapidities up to $|\eta| < 2.7$, and the trigger system only extends to $|\eta| < 2.4$.

3.2.4 Summary of ATLAS

The main four parts of the ATLAS detector; *inner-detector* (ID), *electromagnetic calorimeter* (ECAL), *hadronic calorimeter* (HCAL), and the *muon spectometer* (MS), have briefly been explained. For an easier understanding on the different parts I refer the reader to Figure 3.3 which shows visually how different particles interact with the detector. We can start by noting that there are only three types of particles that interact with the ID, muons, protons and electrons (there might also be other ellectrically charged hadrons that interact withg the ID). From the ID pixel and SCT detectors we can get information about ϕ, η, p_T and from the TRT we can get to know the *the charge* of the particles (depending on how the magnet bends their trajectories)[7]. Going to the next layer we see that in the ECAL the electron is completely stopped, such that we can know its energy, E. We can also note that the photons will interact with matter here and also be completely stopped. It is worth noting that in just the two first detector parts we have all the information we need to calculate interesting kinematic variables for the electron and photons. Going further into the detector, we see that the HCAL completely stops the proton and that now the detector interacts with the neutrons, giving us information about their energies (and location for neutrons). Lastly we see that the muon flyes through all of this and finally hits the MS where we can get information about its energy. ***Note/Question to Farid/Eirik: Why does it not stop?***.

The noteworthy particle in Figure 3.3, which was not mentioned in the summary above either, is the *neutrino*. The reason for this is because the neutrino, just like DM, is an *invisible particle*, meaning that it does not interact with matter. We can still reconstruct information about the neutrino and DM howevever, we do this as explained in Section 3.1.1 with variables such as the MET, Eq. (3.5), as we know the centre-of-mass energy that went into the collision.

Another thing to add is that the explanation of the detector is brief in this thesis. The reality is much more complicated than this as there are, i.e. *triggers* (which might be excluding DM events) choosing which events are "important enough to record" as the *hardware and data storage* is a big problem considering the magnitude of events that happen each second. There are as well as algorithmic precedures to calculate the variables aforementioned from the detector signal which both consume energy and time. As this is not the focus of my thesis I will refer the reader to further literature explaining the real hardships when collecting data at the ATLAS detector and other LHC projects. [14]
Question: Is it okay for me to do this and move on? Or should I write about this?

[7] So as we know that the proton is positively charged, then we know that the muon and electron in Figure 3.3 have negative charge

3.3 Data analysis

The time has come to explore how we can search for new physics phenomena, now that it has been established how particles are produced and how we detect them. In this section we will take into account the classic way of searching for new phyics which is called the *cut and count method*, but there are other methods to search for new physics, such as Machine Learning (ML) which is the method pursued in this thesis. There might be other methods, such as Quantum ML, but as of today we are still in an early stage of the technology [15]. To give a short description of the cut and count method, it makes kinematial *cuts* such that we isolate signal from background. The way the signal and background are made is by Monte Carlo (MC) simulations, such that we can more easily identify SM background from DM and then compare our results with real data. As it wouldn't be a real expreimental physics discovery without making a statistical analysis of the results we will also explain how we utilize this tool. To guide us through this process I will use $ZZ^{(*)}$ channel in the discovery of the Higgs Boson in 2012 [16] as an example of the sucess of this methods.

can I include this?

3.3.1 Cut and count

The cut and count method is what is curently the standard method of doing data analysis with CERN related research. As the name implies, the cut and count method works by making cuts on kinematical variables and afterwards counting how many events are left. The goal of using this method is to make cuts such that we remove as many background events as possible while also keeping as many signal events as possible. For example, if we were to study a new physics model with a new light vector boson behaving similarly to the Z boson, but with a higher mass, then a good kinematical cut to remove many background processes would be to require that $m_{ll} > 100$ GeV, as this would remove the majority of Z-resonance from the final state, making it easier to "find" the new physics model.

To more thouroughly explore the cut and count method we can look at the Higgs discovery, in particular the $H \to ZZ^{(*)} \to 4l$[8] channel. In section 4.1 of the original article [16] there is something called *event selection*. The event selection are the kinematical cuts used. The kinematical cuts used in the article, aside from the *standard selection critera* (see Chapter 5.3), were:

- Single-lepton or dilepton triggers
- Four leptons on final state with $p_T > 20, 15, 10, 7$ GeV, in the order of most energetic to least
- Higgs-boson candidates are formed by selecting two same flavour opposite charge lepton pairs

The first "cut" is to make sure that we have leptons on both the data and simulations. The second cut is that there needs to be four leptons, with different p_T cuts to remove background while keeping as much signal as possible. And lastly we want to have two lepton pairs of the same-flavour with opposite-charge, this is to make sure that the leptons that we observe actually decay from Z-bosons. What now remains is to explore the "count" part of this method. When counting the events that pass the event selection one usually counts the background events that pass, the data points that pass, and also the signal events that pass. For the 2012 Higgs discovery, the Higgs channel with a Higgs mass of 125 GeV was used as signal. The results of the 2012 discovery is shown Figure 3.4.

Altought this section made the process look simple, it was through the effort of many scientist working togheter that made this happen. The great computational power needed to *correctly* simulate events and reconstruct objects from detector signal was, and still is a big challenge. Not to mention the state-of-the-art technology to be able to be to both accelerate the protons to an energy high enough to "create" new physics, and to actually be able to detect it. But this alone was not enough to claim the discovery, as we're physicits we need to be completely sure that what we show is true. We do this with statistics which is the next section.

[8] where l is for lepton, but only means $e^{\pm}$ or $\mu^{\pm}$. *question: should i explain why not τ?*

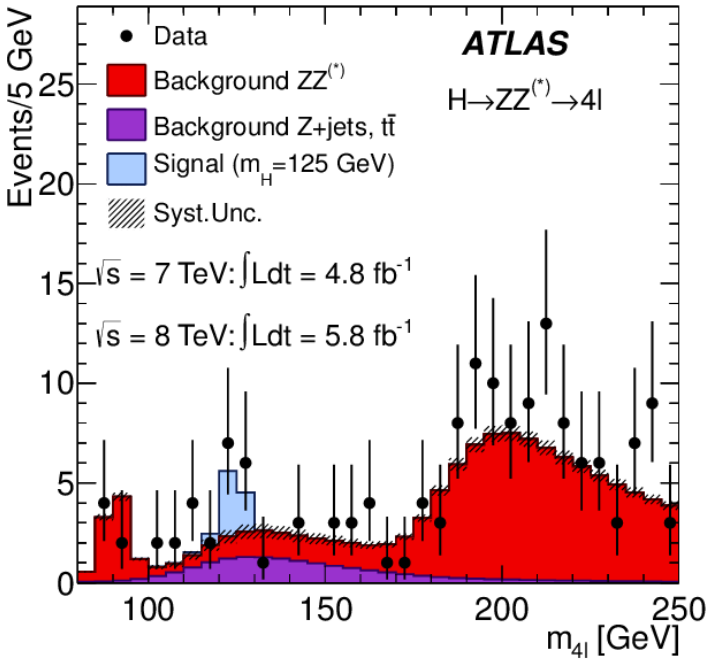


Figure 3.4: The Higgs discovery on the $ZZ^{(*)}$ channel, image taken from Ref. [16]

3.3.2 Statistical analysis

To make any sort of claim in modern physics we should be absolutely certain that what we are claiming is true, as just making the cuts and isolating a signal to background is not enough. To be specific we need to be *at least* 5σ *sure* to claim any new discoveries. But as not everything in experimantal high energy particle physics is a new discovery, but is in fact for the most part *exclusions*. Then we need to explain what an exlusion is, and how to make them.

This thesis is following Bugges lecture notes on statistics [**how do I cite his powerpoint from FYS5555**] and the example set by the ATLAS Collaboration article "Search for new particles in events with one lepton and missing transverse momentum in pp collisions at $\sqrt{s} = 8$ TeV with the ATLAS detector" [17], where a Bayesian analysis is performed to set limits on the studied processes. Using the signal+backround hypotesis, the expected number of events in each channel of a process we are studying is

$$N_{\text{exp}} = \varepsilon_{\text{sig}} L_{\text{int}} \sigma B + N_{\text{bkg}}$$

where L_{int} is the integrated luminosity, ε_{sig} is the signal selection efficiency defined as the fraction of signal events that satisfy the event selection criteria, N_{bkg} is the expected number of background events, and σB is the cross-section times branching ratio of the process. Using Poisson statistics, the likelihood to observe N_{obs} events is

$$\mathcal{L}(N_{\text{obs}}|\sigma B) = \frac{(N_{\text{exp}})^{N_{\text{obs}}} e^{-N_{\text{exp}}}}{N_{\text{obs}}!} \tag{3.15}$$

We include uncertainties by introducing nuisance parameters θ_i, each with a probability density function $g_i(\theta_i)$, and integrating the product of the Poisson likelihood with the probability density function. The integrated likelihood is

$$\mathcal{L}_B(N_{\text{obs}}|\sigma B) = \int \mathcal{L}(N_{\text{obs}}|\sigma B) \prod g_i(\theta_i) d\theta_i \tag{3.16}$$

where a log-normal distribution is used for the $g_i(\theta_i)$. The nuisance parameters are taken to be: $L_{\text{int}}, \varepsilon_{\text{sig}}$ and N_{bkg}, with the appropriate correlation accounted for between the first and the third parameters. The measurements of the two decay channels (muon or electron final state for $H \to lll'l'$) are combined assuming the same branching fraction for each, thus Eq. 3.16 remains valid with the Poisson likelihood replaced by the product of the Poisson likelihoods for the two channels. The integrated luminosities for the electron and muon channels are fully correlated. We can further use Bayes' theorem which gives the posterior probability that the signal has signal strength σB:

$$P_{\text{post}}(\sigma B|N_{\text{obs}}) = N\mathcal{L}_B(N_{\text{obs}}|\sigma B) P_{\text{prior}}(\sigma B) \tag{3.17}$$

where $P_{\text{prior}}(\sigma B)$ is the assumed prior probability, here chosen to be flat in σB, for $\sigma B > 0$. The constant factor N normalises the total probability to one. The posterior probability is evaluated for each mass and decay channel as well as for their combination, and then used to set a limit on σB.

As we can see, the inputs for the evaluation of $\mathcal{L}_B$ (and P_{post}) are ε_{sig}, L_{int}, N_{bkg} and N_{obs} and the uncertainties of the first three. The uncertainties for these should account for experimental and theoretical systematic effects as well as the statistics of the simulated samples. For this thesis the systematic uncertainties will not be calculated, but will rather be asumed to be $\pm$ 30% of the background, as this is roughly the standard value of ATLAS papers.

To make exclusions we can use Eq. (3.17) to establish a *confidence limit* (CL). CLs are deined as the probablity to observe the number of events observed in an experiment, N_{obs}, or *less* given singal+background. We usually define a signal+background hypothesis to be excluded whene $\text{CL}_{s+b} < 5\%$. Meaning a 95% CL. Such that the probability to falsely exclude an existing signal(+background) is 5%. The standard practice in particle physics when using CL is to *set limits* on theoretical models, rahter than exclude them.

On the other side, to claim any discovery in particle physics we need to know the *significance* of any statistical fluctiation. Before getting to the significance we can discuss the *p-value*, defined as the probability to observe the number of events observed in the experiment, n_{obs}, or *more* given only background

$$p = P(N \geq N_{\text{obs}}|\lambda = N_{\text{bkg}}) = \sum_{k=N_{\text{obs}}}^{\infty} \mathcal{L}(k|N_{\text{bkg}}) \tag{3.18}$$

The smaller the $p-$value, the less compatible an observation is with the background only hypothesis, meaning more likely to be a discovery! From this we can find the significance Z by

$$p = \int_{-\infty}^{-Z} \frac{e^{-x^2/2}}{\sqrt{2\pi}} dx$$

As mentioned in the start of this subsection, a discovery in particle physics is defined to be at least a $Z = 5\sigma$ deviation from the background hypothesis, meaning that we would have a $p-$value of $p \leq 2.87 \times 10^{-7}$. In other words, with a 5σ deviation, the probabilty to falsely discover something is at worst **one in roughly 3.5 million**.

As the significance is an interesting quantity we can give it its own definition. In this thesis we will use the low statistics formula for the significance, as this is the most general one. We can either define the significance as the *observed significance* by

$$Z = \sqrt{2\left[N_{\text{obs}} \ln \frac{N_{\text{obs}}}{N_{\text{bkg}}} - N_{\text{obs}} + N_{\text{bkg}}\right]} \tag{3.19}$$

or as the *expected significance* by changing $N_{\text{obs}} \to N_{\text{sig}} + N_{\text{bkg}}$, where N_{sig} is the number of signal events

$$Z = \sqrt{2\left[(N_{\text{sig}} + N_{\text{bkg}}) \ln\left(1 + \frac{N_{\text{sig}}}{N_{\text{bkg}}}\right) - N_{\text{sig}}\right]} \tag{3.20}$$

However, Eq. (3.18) did not include any nuisance parameters, it used Eq. (3.15) instead of Eq. (3.16). We want to express the significance with uncertainties. From "Discovery sensitivity for a counting experiment with background uncertainty" from Glen Cowan [18], we can use Eq. (17) on his paper that reads

$$Z = \left[-2\left(N_{\text{obs}} \ln\left[\frac{N_{\text{obs}} + m}{(1+\tau)N_{\text{obs}}}\right] + m \ln\left[\frac{\tau(N_{\text{obs}} + m)}{(1+\tau)m}\right]\right)\right]^{1/2}$$

where $m = \tau N_{\text{bkg}}$ and Eq. (19) that reads

$$\tau = \frac{N_{\text{bkg}}}{\sigma_{\text{bkg}}^2}$$

where σ_{bkg} is the uncertainty of the background. This gives us

$$Z = \sqrt{-2\left(N_{\text{obs}} \ln\left[\frac{N_{\text{obs}} + \frac{N_{\text{bkg}}^2}{\sigma_{\text{bkg}}^2}}{(1 + \frac{N_{\text{bkg}}}{\sigma_{\text{bkg}}^2})N_{\text{obs}}}\right] + \frac{N_{\text{bkg}}^2}{\sigma_{\text{bkg}}^2} \ln\left[\frac{\frac{N_{\text{bkg}}}{\sigma_{\text{bkg}}^2}(N_{\text{obs}} + \frac{N_{\text{bkg}}^2}{\sigma_{\text{bkg}}^2})}{(1 + \frac{N_{\text{bkg}}}{\sigma_{\text{bkg}}^2})\frac{N_{\text{bkg}}^2}{\sigma_{\text{bkg}}^2}}\right]\right)} \tag{3.21}$$

Which makes for a better estimate of the significance one has in reality. ***Should I add plots showing CLs and significance?***

3.4 Summary

In this chapter we have studied how one can calculate the number of expected events from a $pp-$collision can lead to the discovery of new particles. Using special relativity as a tool, we can express the four-momentum of the particles with detector-coordinates, $p^\mu = (E, p_T \cos\phi, p_T \sin\phi, |p| \cos\theta)$. From this four-momentum vector we can thereafter calculate interesting kinematic variables such as the invariant mass m_{ll}, missing transverse energy E_T^{miss}, stransverse momentum, m_{T2}, among others. Using the advanced ATLAS detector the four-momentum can be recorded from the accelerated protons at the LHC. With the recorded data and MC simulations taking into account the experimental features such as the PDFs and Breit-Wigner resonance, as well as the ATLAS kinematics, we can compare how the simmulated events fare with the data recorded. By playing around with the kinematical variables of the particles and making cuts to isolate new phyics signal we can see if there is a discrepancy between the data recorded and the SM bacgkround. After creating this signal region with the cut and count method we conduct a statistcal analysis to see how the new theory/observed data deviates from our current understanding of physics.

This state-of-the-art method is what currently is being used at CERN and has lead to a great advancement in the field. However with the rise of new technolgies, such as *machine learning*, which excel at classification tasks, a door has been opened to try new methods. In this thesis we will use ML to hopefully create a better and more general signal region than what the current cut and coount method does. Before describing how, we will explain what machine learning is, this is the subject of the next chapter.

I don't know if this summary is too dense, or if the opening to ML is too superficial. Feel free to comment as harshly as you want here :-)

Chapter 4

Machine Learning

Machine Learning (ML) has emerged as a powerful tool for analyzing complex datasets and making accurate predictions. Its applications span across various fields, from natural language processing to image recognition, and it has been used successfully to solve a range of problems.

The main approach of this thesis will be to use ML as its popular rise has proven to be effective at binary classification tasks [19, 20] in High Energy Particle Physics. For our purposes it will be a powerful tool to attempt to classify events as SM background or as DM signal.

To give a a short description of the essence of ML we can start by considering a general parameter $\boldsymbol{\beta} = \{\beta_1, \beta_2, \cdots, \beta_n\}$ for a n-dimensional problem, which for our purposes can be seen as what are called *weights* and *biases* ($\boldsymbol{\beta} = \{w, b\}$), the goal is to choose these parameters $\boldsymbol{\beta}$ such that we minimize a cost (also called loss) function $C(\boldsymbol{\beta})$ with respect to a set of data points given by a matrix $\mathbf{X}$. This matrix $\mathbf{X}$ will be our dataset containing n *features* for each event m, and is of the following form

$$\mathbf{X} = \begin{pmatrix} x_{11} & x_{21} & x_{31} & \cdots & x_{n1} \\ x_{12} & x_{22} & x_{32} & \cdots & x_{n2} \\ x_{13} & x_{23} & x_{33} & \cdots & x_{n3} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ x_{1m} & x_{2m} & x_{3m} & \cdots & x_{nm} \end{pmatrix} \tag{4.1}$$

This project will only focus on Supervised Learning, meaning that we know what the output is a binary representation of signal and background, such that we can use target values $\boldsymbol{t}$. Then we give the network a score depending on how close the predicted output is to real values $\boldsymbol{t}$. Then we repeat the process after tweaking the parameters $\boldsymbol{\beta}$ and see if the score gets better.

To this end, we will first provide a mathematical foundation for ML, starting with a brief overview of the different types of ML algorithms we will study, such as Neural Networks (NN) and Boosted Decision Trees (BDT). Other aspects such as the importance of feature selection and feature engineering in preparing the data for ML algorithms we will, as well as the concept of model evaluation and optimization will be discussed in Chapter 6. This project will take from granted that the reader is comfortable with linear algebra and jump straight into the mathematical fundations of ML.

4.1 Neural Networks

The theoretical foundation for this chapter is mainly based of Hjort-Jensen lecture notes [21, 22, 23]. Before begining we can briefly explain the idea behind NNs. As stated by Hjorth-Jensen in [21]:

> *The idea of NN is to to mimic the neural networks in the human brain, which is composed of billions of neurons that communicate with each other by sending electrical signals. Each neuron accumulates its incoming signals, which must exceed an activation threshold to yield an output. If the threshold is not overcome, the neuron remains inactive, i.e. has zero output*

That takes us to what a *neuron* is.

4.1.1 Artificial neurons

To describe the behaviour of a neuron mathematically we can use the following model that mimics how one neuron works

$$y = f\left(\sum_{i=1}^{n} w_i x_i\right) = f(z) \tag{4.2}$$

Where y, the output of the neuron, is the value of its *activation function* (See section 4.1.3), which has the weighted, w_i, sum of signals $x_i, \cdots, x_n$ received by n neurons that are in a preciding layer.

The goal of NNs is to mimic the biological nervous system by letting each neuron interact with each other by sending signals, for us is in the form of a mathematical function between each layer, callled the activation function. Most NNs consist of an input layer, an output layer and intermediate layers, called hidden layers. All the layers can contain an arbitrary number of neurons, and each connection between two neurons is associated with a weight variable w_i. The goal of using NNs is to teach the network the patterns of the data to then predict some target. In the context of our search for DM, by giving a NN our data set of events as its input layer, we can then train the network to classify events as signal or background.

Explained in greater detail if we were to look at a single event of the data, we start with an input with all the relevant features of the event, $\boldsymbol{X}$. Using Eq. (4.2) on every neuron on the next layer we can teach the network if there are any connections between the features, we can repeat this process for n layers. As an output we want a single neuron to see if it has predicted the event to be a signal or background, since this is binary output. After analysing the prediction we can use the labels on the target data $\boldsymbol{t}$ to tell (the network) whether it predicted correctly or wrong. We can then use a *cost function* and a specific *metric* to evaluate numerically how well the network predicted the output by giving it a score. Seeing how the results fare we can then back-propagate to shift the weights and biases and repeat the process until we are satisfied with our result. Each of these iterations is called an epoch.

To generalize our artificial neuron to a whole network we can look at a Multilayer Perceptron (MLP). An MLP is a network consisting of at least three layers of neurons, the input, one or more hidden layers, and an output. The number of neurons can vary for each layer. The above explanation is a very dense and simplified one. In reality it is complicated to find out which cost function, activation function, metric, etc. are best suited to the given problem. But before we get into the gory details we can explore the mathematical model that illustrates what was tried to be explained above.

4.1.2 Stochastic Gradient Descent

The way we "tweak the parameters $\boldsymbol{\beta}$ to see if the network prediction gets better" is by using something called the *Stochastic Gradient Descent* (SDG). Before explaining the SDG we have to look at the Gradient Descent (GD).

Given a cost function $C(\boldsymbol{\beta})$ we can get closer to the minimum by calculating the gradient $\nabla_\beta C(\boldsymbol{\beta})$ wrt. the unknown parameters from the NN $\boldsymbol{\beta}$. If we were to calculate the gradient at a specific point $\boldsymbol{\beta}_i$ in the parameter space, the negative gradient would correspond to the direction where a small change $d\boldsymbol{\beta}$ in this parameter space would result in the biggest decrease in the cost function. In the same way we in physics would determine where the local (or global) minima at a complex multidimensional potential numerically. In GD we can chose (which needs to be optimized) a step size η related to the size of

an iteration in the parameter space; this is called the *learning rate*. The mathematical function for an iteration in parameter space to optimize the parameter $\boldsymbol{\beta}$ such that it minimizes the cost function is given as

$$\boldsymbol{\beta}_{i+1} = \boldsymbol{\beta}_i - \eta \nabla_\beta C(\boldsymbol{\beta}_i) \tag{4.3}$$

To converge towards a minimum we should select a learning rate η small enough to not "step over" the minimum point of the cost-function-space, but also not too small to get stuck on a local minimum rather than the global minimum. Thus using the learning rate as a hyperparameter in a grid search is a good way to optimize a NN for a given task.

In GD one computes the cost function and its gradient globally for all data points. This quickly becomes computationally heavy when dealing with large datasets. Thus a common approach is to compute the gradient over batches of the data. For our purposes it would be optimal to use GD, but our data size is massive, of the order of 10^8 events (13 GB), becoming comupationally impossible. Thus instead of making a $n \times 10^8$ matrix (see Eq. (4.1)), we could for example split it into ten smaller matrices of $n \times 10^7$ to then perform a parameter update, making the computation possible. This is where SGD comes in, for each step, or epoch the data is divided randomly into N batches of size n. Then for each batch we use Eq. (4.3) to update the parameters, thus updating $\boldsymbol{\beta}_{i+1}$ N-times for each epoch. The idea of SGD comes from the observation that the cost function can almost always be written as a sum over n data points. The main advantage of SGD is not to ease the computation however, as using more batches also reduces the risk of getting stuck in a local minimum since it introduces a randomness of which part of the parameter space we move through, but this is at the cost of reducing statistics which might not be ideal for every problem.

4.1.3 Activation functions

As seen in Section 4.1.1, an important aspect of NNs are activation functions and cost functions. As shall become apparent in Section 4.1.4, when evaluating an activation function we get the neuron output, but what are these activation functions? Mathematically speaking, activation functions are: Non-constant, Bounded, Monotonically-increasing and continuous functions. In this project we will utilize two different activation functions at different layers. The first one is a sigmoid activation function

$$f(x) = \frac{1}{1 + e^{-x}} \tag{4.4}$$

which is the most basic activation function, this will be used from the input layer to our fist hidden layer as the risks of having little neuron activation is minimal here. On all other layers I will utilise a Rectified Linear Unit (ReLU)

$$f(x) = x^+ = \max(0, x) = \begin{cases} x & \text{if } x > 0, \\ 0. & \text{otherwise.} \end{cases} \tag{4.5}$$

which has better gradient propagation, meaning that there are fewer vanishing gradient problems compared to the sigmoidial function.

4.1.4 Feed Forward network

To describe how the network "guesses" outputs in a mathematical model we can start by looking at Eq. (4.2) where we got an output y from an activation function f that receives x_i as input. We can expand the function as as following

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b_i\right) = f(z) \tag{4.6}$$

where w_i is still the weight and we now use the previously introduced bias b_i which is normally needed in case of zero activation weights or inputs. The difference comes now in the interpretation; where in the activation $z = (\sum_{i=1}^{n} w_i x_i + b_i)$ the inputs x_i are now the outputs of the neurons in the preceding layer. Furthermore an MLP is fully-connected, meaning that each neuron received a weighted sum of the output of **all** neurons in the previous layer. To expand Eq. (4.6) we can first look at the output of every neuron i in a weighted sum z_i^1 for each input x_j on a layer

$$z_i^1 = \sum_{j=1}^{M} w_{ij}^1 x_j + b_i^1 \tag{4.7}$$

Where M stands for all possible inputs to a given neuron i in the *first* layer. Such that if we evaluate the weighted sum in an activation function f_i for each neuron i, then the output of all neurons in layer 1 is y_i^1

$$y_i^1 = f(z_i^1) = f\left(\sum_{j=1}^{M} w_{ij}^1 x_j + b_i^1\right)$$

To generalize this for l-layers, which may have different activation functions, we write it as

$$y_i^l = f^l(z_i^l) = f^l\left(\sum_{j=1}^{N_{l-1}} w_{ij}^l y_j^{l-1} + b_i^l\right)$$

Where N_l is the number of neurons in layer l. Thus when the output of all the nodes in the first hidden layer is computed, the values of the subsequent layer can be calculated and so forth until the output is obtained. With this we can show that we only need the inputs x_n to calculate the output with l hidden layers

$$y^{l+1} = f^{l+1}\left[\sum_{j=1}^{N_l} w_{ij}^{l+1} f^l\left(\sum_{k=1}^{N_{l-1}} w_{jk}^l \left(\cdots f^1\left(\sum_{n=1}^{N_0} w_{mn}^1 x_n + b_m^1\right)\cdots\right) + b_j^l\right) + b_i^{l+1}\right] \tag{4.8}$$

This shows that an MLP is nothing more than an analytic function, specifically a mapping of real-valued vectors $\hat{x} \in \mathbb{R}^n \to \hat{y} \in \mathbb{R}^m$. We can also see that the above equation is essentially a nested sum of scaled activation functions of the form

$$f(x) = c_1 f(c_2 x + c_3) + c_4$$

where the parameters c_i are the weights and biases. By adjusting these parameters we shift the activation function to better match the label we are training the data on, this is the flexibility of a NN. Something else we can note is that Eq. (4.8) can easily be changed into matrix notation, as hinted with Eq. (4.1). However this realization can help make computing the values a much easier task by for example utilizing TensorFlow or other mathematical packages in Python. An illustration showing the main idea of how a Feed forward network is set up is shown in Figure 4.1.

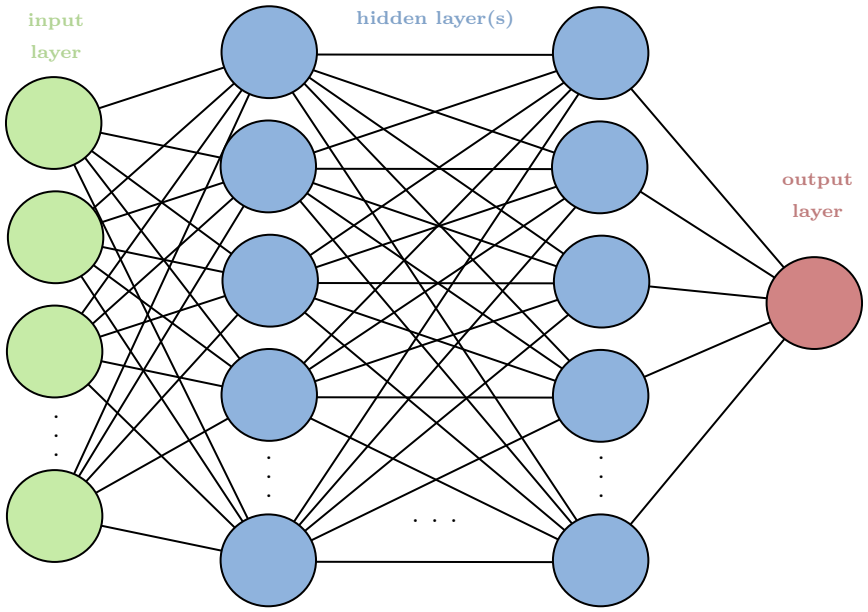


Figure 4.1: Basic illustration of a network with two hidden layers.

4.1.5 Back Propagation algorithm

So far we have only explained Feed Forward networks, which helps us to compute the output of the NN in term of basic linear algebra. We mentioned the possibility to adjust the weight and biases, but never explained how. Now is the time to dive into that subject, as we will explain the back propagation

algorithm. What we want to know is how the changes in the biases and weights in the network change the cost function, and how we could use the final output to modify the weights? Before we derive these equations we start by a plain regression problem, using the Mean Squared Error (MSE) as a cost function for pedagogical reasons

$$C(\hat{W}) = \frac{1}{2}\sum_{i=1}^{n}(y_i - t_i)^2 \tag{4.9}$$

where $\hat{W}$ is the matrix containing all the weights and (more importantly) t_i are the targets, which are the labels of events telling whether we have a signal or background event. To generalise this we first have to generalise to Eq. (4.7) for a layer l

$$z_i^l = \sum_{j=1}^{M} w_{ij}^l y_j^{l-1} + b_i^l \Leftrightarrow \hat{z}^l = \left(\hat{W}^l\right)^T \hat{y}^{l-1} + \hat{b}^l$$

where the right side is written in matrix notation. From the definition of z_j^l with an activation function, Eq. (4.6), we have

$$\frac{\partial z_j^l}{\partial w_{ij}^l} = y_i^{l-1} \tag{4.10}$$

and

$$\frac{\partial z_j^l}{\partial y_i^{l-1}} = w_{ij}^l$$

which again, with the definition of the sigmoid activation function, Eq. (4.4), gives us

$$\frac{\partial y_j^l}{\partial z_j^l} = y_j^l(1 - y_j^l) = f(z_j^l)(1 - f(z_j^l)) \tag{4.11}$$

Furthermore, we need to take the derivative of Eq. (4.9) with respect to the weights, doing so for a respective layer $l = L$ we have

$$\frac{\partial C(\hat{W}^L)}{\partial w_{jk}^L} = \left(y_j^L - t_j\right)\frac{\partial y_j^L}{\partial w_{jk}^L}$$

where the last partial derivative is easily computed using the chain rule with Eq. (4.10) and Eq. (4.11)

$$\frac{\partial y_j^L}{\partial w_{jk}^L} = \frac{\partial y_j^L}{\partial z_j^L}\frac{\partial z_j^l}{\partial w_{jk}^L} = y_j^L(1 - y_j^L)y_k^{L-1}$$

Such that

$$\frac{\partial C(\hat{W}^L)}{\partial w_{jk}^L} = \left(y_j^L - t_j\right) y_j^L(1 - y_j^L)y_k^{L-1} := \delta_j^L y_k^{L-1} \tag{4.12}$$

where we have defined the error

$$\delta_j^L := \left(y_j^L - t_j\right) y_j^L(1 - y_j^L) = f'(z_j^L)\frac{\partial C}{\partial y_j^L} \tag{4.13}$$

or in matrix form

$$\delta^L = f'(\hat{z}^L) \circ \frac{\partial C}{\partial \hat{y}^L}$$

where on the right hand side we wrote this as a Hadamard product[1]. This error δ^L is an important expression, since as we can see in the index form of this expression in Eq. (4.13), we can measure how fast the cost function is changing as a function of the j-th output activation. This means that if the cost function doesn't depend on a particular neuron j, then δ_j^L would be small.

We also notice that everything in Eq. (4.13) is easily computed. Thus we can also see how the weight changes the cost function using Eq. (4.12) quite easily. Another thing we can compute with Eq. (4.13) is

$$\delta_j^L = \frac{\partial C}{\partial z_j^L} = \frac{\partial C}{\partial y_j^L}\frac{\partial y_j^L}{\partial z_j^L}$$

[1] Also called *element-wise* product, $(A \circ B)_{ij} = (A \odot B)_{ij} = (A)_{ij}(B)_{ij}$

which can be interpreted in terms of the biases b_j^L as

$$\delta_j^L = \frac{\partial C}{\partial b_j^L}\frac{\partial b_j^L}{\partial z_j^L} = \frac{\partial C}{\partial b_j^L} \tag{4.14}$$

where the error δ_j^L is exactly equal to the rate of change of the cost function as a function of the bias.

Something interesting is that when using Eq. (4.12 - 4.14) we see that if a neuron output y_j^L is small, then the gradient term, Eq. (4.12), will also be small. We say then that the weight learns slowly, meaning that the contribution of said neuron is less important "to fix" than those that have a higher weight. Of course this example is a very simple one to wrap our heads around, but the magic comes when the algorithm is evaluating a random neuron in a layer n, after using many layers the NN becomes a **black box** for us to wrap our heads around!

It is also worth noting that when the activation function is flat at some specific values (which also varies with the chosen function!) the derivative will tend towards zero, making the gradient small, meaning the network is learning slow as well. To finish up our back propagation algorithm we still need one equation. We are now going to propagate backwards in order to determine the weights and biases. We start by representing the error in the layer before the final one $L-1$ in term of the errors of the output layer. If we try to express Eq. (4.13) in terms of the output layer $l+1$, using the chain rule and summing over all k entries we get

$$\delta_j^l = \sum_k \frac{\partial C}{\partial z_k^{l+1}}\frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k \delta_k^{l+1}\frac{\partial z_k^{l+1}}{\partial z_j^l}$$

recalling Eq. (4.7) (replacing 1 with $l+1$) we get

$$\delta_j^l = \sum_k \delta_k^{l+1} w_{kj}^{l+1} f'(z_j^l) \tag{4.15}$$

Which is the final equation we needed to start back propagating.

4.1.6 Summary

To summarize the whole process of the NN

- First take the input data $\mathbf{x}$ and the activation $\mathbf{z}_1$ of the input later, and then compute the activation function $f(z)$ to get the next neuron outputs $\mathbf{y}^1$. Mathematically this is taking the first step of the feed forward algorithm, i.e. choosing $l = 0$ in Eq. (4.8)
- Secondly we commit all the way in Eq. (4.8) and compute all $\mathbf{z}_l$, activation function and $\mathbf{y}^l$.
- After that we compute the output error $\boldsymbol{\delta}^L$ by using Eq. (4.13) for all values j.
- Then we back propagate the error for each $l = L-1, L-2, \cdots, 2$ with Eq. (4.15).
- The last step is then to update the weights and biases using Eq. (4.3) for each l and updating using

  $$w_{jk}^l \leftarrow w_{jk}^l - \eta\delta_j^l y_k^{l-1}$$

  and

  $$b_j^l \leftarrow b_j^l - \eta\delta_j^l$$

This whole procedure is usually called an epoch, which we can repeat as many times as we want to better reduce the cost function in hope of converging to the global minima.

4.2 Boosted Decision Trees

In the previous chapter, we explored how NNs can aid in the task of signal and background classiciation, such as the one in this thesis where we are looking DM signal in the SM background. In this chapter, we will delve into Boosted Decision Trees (BDTs), another powerful tool for binary classifications. BDTs, unlike neural networks, are not based on simulating biological neurons. Instead, they rely on decision trees, a simple yet powerful idea that has been around for decades. [24, 25] Decision trees are built by recursively splitting data based on the values of features until a stopping condition is met. BDTs take this idea one step further by training an ensemble of decision trees, where each tree attempts to correct the mistakes of its predecessors.

BDTs have several advantages over other machine learning algorithms, such as neural networks, for binary classification problems. They are highly interpretable, which allows us to understand how they arrived at their decisions. This is particularly important in this field, high energy particle physics, where the ability to interpret results is crucial. BDTs are also highly resilient to overfitting and can handle missing data effectively. Given their strengths, BDTs have been widely used in particle physics for binary classification tasks, such as distinguishing signal from background events. The idea of the BDTs used in high energy physics today was proposed as early as 2001 by Friedmann in the paper *Greedy Function Approximation* [26], and since then have become an indispensable tool in the field. In particular, in the ATLAS collaboration challenge *The Higgs boson machine learning challenge* [27] the creation of a BDT package called `XGBoost` [28] increased the popularity even more as they won the challenge. Today `XGBoost` has become a standard ML tool used in the field.

In this chapter, we will explore the theory behind BDTs in the context of *extreme gradient boosting*, for this we we are mainly interested in two ways of making DTs, a set of Classification And Regression Trees (CART). The theory is mainly based of Hjort-Jensens lecture notes [29, 30] and section 2 on the original `XGBoost` paper [28].

4.2.1 Decision trees

Starting with Decision Trees (DT) in the context of this DM search. The idea behind DTs is to find the most important features which contain the most information of signal (DM), and then split the dataset along the values of these features with the goal of creating a dataset containing pure signal. As we are going to use kinematical variables as features we will use this to both augment and automate the classical cut and count method (Chapter 3.3), which is based of making kinematical cuts on the variables. To find which kinematical features are most important in splitting the dataset we have to achieve a stopping criteria where we end up on a so-called *leaf node*.

A DT is typically divided into a *root nodes*, *interior nodes* and a final *leaf nodes*, also called leaves, which are all connected by *branches*, hence the name decision *tree*. As mentioned in the start of this chapter, we will look at two ways of making DTs, the first one we will look at is the *regression tree*.

4.2.2 Regression trees

As previously mentioned the leaves contain the prediction of a trained network, and are used to make new predictions on new datasets based on the information the DT learning in the branching process. As how to construct trees, there are mainly two steps we need to follow:

1. Split the predictior space (set of possible values $x_1, x_2, \cdots, x_p$) into J distinct and non-overlapping regions, $R_1, R_2, \cdots, R_J$
2. For every observation that fals into the region R_j, we make the same prediction, which is the mean of the response values for the training observations in R_J

is this too similar to 9.3 in [29]?

But how do we construct the regions $R_1, R_2, \cdots, R_J$? In theory these regions could have any shape, but for simplicity and pedagogial reasons we will choose to divide the predictor shpe into high-dimensional rectangles, or boxes. This means that the goal is to find boxes $R_1, R_2, \cdots, R_J$ that minimize a *cost*

function, which again, for pedagogical reasons will be the a MSE (Eq. (4.9)), defined as

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \bar{y}_{R_j})^2$$

where $\bar{y}_{R_j}$ is the mean response for the training observations withing box j. Ideally we would consider every possible partition of the feature space into J boxes, but his is computationally infeasable. Thus the common apporach is to begin at the top of the tree, where all observations belong to a single region, and then split the predictor space; where every split is indicated via two new branches down in the tree. This is greedy (as in *Greedy Function Approximation* [26]) since the best split is mad in the first step, rather than looking ahead and picking a split that might lead to a better tree in a future step.
To make any split we start by selecting the predictor x_j and a cutpoint s that splits the predictor space into two regions R_1 and R_2

$$\{X|x_j < s\} \quad \text{and} \quad \{X|x_j \geq s\}$$

so that we obtain the lowest MSE,

$$\sum_{i:x_i \in R_1} (y_i - \bar{y}_{R_1})^2 + \sum_{i:x_i \in R_2} (y_i - \bar{y}_{R_2})^2$$

where we consider all predictors $x_1, x_2, \cdots, x_p$ and each possible value s for each of them, where these values can be randomly assigned numbers. For any j and s we define the pair of half-planes where $\bar{y}_{R_1}$ and $\bar{y}_{R_2}$ are the mean responses for the training observations in $R_1(j,s)$ and $R_2(j,s)$ respectively. The goal is to find the value j and s such that we minimize the cost function, which can be done quickly depending on the number of predictors.

Then we repeat the process looking for the best predictor and best cutpoint, but instead of stopping with two regions, we split one of the two into another two regions creating a total of three regions. This is called the depth of a tree, and we can in principle continue to split indefinetely, however one usually sets a stopping criterion on how many events should be at a leaf.

The method explored above is straight forward, but often leads to overfitting and unnecessarily large and complicated trees. To mediate this one usually uses a so-called *Cost complexity* pruning algorithm. For regression precedures, the algorithm, rather than considering every possible subtree, comnsiders a sequence of trees indexed by a nonnegative tuning parameter α. For each value of this α there corresponds a subtree $T \in T_0$ such that

$$\sum_{m=1}^{\bar{T}} \sum_{i:x_i \in R_m} (y_i - \bar{y}_{R_m})^2 + \alpha T \tag{4.16}$$

is as small possible. In the equation above T is the number of terminal nodes of the tree T, R_m is the rectangle corrresponding to the m-th terminal node. This tuning parameter α controls a trade-off between the subtree's complexity and its fit to the training data. As α increases, the above equation will give us a higher value, meaning that our trees would normally prioritize less complex models. This precedure is repeated untill we minimize Eq. (4.16) and choose a suitable value for α. This is the essense of regression trees.

4.2.3 Classification trees

The second DT type we will study is the *classification tree*. Classification trees are very similar to regresion trees, except that they are used to predict a qualitative response rather than a quantitative one. This means that classification trees are used for predictive modeling in which the output variable is discrete, such as classifying an email as spam or not spam, or predicting whether an event in our dataset is a DM signal or SM background event.

Like regression trees, classification trees recursively split the data based on the values of input variables until a stopping condition is met. However, the splitting criteria for classification trees are different. Classification trees cannot use the MSE function used in regression trees as a criterion for making binary splits, instead they use the *classification error rate*. The classification error rate, classification trees have discrete outputs, is simply the fraction of the training observations in a region that do not belong to the

most common class.

Classification trees use measures such as Gini index or the entropy to determine quality of a split at each node. The idea behind these quality measures is to select the split that maximizes the separation between the different categories of the output variable.

If a classification tasks takes for example $k = 1, 2, \cdots, K$ values as outputs, we can define a probability distribution function p_{mk} that represents the number of observations of k in a region R_m with N_m observations. We can represent this likelihood function in terms of the proportion $I(y_i = k)$ of obervations of this output in region R_m as

$$p_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k)$$

where p_{mk} represents the majority class of observations in region m. The tree most common ways of splitting a node are given by: The misclassification error

$$p_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i \neq k) = 1 - p_{mk}$$

The Gini index G

$$G = \sum_{k=1}^{K} p_{mk}(1 - p_{mk}) \tag{4.17}$$

and the entropy s

$$s = -\sum_{k=1}^{K} p_{mk} \log(p_{mk})$$

4.2.4 The CART algorithm

As we are going to look at extreme gradient boosting, the main algorithm for this type of DT is the Classification And Regression Tree (CART). For classification, the CART algorithm splits the dataset in two subsetsusing a single feature k and treshold t_k. This could for example be a treshhold set by a value of the invariant mass of an event we are trying to classify as background or signal. The way we optimize the pair (k, t_k) such that we get the purest subset is by for example using the gini factor G. With this the cost function tries to minimize

$$C(k, t_k) = \frac{m_{left}}{m} G_{left} + \frac{m_{right}}{m} G_{right}$$

where $G_{left/right}$ calculats the impurity of the left/right subset using Eq. (4.17) and $m_{left/right}$ is the number of events in the left/right subset. Once the algorithm successfully splits the training set into two, it splits the subsets using the same agorithm, and so on, recursively. For our purposes we will make the DT stop searching for the pair (k, t_k) once we have reaches the maximum depht we chose, or whenever the sum of the weights on a leaf is one. Both of these are hyperparameters that need to be optimized, for that see Chapter 6.3.3.

The CART algorithm for regression works similarly to the one for classification, but instead of trying to split the training set in a way that minimizes the gini or entropy impurity, it tries to split the training set in a way that minimizes the (MSE). Meaning that we have

$$C(k, t_k) = \frac{m_{left}}{m} MSE_{left} + \frac{m_{right}}{m} MSE_{right}$$

with

$$MSE_{node} = \frac{1}{m_{node}} \sum_{i \in node} (\bar{y}_{node} - y_i)^2$$

and

$$\bar{y}_{node} = \frac{1}{m_{node}} \sum_{i \in node} y_i$$

4.2.5 Ensemble modeling and (extreme) gradient boosting

Now that we have explained how DTs wor, we still need more information to get the full picute of *Boosted* DTs. For this section of the chapter we. As we are going to look at a massive dataset, a single tree is not strong enough to be used in practice. What is actually used is the *ensemble model*, which sums up the prediction of multiple trees togheter. What is important of this models is that the different trees we are summing togheter try to complement each other. For example if the first split on one tree is on the invariant mass of a dilepton pair, while on another tree it is the MET, then we know physically that these are important features that DM + dilepton models have as a signature, meaning that it should be a good combination. Mathematically, we can write the prediction of an ensemble model in the form

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), \quad f_k \in \mathcal{F} \tag{4.18}$$

where K is the number of trees we want to sum over, f_k is a function of the functional space $\mathcal{F}$, where $\mathcal{F}$ is the set of all possible CARTs. With this we want to minime a the objective function of the form

$$\mathcal{L} = \sum_{i}^{n} C(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

where we sum over n leaves, C is training cost function (for example it can be the MSE), and $\Omega(f_k)$ is the complexity of the tree f_k, which penalizes the more complex models. We can see from both functions above that we need to learn all trees at once to make predictions, this is however both computationally demanding and intractable. Instead, we can use an additive strategy, where we fix what we have learned, and add a new tree at a time. Using Eq. 4.18 with 0 trees and going up we see that we get

$$\begin{aligned}
\hat{y}_i^{(0)} &= 0 \\
\hat{y}_i^{(1)} &= f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i) \\
\hat{y}_i^{(2)} &= f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i) \\
\hat{y}_i^{(3)} &= f_1(x_i) + f_2(x_i) + f_3(x_i) = \hat{y}_i^{(2)} + f_3(x_i)
\end{aligned}$$

and so on. Mathematically we see that a prediction at step t as in $\hat{y}_i^{(t)}$ takes the form

$$\hat{y}_i^{(t)} = \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i) \tag{4.19}$$

Such that we can now write the objective to be

$$\mathcal{L}^{(t)} = \sum_{i}^{n} C(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) \tag{4.20}$$

from which we can set up our choice for cost function. In general case we take the Taylor expansion of the loss function up to the second order such that we get

$$\mathcal{L}^{(t)} = \sum_{i}^{n} \left[C(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) + const$$

where

$$g_i \equiv \partial_{\hat{y}_i^{(t-1)}} C(y_i, \hat{y}_i^{(t-1)}) \quad \text{and} \quad h_i \equiv \partial^2_{\hat{y}_i^{(t-1)}} C(y_i, \hat{y}_i^{(t-1)})$$

Such that the specific objective at step t becomes

$$\sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) \tag{4.21}$$

where we removed all constants. Such that Eq (4.21) becomes the optimization goal for the new tree. As we can see the above equation can take into account any cost function as it is written in a general form with a Taylor expansion. Furthermore, we can rewrite this by expanding

$$\Omega(f_t) \to \alpha T + \frac{1}{2} \lambda \sum_{j=1}^{T} w_j^2$$

where the first term is the same as in Eq. (4.16), and the second is called the *regularization term* (see Chapter 4.3.1). Inserting this into Eq. (4.21) we get

$$\mathcal{L}^{(t)} = \sum_{j=1}^{T} \left[\left(\sum_{i=1}^{n} g_i \right) w_j + \frac{1}{2} \left(\sum_{i=1}^{n} h_i + \lambda \right) w_j^2 \right] + \alpha T$$

such that with a fixed tree structure, q, we can define the optimal weight w_j^* on a leaf j as

$$w_j^* = -\frac{\sum_{i=1}^{n} g_i}{\sum_{i=1}^{n} h_i + \lambda} \tag{4.22}$$

and correspondingly the optimal objective reduction

$$\mathcal{L}^{(t)}(q) = -\frac{1}{2} \sum_{j=1}^{T} \frac{(\sum_{i=1}^{n} g_i)^2}{\sum_{i=1}^{n} h_i + \lambda} + \alpha T \tag{4.23}$$

With both of these equations we have all we need to use BDTs, the extreme part of the gradient boosting is something the `XGBoost` algorithm has implemented in terms of system optimization, such as parallelising the tree boosting task, and can be read in more detail on section 4 of the original paper [28].

4.2.6 Summary of idea

So in short BDTs split datasets based on different values of the features on the dataset, figure 4.2 shows an illustration of how DTs work.

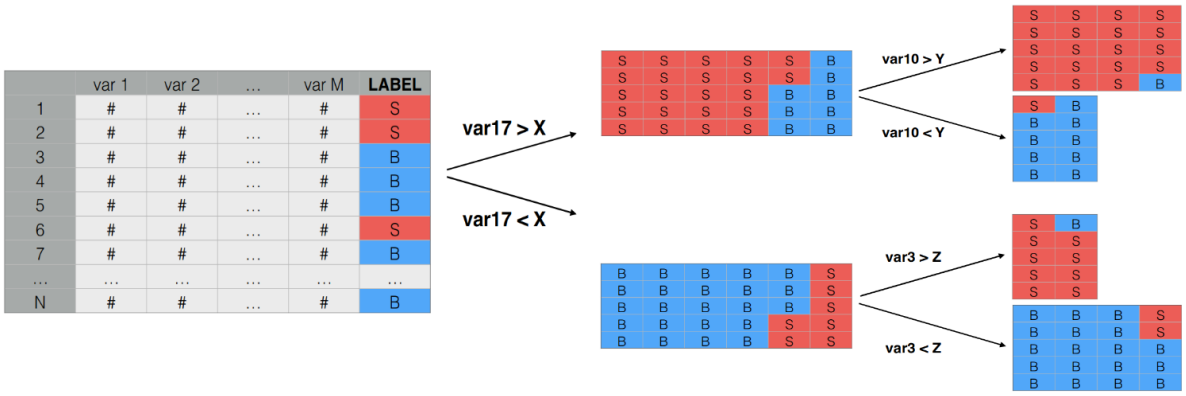


Figure 4.2: Image showcasing how decision trees work by splitting a dataset of N events with M features whenever a treshhold X, Y or Z for a feature is passed. The label S and B state whether the event is signal or background respectively.

The above illustration just shows how one tree might work, in reality we can choose to have an arbitraty number of trees and combine their results with the ensembling and boosting method. Then after combining the generated tree structures we use Eq. (4.22) and Eq. (4.23) to minimeze the objective to get the most accurate predictions as possible.

4.3 Tools used for both algorithms

more motivation

Now that we have explained how both NNs and BDTs work, we will explain how tools used by both algorithms work.

4.3.1 Cost functions

Cost functions have been mentioned throughout this chapter, but what are they? Cost functions are what we will utilize to evaluate how well the output of the network fares against the target, i.e. if our network "guesses" right whether an event is signal or background, thus making this a very important part of our network! Before getting into this we first have to look at logistic regression. Since we are studing a binary classification task where the output is either $t_i = 0$ or $t_i = 1$, meaning background or signal. We can introduce a polynomial model of order n as

$$\hat{y}_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \cdots + \beta_n x_i^n$$

where the hat notation, $\hat{y}$, symbolises a condensed matrix form of the output of a layer. We can define the probabilities of getting $t_i = 0$ or 1 given our input x_i and $\boldsymbol{\beta}$ with the help of a logistic function. Using this we get the probability as

$$p(t_i = 1|x_i, \boldsymbol{\beta}) = \frac{1}{1 + e^{-t_i}}$$

and

$$p(t_i = 0|x_i, \boldsymbol{\beta}) = 1 - p(t_i = 1|x_i, \boldsymbol{\beta})$$

We want to then define the total likelihood for all possible outcomes from a dataset $\mathcal{D} = \{(t_i, x_i)\}$, with the binary labels $t_i \in \{0, 1\}$, applying the Maximum Likelihood Estimation (MLE) principle. This gives us

$$P(\mathcal{D}|\boldsymbol{\beta}) = \prod_{i=1}^{n} \left[p(t_i = 1|x_i, \boldsymbol{\beta})\right]^{t_i} \left[1 - p(t_i = 1|x_i, \boldsymbol{\beta})\right]^{1-t_i}$$

from which we obtain the log-likelihood

$$C(\boldsymbol{\beta}) = \sum_{i=1}^{n} \left(t_i \log p(t_i = 1|x_i, \boldsymbol{\beta}) + (1 - t_i) \log[1 - p(t_i = 1|x_i, \boldsymbol{\beta})]\right)$$

By taking the parameter $\boldsymbol{\beta}$ to first order in x_i and reordering the logarithm we get

$$C(\boldsymbol{\beta}) = -\sum_{i=1}^{n} (y_i(\beta_0 + \beta_1 x_i) - \log(1 + \exp(\beta_0 + \beta_1 x_i))) \tag{4.24}$$

This equation we will use throughtout the thesis and is known as the *cross entropy*. The two beta parameters used are the weight and biases, $\beta_1 = w$ and $\beta_0 = b$. The goal is to change these parameters such that it minimizes the cost function.

L2-regularization

add source for article?

Furthermore we will add an extra term to the cost function, proportional to the size of the weights. We do this to constrain the size of the weights, so they don't grow out of control, this is to reduce *overfitting*. We will use the so-called *L2-norm* where the cost function becomes

$$C(\boldsymbol{\beta}) \to C(\boldsymbol{\beta}) + \lambda \sum_{ij} w_{ij}^2 \tag{4.25}$$

meaning that we add a term where we sum up **all** the weights squared. The factor λ is called the regularization parameter. The L2-norm combats overfitting by forcing the weights to be small, but not making them exactly zero. This is so that less significant features still have some influence over the final prediction, although small.

4.3.2 Sample weight

A "sample weights" array is an array of numbers that specify how much weight each sample in a batch should have in computing the total loss. It is commonly used in imbalanced classification problems (the idea being to give more weight to rarely-seen classes). Taken from the original documentation. To mathematically illustrate how this weight works is by multiplying a constant term χ into the cost/loss function that is used further on the error propagation. As an example we can extend the simple MSE, Eq. (4.9) to

$$C(\hat{W}) = \frac{1}{2}\sum_{i=1}^{n} \chi_i (y_i - t_i)^2 \tag{4.26}$$

This will become an important feature when discussing network optimization in chapter 6, and especially for particle physists as we want to re-weight Monte Carlo events to expected events to correctly showcase kinematical distributions.

4.3.3 Network evaluation methods

There are many ways to evaluate how well a ML network does. For this thesis, we will mainly use two methods. As the empasis of the thesis is to conduct a binary search (DM or not). We will utilise the Area Under the "Receiver Operating Characteristic"-Curve as well as validation plots.

write more motivation?

Area under the ROC-curve

To evaluate how well our networks do. We will use the acuracy, which literally tells us how many events it guessed right. But also take into account a more advanced metrics, this is the Area Under the "Receiver Operating Characteristic (ROC)" Curve (AUC). Before explaining what the AUC is we first need to expclote the ROC curves. The ROC curves helps us illustrate how successfull a network is a *binary classification* (only). The ROC curve, as the name states is a curve, where we plot the *True Positive Rate* (TPR) against the *False Positive Rate* (FPR). The TPR is defined by dividing the *True Positive* (TP), which is when the netowrk correctly guesses a signal event to be a signal event, by the *Positive* (P), which is the total number of signal events in the data.

$$\text{TPR} = \frac{\text{TP}}{\text{P}}$$

On the other hand, the FPR is when we divide the *False Positive* (FP), when the network guesses a background event to be signal event, divided by the *Negatives* (N), the total number of background events in the data.

$$\text{FPR} = \frac{\text{FP}}{\text{N}}$$

To get a numerical value of how well a network classifies data we thus calculate the AUC. The TP and the FP are both going to be predictions from the networks, which for this thesis will gave an event a score from 0 to 1[2], meaning that if we had a perfect network which guesses everything correctly we would only get TPs and thus an AUC of 1. If we were to get an area of 0.5 this would mean that the network is randomly guessing whether an event is a signal or background, practically making a coin toss for every event. The goal is to have a network to give an AUC score as closely to 1 as possible.

While only using the accuracy as a metric is not a bad start, it is not favorable to use as a metric if datasets are ubalanced. As en example, if we had a dataset with 100 events, where 95 were background events and 5 were signal events, if we only used accuracy as a metric we would be inlcined to think that an accuracy of 95% is great. However, as the dataset is unbalanced, the network could easily take a shortcut and guess every event to be a background event. Meaning that the network learned nothing. This is where the AUC comes into play, as this metric highlights whenever a network guesses wrongly in a binary classification. Meaning that if we used the worst case scenario with 95% accuracy when guessing everything to be background, we would have a TPR = 0 and a FPR = 0, if we were to plot this we would get a flat line on the FPR and TPR axis, meaning that it would give us an AUC of 0.5. Highlighting that our network is randomly guessing, and that it needs to be optimized further.
Comment for farid/eirik, I used wikipedia as a source on the theory here, should I cite that?

can I include my own made figures here (might be results)

[2]Where 0 means that the network believes an event has 0% chance of being a signal event, and 1 means that the networks believes with 100% certainty that an event is a signal event.

Validation plots

Is this the real name of the plots?

Another tool that we will use to evaluate how well a network does is by making so-called *validation plots*. The way these plots work is by first dividing the *already predicted* data set into background and signal. Then making histograms from 0-1 with the prediction score of the network for each event. As a score closer to 1 means that the network predicts an event to be a signal event, and 0 means that the network predicts a background event. Then we would ideally have all the signal events shoved to the right and all the background events shoved to the left. Usually this is not the case in high energy particle physics, as some background events might be similar to signal events. We will therefore utilize this kind of plots to see the ditribution of the network predictions.

It is standard practice to make fill the histrograms with semi-transparent colors, such that one can visualise the whole spectra for both background and signal. In this thesis we will use this, as well as validation plots where we divide the SM events into their respective background channels and fill them with opaque colors, while leaving the signal unfilled. We will in adition include real data points in the latter plots, to see how well SM simulations agree with the data. Ideally we want our data to not agree with the SM simulations when the network predicts a score close to one, as this discrepancy could be a hint of new physics.

can I include my own made figures here (might be results)

Feature importance

For BDTs, as we can calculate which features had the most impact when splitting the data, we will also include feature importance plots. As the name states these plots tells us how important the features in the dataset were in the training to separate signal from background. As we are using the `XGBoost` package when working with BDTs, we will use the inbuilt functions of the package to plot the feature importance.

When plotting the feature importance there are several metrics we can use to determine how important they are, for this thesis we will mainly look at

- Gain: which measures the relative contribution of a feature to the model's performance
- Weight: the number of times a feature appears on a tree to split the data
- Cover: the number of samples affected by the split with a feature

Where the first one is the standard used by `XGBoost`.

Part II

Implementation / Methods

Chapter 5

Data Preparation

Now is the time to focus on the critical task of preparing the data for our DM search using ML. This task is essential because accurately distinguishing between signal and background events is a key challenge in searching for any new signal event, as seen on Section 3.3. In this chapter we will define what *background* and *signal* actually are, as well as selecting appropriate kinematical cuts to define the control region for our search. In addition we will explain the data preparing procedure such that it is in a format that is well-suited for our ML algorithms. The reason this process is of great importance is because we can maximize the chances of our ML agrorithm to accurately identify any DM signals. Moreover, by providing a detailed explanation of the data preparation process, we can ensure that our dataset is both reliable and effective, and that our analysis is robust and trustworthy. In short, this chapter represents a critical step towards achieving our goal of using ML to classify DM from SM events using ATLAS data, and it is essential to the success of our overall research goal.

5.1 Standard Model background estimation

As we are going to look at dilepton final states with missing trasnverse energy to try to teach our ML to learn DM signatures, then we need to take into account all possible SM background event that have these kind of final states. The SM backgrounds processes we will look at are explained in the next subsections. In Appendix A is the full list of dataset IDs for background process used in this thesis.

5.1.1 W and Drell Yan

$W/Z/\gamma^*$+jets as all of these can directly decay into two leptons, $W/Z/\gamma \to ll'$, where the prime marks a different flavour. All the production mechanism for these background processes can be seen in Figure 5.1. As the W boson behaves differenty than the Z and γ, since the W boson can decay into neutrinos giving us MET, while the Z and γ processes have the highest cross section. We will divide these background processes into two, W and Z/γ^* + jets, also called *Drell Yan* processes. To simulate these background processes Sherpa 2.2.11 [31] was used.

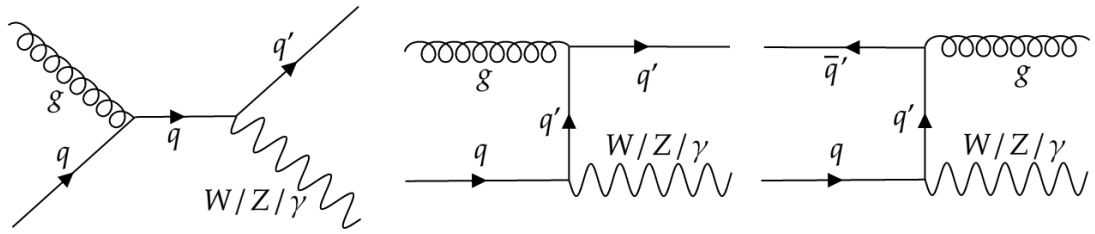
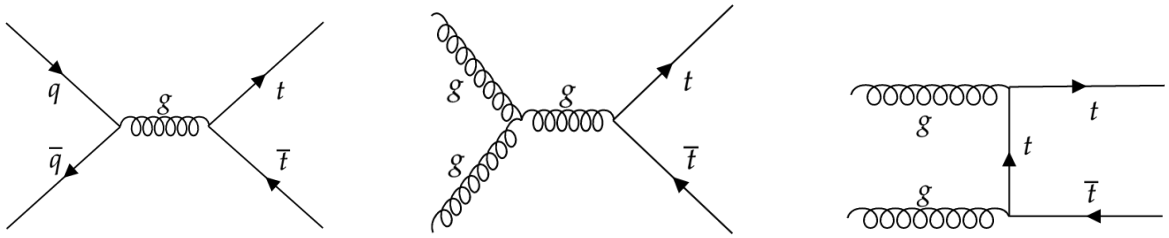


Figure 5.1: Diagrams showcasing SM $W/Z/\gamma$+jets production. A quark marked with a prime, indicated that the quark might be a diferent flavour after interacting with a boson.

5.1.2 TTbar

Another SM background process we will look at are processess that have a $t\bar{t}$ in them. Since the top has a high branching ratio of decaying into a b-quark and a W boson, $t \to bW^+$, where the W boson can again decay into a neutrino giving us MET. The main production mechanism for these processes can be seen in Figure 5.2. To simulate these background processes Powheg-Box v2 [32] interfaced with Pythia 8 [33] was used.

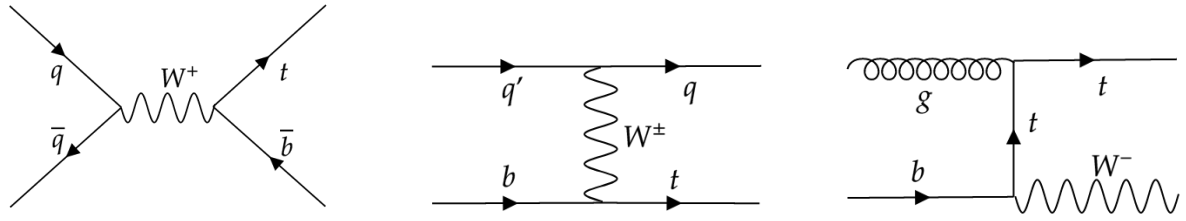


(a) s-channel (b) Gluon fusion (c) t-channel

Figure 5.2: Diagrams showcasing SM $t\bar{t}$ production

5.1.3 Single top

On the same note we have processes with a single top. Where again the the top has a high branching ratio of decaying into a b-quark and a W boson, $t \to bW^+$, where the W boson can again decay into a neutrino giving us MET. The main production mechanism for these processes can be seen in Figure 5.3. To simulate these background processes Powheg-Box v2 [32] interfaced with Pythia 8 [33] was used.

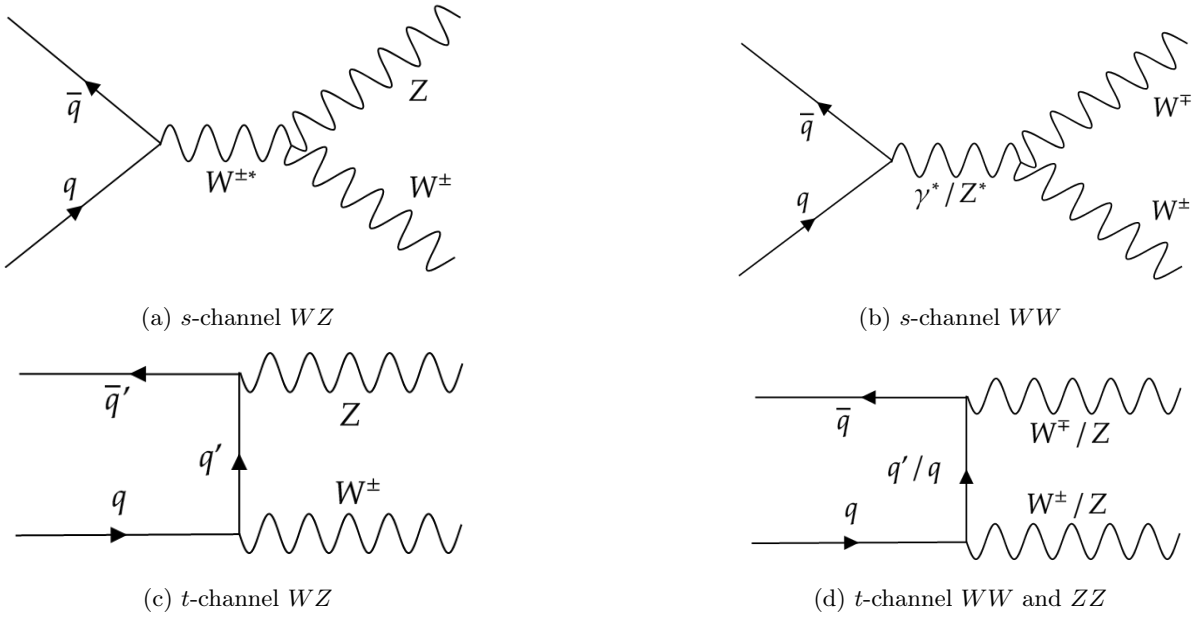


(a) s-channel (b) t-channel (c) tW

Figure 5.3: Diagrams showcasing SM single top productions. A quark marked with a prime, indicated that the quark might be a diferent flavour after interacting with a boson.

5.1.4 Diboson

The last SM bacgkround process we will look at are processes containing two bosons, called *diboson* backgrounds. The two SM bosons we will consider when looking into these final states are the W and Z, as these can decay as $W/Z \to ll'/ll$ or $W \to l\nu_l$, where the prime means a different lepton flavour. The main production mechanism for these processes can be seen in Figure 5.4. To simulate these background processes Sherpa 2.2.11 [31] was used.



(a) s-channel WZ (b) s-channel WW

(c) t-channel WZ (d) t-channel WW and ZZ

Figure 5.4: Diagrams showcasing SM diboson production. A star superspcript on a boson indicates that the boson needs to be virtual and off mass shell. A quark marked with a prime, indicated that the quark might be a diferent flavour after interacting with a boson.

5.2 Dark Matter samples

5.3 Object selection

Before getting into our DM search and regions we will study we have to define how standard objects are defined. By standard objects we mean electron, muons and jets. As this thesis is made using ATLAS data and simulations we will use the same standard crietria cuts they choose.

The electrons are selected using the criteria on Table 5.1.
The muons are selected using the criteria on Table 5.2.
The jets are selected using the criteria on Table 5.3.

Table 5.1: Electron selection criteria.

Feature	Selection criteria
Transverse momentum	$p_T > 25$ GeV
Pseudorapidity range	$\|\eta\| < 1.37$ or $1.52 < \|\eta\| < 2.47$
d_0 significance cut	$\|d_0(\sigma)\| < 5$
z_0 cut	$\|\Delta z_0 \sin\theta\| < 0.5$mm

Table 5.2: Muon selection criteria.

Feature	Selection criteria
Transverse momenum	$p_T > 27(20)$ GeV for leading (subleading)
Pseudorapidity cut	$\|\eta\| < 2.5$
d_0 significance cut	$\|d_0(\sigma)\| < 5$
z_0 cut	$\|\Delta z_0 \sin\theta\| < 0.5$mm

Table 5.3: Jet selection criteria.

Feature	Selection criteria
Algorithm	Anti-k_t
R-parameter	0.4
Transverse momentum	$p_T > 20$ GeV
Pseudorapidity cut	$\|\eta\| < 4.5$
Jet Vertex Tagger	> 0.5 for $p_T < 60$ GeV, $\|\eta\| < 2.4$
forward JVT	fJVT < 0.4 and $\|$timing$\| < 10$ ns for $p_T < 120$ GeV, $2.5 < \|\eta\| < 4.5$
b-jet tagging	DL1r score > 0.665 with 85% efficiency

The missing transverse energy is reconstructed from jets selected and calibrated according to the information in Table 5.3. In addition to these a kinematical cut of $m_{ll} > 10$ GeV will be made to exclude hadron productions.

5.4 Control region

Now that we have defined what our background and signal are, the next step is to create a so-called *control region*. The control region is the kinematical region we will use as a base for our search. As we are conducting a model inedpendent search we want our kinematical cuts to be minimal and as general as possible. As we are looking at a dilepton final state, then we need to first define what we mean by that. If we were only searching for DM with the Z' model, then a sensible definition would be a Same Flavour Opposite Sign (SFOS) leptons ($e^{\pm}e^{\mp}, \mu^{\pm}\mu^{\mp}$). However to stay as general, and model independet, as possible we will also study other possible combinations as these might be important for theories such as SUSY or Lepton Flavour Violating (LFV) models. These are Different Flavour Same Sign (DFSS), DFOS and SFSS lepton pairs.

Since we are looking for DM, which we expect to behave similarly to a neutrino, then a nice kinematical variable to set a general cut to isolate signal from background is the MET, we can see the distribution of MET on all of Run II in a dilepton final state in Figure 5.5.

As we are conducting a model independent search, we want to use minimal cuts. The MET cut made in this thesis was chosen to be of 50 GeV (violet line in plot), meaning we will *only* look at events where the MET is greater than this. As we can see from Figure 5.5 by making a kinematical cut on 50 GeV we are cutting out a massive part of the background processes while only losing a small part of the signal.

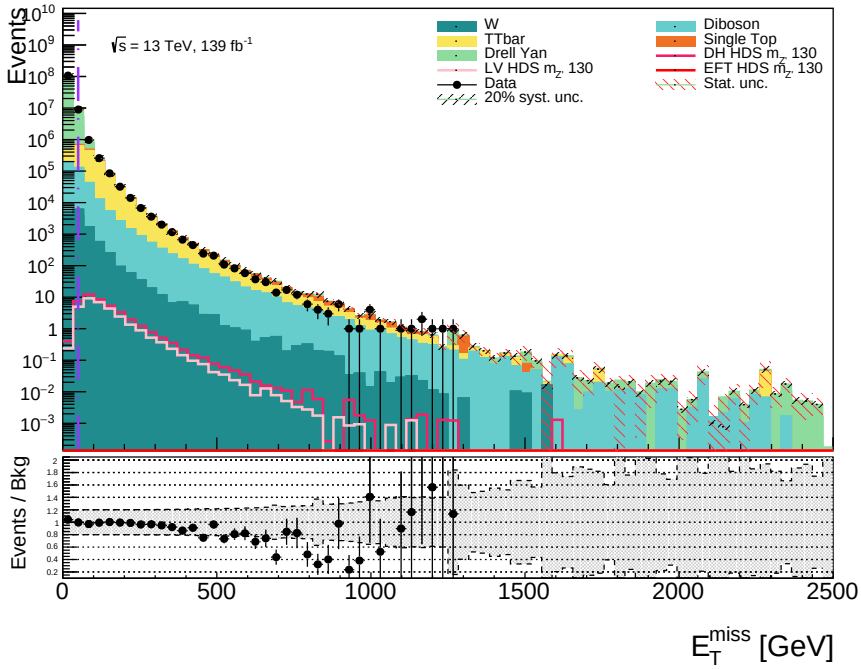


Figure 5.5: Distribution of MET when looking at a dilepton final state in all of Run II. The violet line shows where we will make out MET cut to create a control region. The DM models here have the highest cross section making them the most visible to our purposes.

Another kinematical cut that will be included in this thesis is another jet p_T on top of the one from Table 5.3. This is because jet reconstruction is already a hard task, but when pushing the limits of our selection criteria with a MET cut becomes even harder. And the worse the reconstruction of the jets the worse the agreement between MC simulations and real data is, and as we already know, the SM agrees with what we observe with an incredible accuracy. In Figure 5.6 we can see how the data and MC simulations of the SM agree when ounting the number of b- and light jets with a p_T cut of $\geq$ 30 GeV and $\geq$ 40 GeV respectively. The agreement with different p_T cuts can be seen in Appendix C

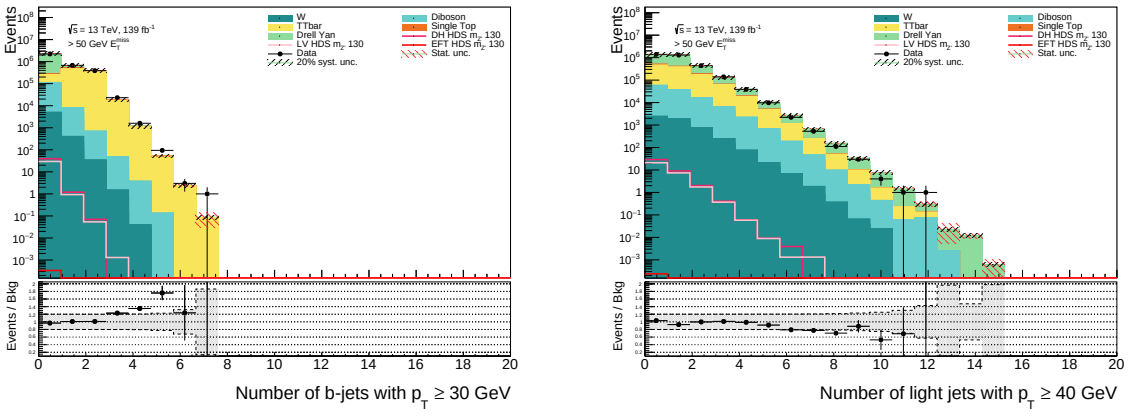


Figure 5.6: b- and light jet distributions with $p_T \geq 30$ and ≥ 40 GeV, respectively.

Other than the object selection criteria, these are the only cuts that will be used to define the control region. Their summary can be seen in Table 5.4

Table 5.4: Table showcasing the cuts used to define the control region for our model independent search.

Feature	Selection criteria
Dilepton final state	$\ell^{\pm}\ell^{\mp}$, $\ell^{\pm}\ell^{\pm}$, $\ell^{\pm}\ell'^{\mp}$ and $\ell^{\pm}\ell'^{\pm}$
Missing Transverse Energy	$E_T^{miss} > 50$ GeV
b- (light) jets	$p_T \geq 30(40)$ GeV

5.5 Feature selection for ML

For this thesis there are many possible kinematic variables that can be used as features for our ML algorithms, in this section we will make use of the kinematical variables presented in Section 3.1.1 as well as introduce new variables that might help our ML algorithm to correctly learn the patters of SM bacgkround and DM signal events.

As we are stuyding a final state with two leptons it is therefore natural to look at the kinematics for both of these as these will give us as general information as possible. The first thing we will look at is the transverse momentum, p_T, of each lepton as defined in Eq. (3.2). We will also look at the azimuthal angle, ϕ and the pseudorapidity, η defined in Eq. (3.8), to know where in the detector the leptons are located. With this we have practically constructed a four-momentumv from which we could learn all particle kinematics. However, we want to help our ML algorithm as much as possible in the task of learning SM background and DM signal. A powerful kinematical varable for this is therefor the invariant mass, m_{ll} defined in Eq. (3.3), which can help the ML algorithm sort out resonant models. Another variable of interest is the trnsverse energy, E_T defined in Eq. (3.4) for the lepton pair. The distribution of the invariant mass can be seen in Figure 5.7.

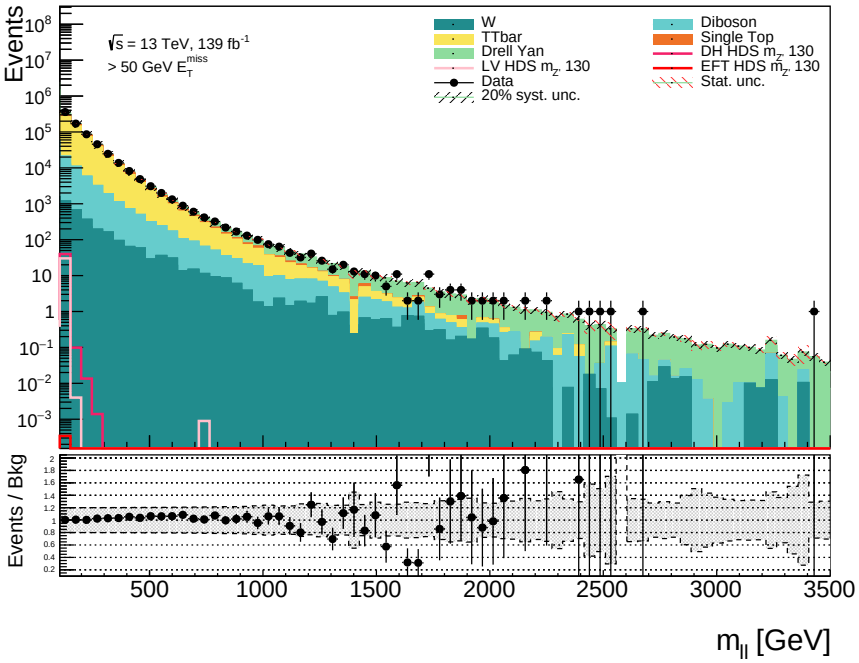


Figure 5.7: Distribution of m_{ll} in control region.

As we are studying DM, an invisible particle, the most important kinematical variable to distinguish background from signal is the missing transverse energy, E_T^{miss} defined in Eq. (3.5), as this how we expect DM to be recorded. Another version of the MET is the so-called *Object-based* E_T^{miss} *significance*, or $E_T^{miss,sig}$ for short, this variable is used tto deal with artifical or fake E_T^{miss}. The way $E_T^{miss,sig}$ works

is by weighing the the value of E_T^{miss} by the precision of its reconstruction. It is defined as

$$E_T^{miss,sig} = \frac{E_T^{miss}}{\sigma(E_T^{miss})} \tag{5.1}$$

where $\sigma(E_T^{miss})$ is the uncertainty of the reconstruction of the E_T^{miss}, which consider the indidual uncerainties of the objects that enter the E_T^{miss} calcuation. The distribution of this variable can ve seen in Figure 5.8. In this thesis we will use both E_T^{miss} and $E_T^{miss,sig}$ even though they are correlated, as the algorithm might chose in different instances[1] which of these is of more importance.

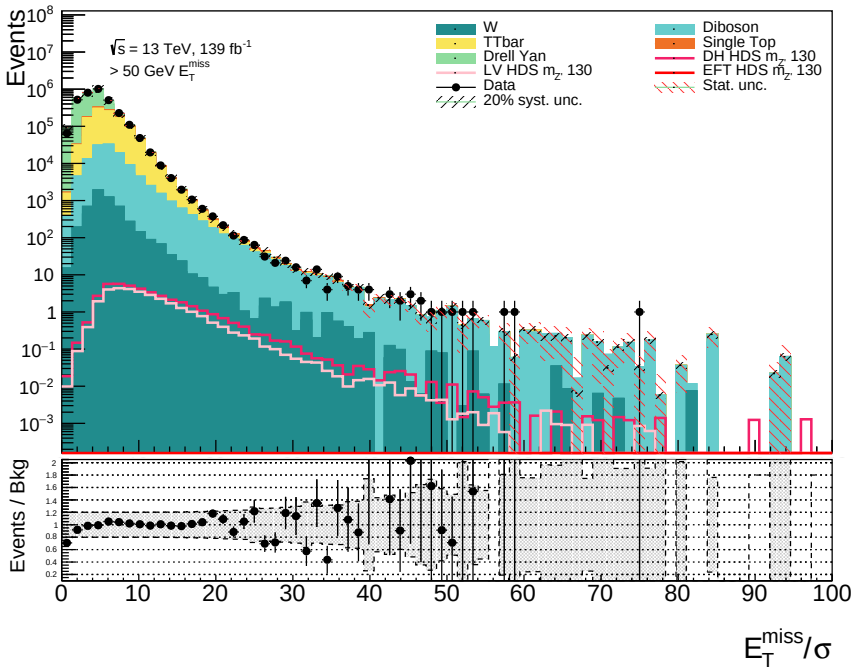


Figure 5.8: Distribution of $E_T^{miss,sig}$ in control region.

To keep the trend of variables for invisible particles, we will also study the transverse mass, m_T defined in Eq. (3.10), and stransverse mass, m_{T2} defined in Eq. (3.11) which might help our ML algorithm to sort i.e. W backgrounds from the signal. The distribution of the stransverse mass can be seen in Figure 5.9. Moving on to more jet related variables, we will look at the p_T, η and ϕ of the three most energetic jets, as these have the best MC and data agreement (see Figure 5.6), the invariant mass of the two most energetic jets m_{jj}, as well as count the number of b- and light jets. Altough these jet variables might not sound like the most useful when singleing out DM signal from SM background, there is always the possibility that our ML algorithm sees a pattern which is overlooked by us when conducting these kind of searches. Another varible we will look at is the hadronic activity, H_T defined in Eq. (3.6), from which we will also look at the ratio between the MET and hadronic activity, E_T^{miss}/H_T, as this can help the ML algorithm sort out DM events from background events.

To know the distance between the "lepton-jet" and "MET-jet" we will look at the difference in azimuthal angle between: the lepton pair, $\Delta\phi(l_1, l_2)$, the dilepton jet and MET jet, $\Delta\phi(ll, E_T^{miss})$, the leading lepton and MET jet, $\Delta\phi(l_l, E_T^{miss})$, and the lepton closest to the MET jet and the MET jet, $\Delta\phi(l_c, E_T^{miss})$. In some of the models we are studying it is expected for DM and the leptons pair to be back to back. The distribution of $\Delta\phi(ll, E_T^{miss})$ can be seen in Figure 5.10.

[1] Meaning different *signal reagions* when conducting a model independent search. See Chapter TO COME

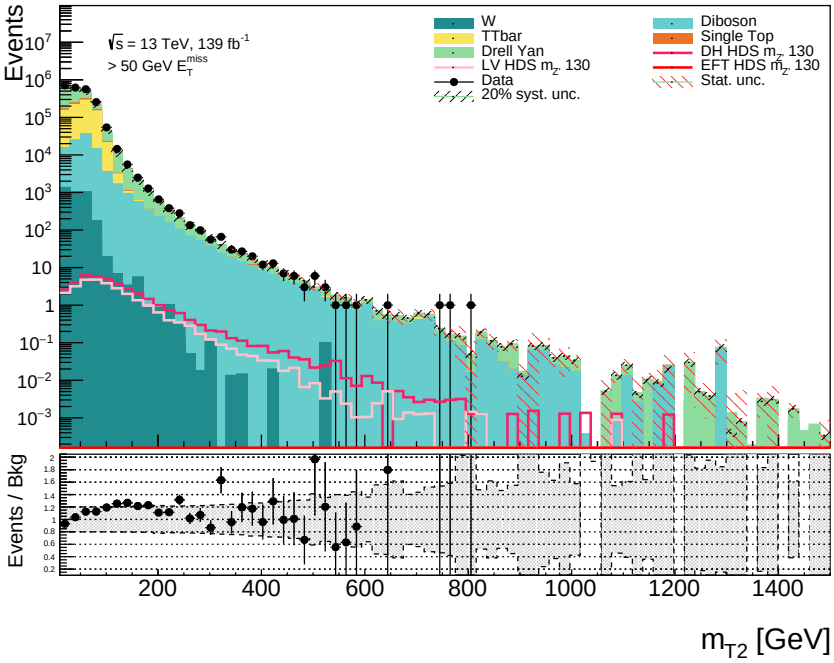


Figure 5.9: Distribution of m_{T2} in control region.

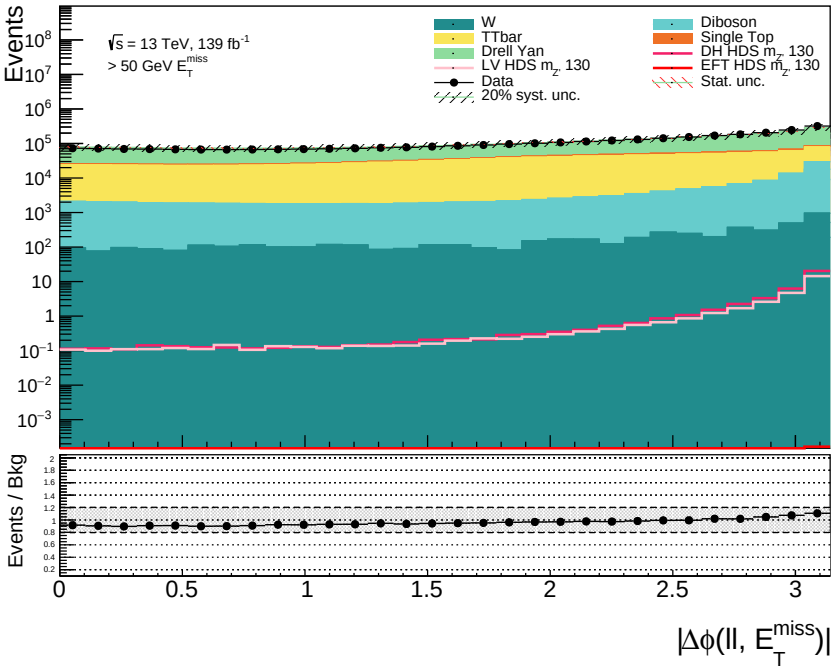


Figure 5.10: Distribution of $\Delta\phi(ll, E_T^{miss})$ in control region.

The kinematic variables used are summarized in Table 5.5 and the distribution of the remaining kinematical variables are shown in Appendix B.

Table 5.5: Table showcasing the kinematic variables that will be used as features on our ML algorithm. * These have poor MC and data agreement. † These create *jagged arrays*.

Kinematic variable	Feature name
p_T of both leptons	lep1pt & lep2pt
ϕ of both leptons	lep1phi & lep2phi
η of both leptons	lep1eta & lep2eta
Invariant mass of dilepton pair, m_{ll}	mll
Missing transverse energy in event, E_T^{miss}	met
Missing transverse energy significance in event, $E_T^{miss,sig}$	met_sig
Transverse mass in in event, m_T	mt
Stransverse mass in in event, m_{T2}	mt2
Transverse energy of lepton pair, E_T	et
ϕ between leption pair, $\Delta\phi(l_1, l_2)$*	dPhiLeps
ϕ between leption pair and MET jet, $\Delta\phi(ll, E_T^{miss})$	dPhiLLMet
ϕ between leading lepton and MET jet, $\Delta\phi(l_l, E_T^{miss})$	dPhiLeadMet
ϕ between closest lepton and MET jet, $\Delta\phi(l_c, E_T^{miss})$*	dPhiCloseMet
Hadronic activity, H_T	ht
Ratio between E_T^{miss} and H_T, E_T^{miss}/H_T	rt
Number of b-jets	nbjets
Number of light jets*	nljets
p_T of three jets with highest p_T†	jet1pt & jet2pt & jet3pt
ϕ of three jets with highest p_T†	jet1phi & jet2phi & jet3phi
η of three jets with highest p_T†	jet1eta & jet2eta & jet3eta
Invariant mass of two jets with highest p_T, m_{jj}†	mjj

5.6 Transfer to ML friendly syntax

We have so far explained how the data will be used and carefully selected, however the question of how the data is made, and the number of samples we will work with still remains unanswered.

As mentioned when defining what our background and signal is, we will mainly look at MC simulations.The MC simulations are made into `ROOT` [34] NTuples which already have passed the object selection criteria on section 5.3. What remains is to set the kinematical cuts so we have our control region. To do this as well as saving the remaining events into a new file which will be used for our ML algorithm I utilized the algorithms on `EventSelector`[2], which also saved the event that passed the event selection crieria as `ROOT` histrograms to make plotting the distributions easier. After saving the events that passed the selection critera I used the algorithms on `DataPrep`[3] to plot the actual distributions of kinematical variables, and more importantly converting all of the events that passed the event selection into a ML friendly syntax. For this thesis we are converting from `ROOT` NTuples to `pandas DataFrame` [35] which can furhtermore be saved as `h5` files to be read more efficiently. The reason we chose `pandas DataFrame` is because of the easily readable kinematics per event, the easily aplicable kinematical cuts to the whole dataset (to more effectively create signal regions), and most importantly because of the compatibility with `XGBoost` [28] and `TensorFlow` [36] which will be the ML packages we will utilize for both BDTs and NNs.

Check version and fix bib DOI

5.7 Problems

There are however still problems when we....

[2] Available here: https://github.com/rubenguevara/Master-Thesis/tree/master/EventSelector
[3] Available here: https://github.com/rubenguevara/Master-Thesis/tree/master/DataPrep

5.7.1 Missing variables

For this theis, as shown in Table 5.5 with the †, we will record events with up to three jets in the final state. However, there isn't always three jets in the recorded events that we will be studying, this creates *jagged arrays* which we can interpret as arrays with missing values. This is an unwanted feature that we need to avoid when training a ML algorithm, and is a problem that is not common in other ML tasks aside from HEPP. To mediate this problem we chose to set the p_T to zero for the missing jets and m_{jj} to zero if there are less than two jets, this is something that is physically reasonable as it doesn't violate any conservation laws. More problematic however is the η and ϕ when there aren't jets. To mediate this I've set the values to -999, which has no physical meaning and is impossible to achieve, this we did so it becomes easier for us to identify the jagged arrays that need different interventions when preparing both NNs and BDTs. The methods we explored to mediate the jagged array problem is further explained in Chapter 6.

5.7.2 MC and data disagreement

There is also another problem,with the final states that are not SFOS, as the MC generated background on these tend to be lower than the recorded data. The number of events that are not SFOS are minimal though, and we think the reason it doesn't fit the data is because we are not including fake leptons.

something more to add?

5.8 Summary

Now that we have explained how the data has been prepared and the problems that arise with this we are ready to start playing around with ML! **Question: Anything else to add to this chapter?**

Chapter 6

Machine Learning

Something something Time to prepare our ML algorithms, statistics etc..

6.1 The datasets

The way we are making datasets in this project, to keep it as close to model independent as possible, will be of the following format

- Make a dataset with all of the SM backgrounds and one DM model.
- Make a dataset with all of the SM backgrounds and all of the DM models, divided into different signal regions.

The idea behind the first one is to start with a model dependent approach, where we train a ML algorithm to learn just one model at a time. The reason for taking this approach and not just having one dataset with all the models included, which would indeed make it a model independent approach, is because of the different phenomenology and experiemntal signatures of the models we are studying. **Question: Should I give concrete examples on how SUSY vs Z' differs or is this sentence enough?** To achieve model indepence with this dataset, the plan is to combine the results of the different networks into one. **Question: Is it correct to say statisticslly combine the results? Or what exactly are we thinking of combining using this method?**

For the second one, even if we were to include models with different signatures it would work out in our favour. For example, if we created Signal Regions (SR) in different regions of the m_{ll} phasespace, *and* trained a ML network in each respective SR, then the network would learn the DM resonance appearing in each area, instead of learning the model resonance. Indeed, achieving model indepence. Furthermore, the plan is to combine the results of each respective region to get an overall view of all of Run II.

In this project we will look at real and simulated data from the ATLAS detector from all running periods of Run II. This is from period a, d and e each with an integrated luminosity of $36.4, 44.3$ and 58.5 fb^{-1} respectively. In Table 6.1 we can see the overall number of MC samples and expected events for all background channels as well as DM models. In addition to using the features in Table 5.5, we will also include the EventID, dataset ID, period, dilepton final state, label telling us wether an event is signal or background and the *weight* of each event.

6.1.1 Weights

The weights are crucial for most of what is to come further on this thesis. The weights only apply to MC simulations and can be interpreted as corrections to simulate real data, which is why the MC samples and expected events on Table 6.1 differ, where the latter has applied weights. The weights on this project are defined as Eq. (6.1) and are in fb^{-1} units.

$$W = w_{mc} \times w_{pu} \times w_{xs} \times w_{lsf} \times w_{nlo,ew} \times w_{ttbar} \times w_{jet} \times w_{bjet} \times \frac{lumi_{period}}{\sum w_{mc,DSID}} \tag{6.1}$$

This equation has a lot of information in it, so every part of this will be discussed. Starting from the raw MC weight, w_{mc}

Table 6.1: Table showcasing the number of MC samples and expected events for every background channel and DM model that will be used in this thesis.
* The expected number of events for any BSM model has prefixed assumptions about for example cross section, decay width, etc. As we do not empirically know if any of these assumptions are correct, these are not set in stone.

Channel	MC samples	Expected events
W	38,684	5,436.325
Drell Yan	47,201,697	2,198,257.648
TTbar	28,126,697	978,530.817
Single top	729,624	93,898.462
Diboson	10,983,689	116,653.466
Standard model total	87,080,391	3,392,776.718
Z' Dark Higgs	1,390,986	*210.286
Z' Light Vector	992,111	*275.518
Z' Effective Field Theory	1,356,372	*0.001

The pile-up weight w_{pu}, takes into account detector featuers
The cross section weight w_{xs}, defined as $w_{xs} := xs \times w_{kf} \times g_{eff}$ where xs is the cross section of the process, w_{kf} is the $k-$factor which has higher order corrections, and g_{eff} is the filter efficiency.
The Lepton and trigger Scale Factor (LSF), w_{lsf}, which...
The Next to Leading Order (NLO) electroweak correction $w_{nlo,ew}$, which ...
The $t\bar{t}$, jet and b-jet weights, w_{ttbar}, w_{jet} and w_{bjet}, which ...
And lastly the luminosity of the period, meaning

$$lumi_{period} = \begin{cases} 36.4, & \text{for } period = a \\ 44.3, & \text{for } period = d \\ 58.5, & \text{for } period = e \end{cases}$$

divided by the Sum Of Weights (SOW) **Question: Sum of every w_{mc} of each event or sum of every W pr event?** of every MC sample for each dataset ID (background process)

6.1.2 Signal regions

Write in more details

Will make three regions:

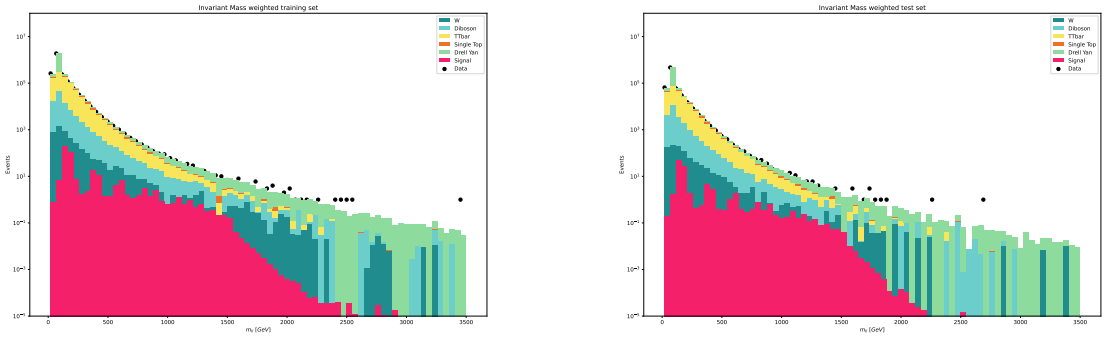
- $m_{ll} > 120$ GeV and $E_T^{miss} \in [50, 100]$ GeV
- $m_{ll} > 120$ GeV and $E_T^{miss} \in [100, 150]$ GeV
- $m_{ll} > 120$ GeV and $E_T^{miss} > 150$ GeV

6.1.3 Train and test split

When training our networks we will use 80% of the dataset, the remaining 20% will be used to test if the networks learning any physical patters or just gets really good at learning the dataset. The only thing that we need to be absolutely certain about when splitting the data is that the distributions follow the same shape and have the same ratio of background channels as well as having good MC and data agreement. In this thesis we will utilize a function from `scikit learn` [37] called `train_test_split` which can easily split our dataset into a training and testing set of our size, while also sorting the label of the event. Another benefit of using this function is that is has included a random seed such that we can always split the datasets in the same form.

add better figure

In Figure 6.1 we can see that the distribution remains the same when we split the dataset into 80-20, where we also see the data and MC agree. To see the distribution of every other kinematical variable see **question: Appendix or github repo? The appendix is already very long!** In addition to splitting the data, when making validation plots (See Chapter 4.3.3), when plotting MC simulations



(a) Training distribution

(b) Testing distribution

Figure 6.1: Train and test distribution of the invariant mass when dividing SM background and DM samples

and real data, we nee to make sure that the luminosity of the plot is correct. That means that if we are plotting the network score of the test dataset, as this is 20% of the whole dataset we need to scale everything with a factor of 5 so we get back to 100% and the full luminosity of Run II.

6.2 Neural Network Training

For this thesis we utilize `TensorFlow v. 2.7.1 GPU` for NN. Before implementing everything into a real NN we need to first prepare the data as explained in Chapter 5. After creating the dataset, the first and most important thing is to marinate the `batch_size` whenever we try **anything** while using the dataset. This is because of both the size of the dataset and because of the imbalance between signal and background, as explained in section 4.1.2 and as seen in Table 6.1.

The highest possible batch size that could be used for this thesis was 2^{24},[1] leading to roughly 17 million samples per batch. This is the best that a dedicated GPU, `NVIDIA A100-PCIE-40GB`, could handle. The batch size also decreases the more complex the NN becomes, as this requires greater computational power.

With this out of the way there are still a few problems to solve to optimize our NN. These are described below.

6.2.1 Padding of data

There are two diffulcuties to overcome when utilizing NNs as compared to BDTs. The the hardest to solve, as reliabe solution seems to exist yet, is the padding of jagged arrays (see Chapter 5.7.1). Padding is the process of filling missing values of a dataset entering a network. This problem doesn't usually appear in other fields than high energy particle physics. To recapitulate what was done in Chapter 5.7.1, the problem arises when the size of some features is not constant. This is the case of number of jets in an event. When creating the dataset the missing values of p_T and m_{jj} were set to zero, while the missing values of η and ϕ were set to -999.

As mentioned before, there is no general consensus of how the padding should be done, and there are many different methods of doing so. The classical data scientist way of solving this problem would be to just take the mean of every feature and use that as a variable for every event with missing values. That means replacing every $p_T, m_{jj} = 0$ and $\eta, \phi = -999$ with the mean of every p_T, m_{jj}, η and ϕ (excluding the 0's and -999's respectively). However this is not popular among physicists since it breaks conservation laws when we say there are jets present in an event when in reality there are none. Another approach is to use Bayesian statistics or ML to estimate the missing values, these options will not be pursued in this thesis due to time, but might be of interest for future projects. Another approach, is setting all the missing values to zero, as this might mean that there isn't anything there, but this also breaks conservation laws since $\eta, \phi = 0$ have physical meaning, this is also highly looked down upon by data scientists since this would affect the weighting when training the network and create a false pattern for the network to follow which would lead to a bias.

The jet p_T and m_{jj} being 0 is a valid form of padding the dataset, as this doesn't break any fundemental law of physics. However setting ϕ as something outside of $[-\pi, \pi]$ doesn't make much sense as this is the angle around the detector. Setting a high value of $|\eta|$ might be physically possible (in the future) but as of today the ATLAS detector has a $|\eta| < 4.9$ (see Chapter 3.2) as the pseudorapidity states how close to the beamline the objects recorded are. However having a p_T of a jet equal to zero while still recording the η and ϕ breaks the laws of physics, so this is a problem that needs to be fixed.

Another approach that will not be pursued in this thesis, would be to just remove all events that have a missing value completely getting rid of the problem, but this reduces the statistics of the dataset which is not desired when searching for new physis, and more importantly completely removes the kinematical distributions that is present in the SM, meaning the NN would learn wrong physics. One could also just remove the features with missing values to conserve statistics, albeit make it harder for the network to see any pattern that we might miss, but this is also not a desirable mitigation.

Instead we have tried to create new kinematic variables that work around the need of padding. These features are just *counting features*, meaning that we count the number of jets that fullgil some criteria, such as the number of b-jets with $p_T > 20$ GeV, the number of light jets with $p_T > 40$ GeV, the number of

[1] It has to be a power of two because of the alignment of virtual processors (VP) onto physical processors (PP) of the GPU. As the number of PP is often a power of 2, using a number of VP different from a power of 2 leads to poor performance.

jets recorded in the central calorimeter ($|\eta| < 2.5$), and the number of jets with $p_T > 50$ GeV recorded in the forward calorimeter ($|\eta| > 2.5$). The p_T cuts are optimized to allow a good agreement between data and simulations, the full distributions with different cuts in the control region can be seen in Appendix D. A summary of these variables is shown in Table 6.2.

RECREATE FIGURES WITH BETTER FORMAT!

Table 6.2: Table showcasing plausible kinematic variables that will not need padding.

Kinematic variable	Feature name
Number of b-jets with $p_T > 20$ GeV	n_bjetPt20
Number of light-jets with $p_T > 40$ GeV	n_ljetPt40
Number of jets recorded in Central calorimeter	n_jetsetaCentral
Number of jets recorded in Forward calorimeter with $p_T > 50$ GeV	n_jetsetaForward50

When training our NN with these new variables inst4ead of dropping features with missing variables we hope that the NN learns more physics by hopefully recognising patters between all high level features.

6.2.2 Normalization of data

Moving onto the second problem, which is not as problematic as the previous one, namely the normalization of data. Since neural networks send a lot of data into multiple neurons and mulitple layers using activation functions and carrying weights and biases that change for every backprogataion iteration, it is important to make sure that a neuron output doesn't die when moving around the network. Meanimng that a neuron output becomes insignificant compared to others when navigating the loss-phasespace. A fast way for neurons to die off is to not normalize the data and send it through the network as it is available. The reason it might die is because we send in numbers which vary significantly to eachother, i.e. the p_T might be as high as thousands GeV, while E_T^{miss}/σ might be as low as 0.1. What might happen when sending such different numbers is that the network might think "obviouly the high number is more important than the low number" thus making the activation function worse for the feature, even though this feature is of high imporatance when looking at MET final states. A way to fix this problem is to normalize all features. There are many ways to do this, one could do *min max scaling* which normalizes every feature from $[0, 1]$, completely solving the problem above. Mathematically speaking this is done by

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{6.2}$$

Where X is the array containing all features, while X_{min} and X_{max} are the lowest and highest values in the said array. Another way to normalize the data is to make the mean of the data 0 and the standard deviation to one, this is called *Z-score normalization*

$$X_{norm} = \frac{X - \bar{X}}{\sqrt{\sigma_X^2}} \tag{6.3}$$

where $\bar{X}$ is the mean of said array and σ_X^2 is the variance. One could also use pre-built functions in `TensorFlow` that provide a normalization, such as `Batch_normalization` which normalizes the data entering the network per batch. This is usually used in Convolutional NN's as it improves computational speed. Another one, `Normalize`, provides the same as Eq. (6.3) for the whole training set going in. This is however computationally heavy to use. The `Layer_normalization`, which normalizes the activations of the previous layer in a batch *independently*, rather than *across* a batch like `Batch_Normalization`. Both `Batch_Normalization` and `Layer_Normalization` use an optimized version of the Z-score when normalizing the data, meaning that all the features take the form of 0 $\pm$1.

There is a big difference when normalizing data ourselves or using `TensorFlow`. `TensorFlow` remembers how the data was normalized when training such that the test data will be normalized the same way, making testing easier. While if we use Eq. (6.2) or Eq. (6.3) ourselves, we have to make sure that we use the same values for $X_{max}, X_{min}, \bar{X}$ or σ_X^2 when normalizing the test data.

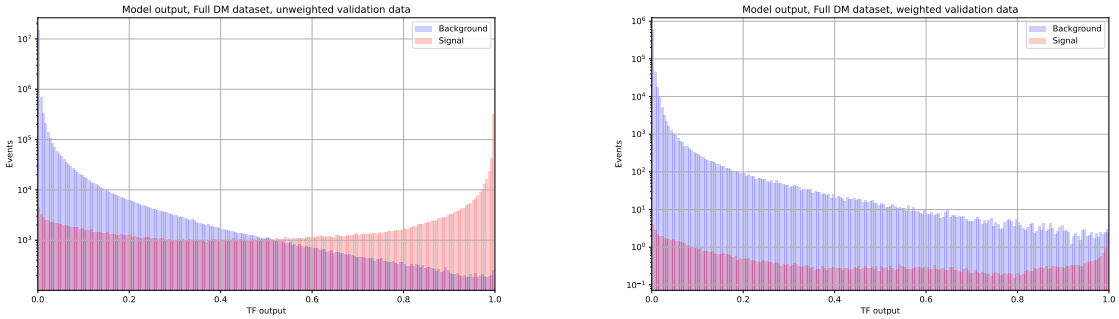
6.2.3 Balancing of signal and background

A big problem that needs to be adressed in this thesis is what we should use as *sample weights* (see Section 4.3.2). If we were to not use any form of sample weights to mitigate the unbalance in our data set it could potentially lead to undertraining where the networks could get "lazy" and guess that everything is background. If we were to split the full Z' data set into a training and test set, where 80% is used for training, and trained a network without any weights then it seems as if the network excels in identifying our signal. This is however misleading as we need to keep in mind two things.

is it called undertraining?

The first one, which **always** will be applied, is that we need to re-weight the MC events the networks trains on to the expected events, this we do by applying the weights explained in Setion ... as well as factoring for the train-test split we made, that means that since our testing set is 20% of the whole data set then we need to multiply this by 5 so we are back at 100%. The difference is shown in Figure 7.4.

will add this later!



(a) Unscaled validation plot.

(b) Scaled validation plot using MC weights.

Figure 6.2: Validation plot of uwweighted network on the first version of the FULL Z' DM dataset.

The second thing, which is more important, is that we are not going to be testing our network on a "full data set", but rather on on one model with fixed parameters. For the purposes of most examples in this chapter, we will conduct our testing on one of the Z' models, more specially it will be a Dark Higgs (DH) with Heavy Dark Sector (HDS) and $m_{Z'} = 130$ GeV. The reason I mention this here, and highlight its importance, is because the data imbalance **will** become a problem when we are studying just one specific model with fixed parameters. To put some numbers, the whole SM background we are studying has roughly 87 million MC events, the whole Z' signal has roughly 3 million MC events, and the DH HDS $m_{Z'} = 130$ GeV model has 40,000 MC events! So if the network were to say that every event is a background event when testing on the small model with fixed parameters, it would be correct over 99.9% of the time. This is obviously something we do not want, as our goal is to make a ML algorithm that actually learns DM signatures.

I trained a network to find the Diboson background channel among the other SM backgrounds to find a general solution to what weights will be used to balance using the sample weight feature, as this is something we know exists. The imbalance of this data set can be seen in Table 6.3.

Table 6.3: Table Showcasing how uneven the training dataset is between signal and background. This is on the Diboson dataset which incorporates all the SM MC samples

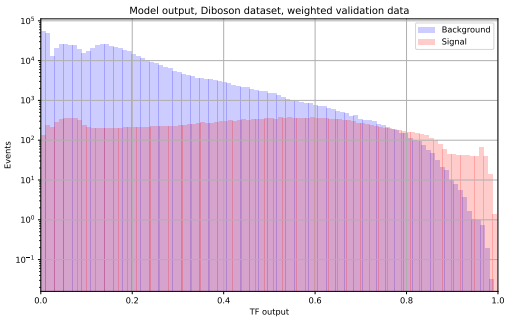
	Number of MC events	Number of expected events	MC events × expected events $[10^{11}]$
Signal	8,813,716	93,304.9	8.2
Background	61,201,010	2,621,498.9	1,604.4

should I write this or skip it entirely?

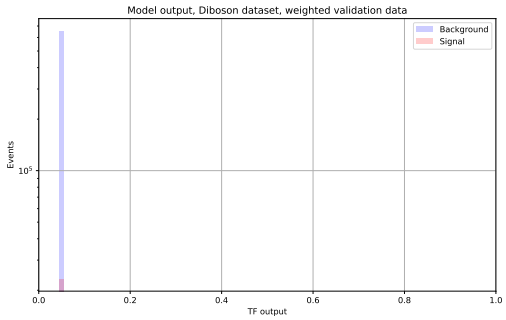
The reason for this choice and not a DM signal, is because I wrongly assumed when first testing the network, that the network *correctly* predicted that there were no DM events. Put in rougher words, I thought that the network completely dismissed the model by claiming it did not exist, and thus excluding the whole model, a very bold, pretentious and wrong claim. As my first attempt to use weights used the

same weights used to re-weight MC events to expected events, which included the cross section of every event... this will become important later.

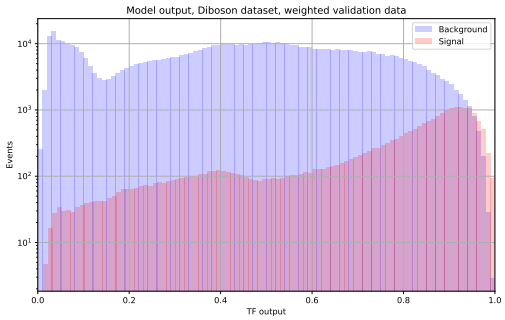
As expected, my interpretation was wrong, the network trained using the weights to re-weight MC events on the Diboson search also predicted that there were "0 Diboson events", something we can empirically and confidently say is wrong. Moreover, the validation plot of the unweighted network works, but gives poor results. As a method to balance the signal and background, I made a new weight array to be used as sample weight. This array weights down all background events with the ratio of MC signal events over MC background events, $\frac{N_{sig}}{N_{bkg}}$. The results are shown in Figure 7.5 and the ROC scores in Figure 7.6.



(a) Validation plot for the unweighted training.



(b) Validation plot for the weighted training using MC weights.



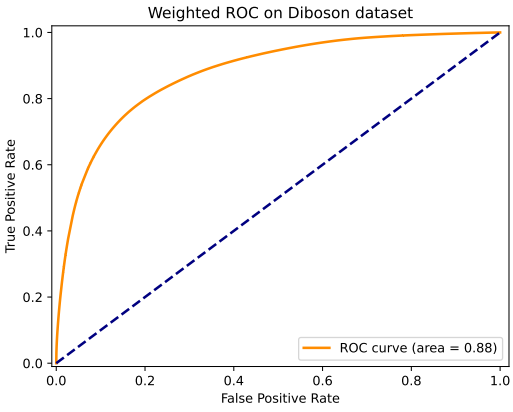
(c) Validation plot for the weighted training using $\frac{N_{sig}}{N_{bkg}}$ as weights on the background.

Figure 6.3: Result of the different network training weighting.

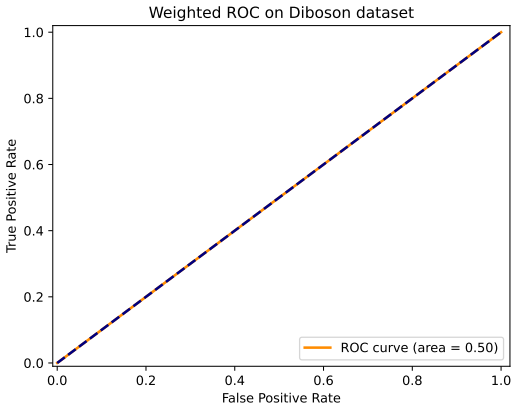
From the results we see that the new weighting method has a poorer result than the unweighted one, and the weighting method that re-weight MC events to expected events just crashes the whole network. The reason for why the latter does not work might be since the network gets punished by guessing "too easily" that an event is signal, and instead of trying harder to learn the patterns and predict with "more confidence" signal events just guesses that everything is a background event and still get a high *accuracy*[2]. Something else to mention, as to why the network does such a poor job at classifying the diboson background, is that the network here was not optimized for the search, and rather just used the defaul NN settings on TensorFlow. And to make matters worse, when testing this I had a bug where I used zero hidden layers by accident, meaning that there was in fact no NN.

mention this or remove completely

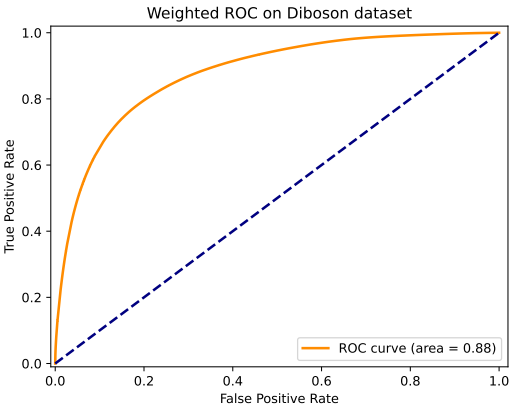
[2]Not to be confused with AUC!



(a) ROC score for the unweighted training.



(b) ROC score for the weighted training using MC weights.

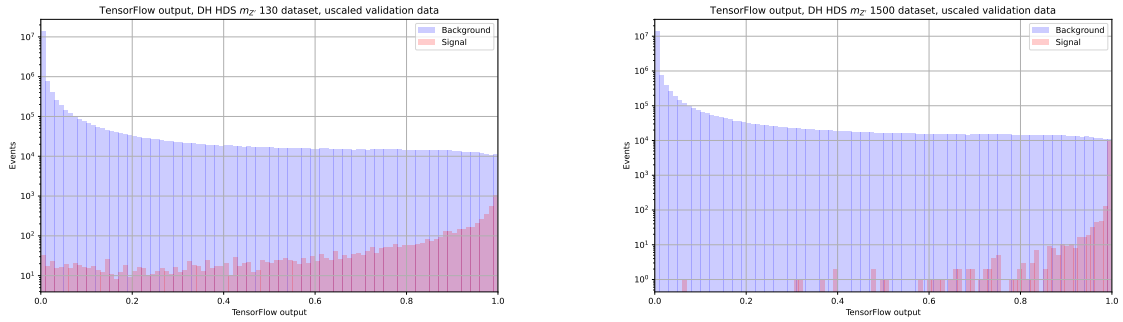


(c) ROC score for the weighted training using $\frac{N_{sig}}{N_{bkg}}$ as weights on the background.

Figure 6.4: Result of the different network training weighting.

6.2.4 Sample weights to get expected events

Even if the weighting method previously described helps the network to give reasonable results. It most likely won't distinguish the importance between signal events that have high resonance and those that do not. As an example, if we look at the Z' DH model in the HDS and compare how the network classifies a model with a $m_{Z'}$ of 130 GeV to one with a $m_{Z'}$ of 1500 GeV we might get an idea of how the network classifies things. We can see how the network predicts wether an event is signal or background using validation plots, this is seen in 7.7.

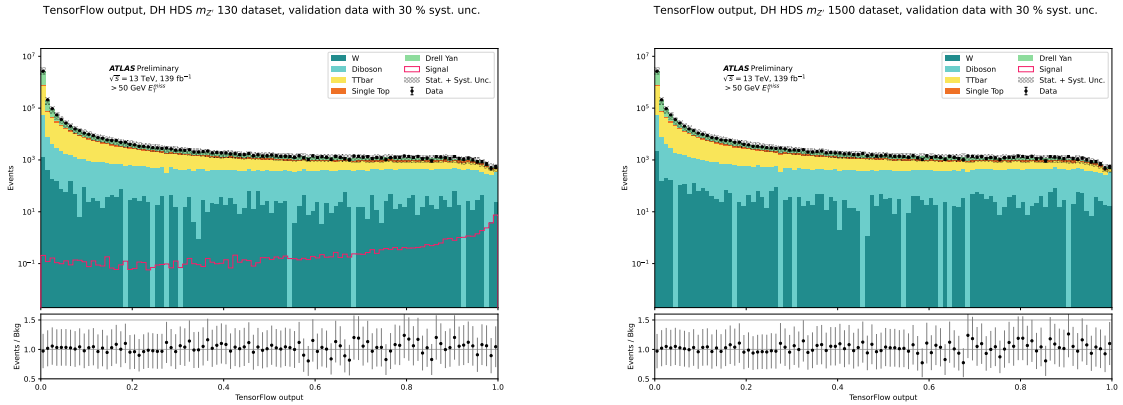


(a) NN prediction of DH HDS $m_{Z'} = 130$ GeV (b) NN prediction of DH HDS $m_{Z'} = 1500$ GeV

Figure 6.5: NN comparison of trained models in FULL Z' dataset

The result purely demonstrate the raw network output, meaning that the validation dataset has not been scaled up using MC weights (i.e. model cross section) or luminosity.

We can see the network is better at classifying the events from the model with $mZ' = 1500$ GeV than the other, even if this model has a much lower cross section, as seen in the correctly scaled plots below.



(a) NN prediction of DH HDS $m_{Z'} = 130$ GeV (b) NN prediction of DH HDS $m_{Z'} = 1500$ GeV

Figure 6.6: Correctly scaled plots.

As we can clearly see, the better predicted model by the network does not even have one event when scaled up correctly. Therefore it is desirable to both take into account the data imbalance between the signal and background as well as the MC weights when training a network. To do this using TensorFlow we could make use of two parameters when training the network: `class_weight` and `sample_weight`.

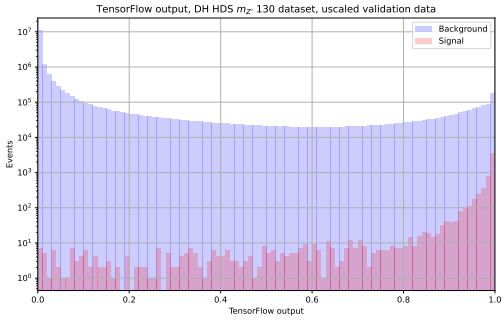
`class_weight` works as a dictionary that weights events that are keys on the dictionary. For our purposes we can make a dictionary where we weight signal and background events differently, this is the same type of scaling that was done in the previous section. `sample_weight` takes in individual weights for every single event that goes into the netwrok, meaning that it is crucial that we know that the desired weight matches the desired event. Ideally we would use both weighting methods, `class_weight` to balance the signal to background ratio and `sample_weight` with the MC weights. However there is

a bug in TensorFlow (up to version `GPU 2.7.1`) that makes it so the program doesn't run when using both parameters. This is not a big problem though, as when looking at the source code one can see that what TensorFlow does with both weights is multiply them togheter. Thus we could try to make use of this "balaned weighting" method to see if the network learns the different models better.

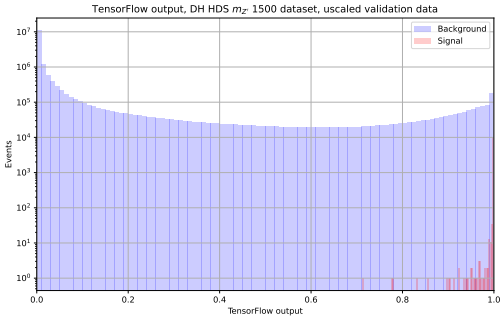
We can see in Table 7.1 how uneven the data (FULL Z') is. To balance the data we have four options. We can either *weigh up the signal* events by the ratio of background over signal. Or *weigh down the bacgkround* with the ratio of signal over background. However this ratio might differ a lot when using the MC raw event ratio or the SOW events ratio. I have tested all four possibilities and these can be see below.

Table 6.4: Table showcasing the data imbalance between background and signal in both raw MC events and actual events. By actual events it is meant weighted events. This is for the training dataset on the FULL Z' dataset.

	Actual events	MC events
Background	2,715,280.4	69,664,290
Signal	388.4	2,991,598

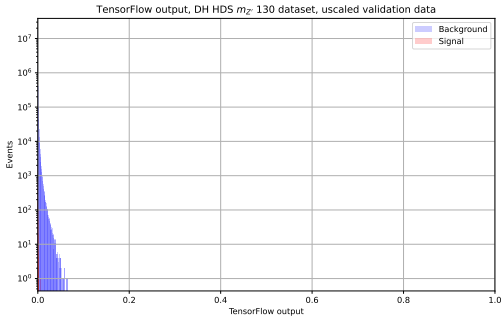


(a) NN prediction of DH HDS $m_{Z'} = 130$ GeV

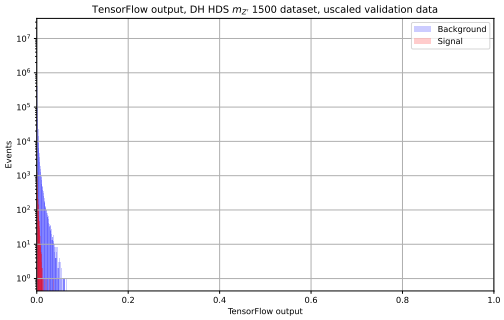


(b) NN prediction of DH HDS $m_{Z'} = 1500$ GeV

Figure 6.7: NN prediction when weighting the signal with the SOW ratio.

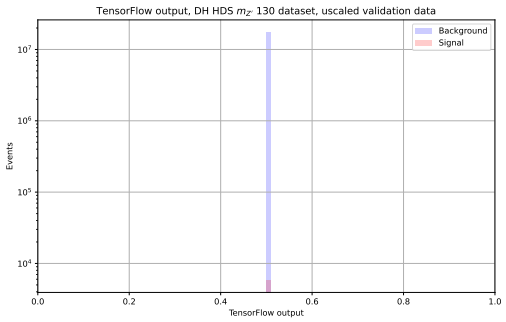


(a) NN prediction of DH HDS $m_{Z'} = 130$ GeV

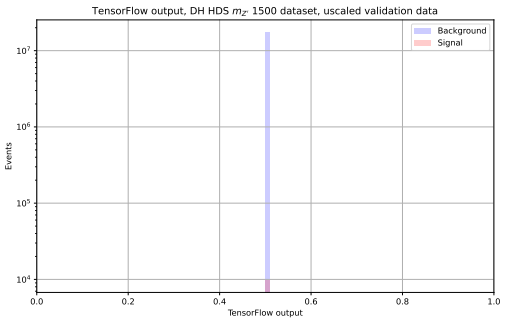


(b) NN prediction of DH HDS $m_{Z'} = 1500$ GeV

Figure 6.8: NN prediction when weighting the signal with the raw event ratio.

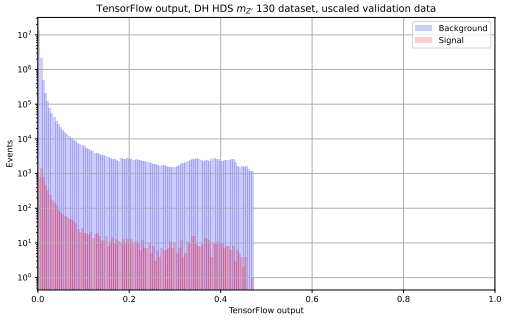


(a) NN prediction of DH HDS $m_{Z'} = 130$ GeV

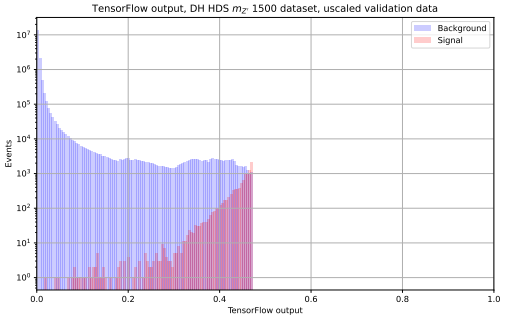


(b) NN prediction of DH HDS $m_{Z'} = 1500$ GeV

Figure 6.9: NN prediction when weighting the background with the SOW ratio.



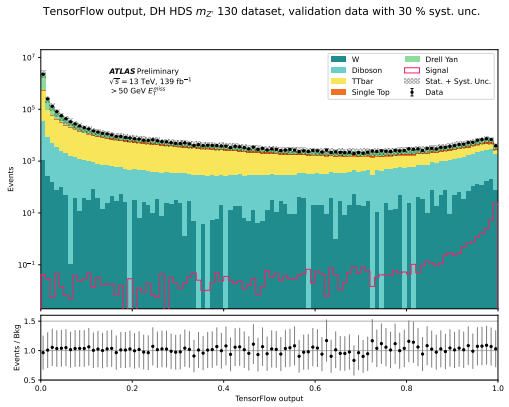
(a) NN prediction of DH HDS $m_{Z'} = 130$ GeV



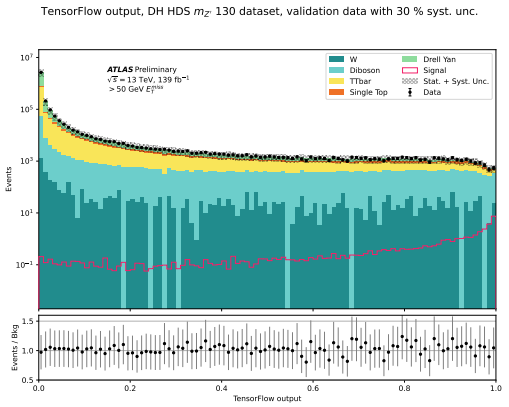
(b) NN prediction of DH HDS $m_{Z'} = 1500$ GeV

Figure 6.10: NN prediction when weighting the bacgkround with the raw event ratio.

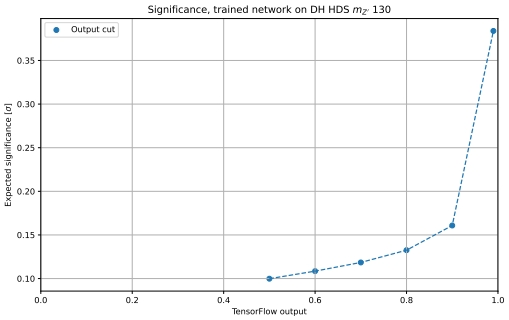
As we can see almost every way of doing this balanced weighting gives poor results. The only one that seems to work is when we we scale up the MC weights with the ratio between the SOW of the bacgkround over signal, however it does not seem to learn the model with lower mass better compared to how it learned the model with higher mass. It seems to have generally learned both models better, which might be a hint that it learns other models worse, i.e. EFT models with much lower cross section. If we now compare the correctly scaled validation plots of DH HDS $m_{Z'} = 130$ GeV.



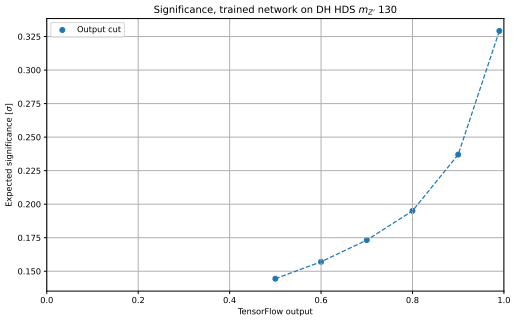
(a) MC weights + scaling signal up with SOW ratio



(b) Only using raw MC events weigh down bacgkround



(c) The expected significance of a)



(d) The expected significance of b)

Figure 6.11: Comparison of the best balanced weighting method to the weighting method of the previous section. Figure a) and b) show the validation data of both cases, c) and d) show the expected significance of the validation plots when making a cut on the output.

From this we can see that the the network that trained when balancing the data is better at classifying the DH HDS $m_{Z'} = 130$ GeV model correctly as signal than the other network. We also observe that it wrongly classifies more background as signal than the other network, but even if this is the case the expected significance is greater in the new network. The reason that the new network wrongly classifies more background as signal might be becuse I utilized the same hyperparameters when training both newtworks, when it might be better to use different hyperparameters on each. Another thing that might be of significance is that I used the ratio of the training set, it might make a difference (better or worse) to use the ratio of the full dataset when training as this is more general. Since the ratio most likely differs for the training and testing set.

Should I add here that this will not be pursued further? Or rather include it on the "results" part of the thesis

6.2.5 Architecture

The architecture of the NN utilized in this project is of the form shown in Figure 6.12.

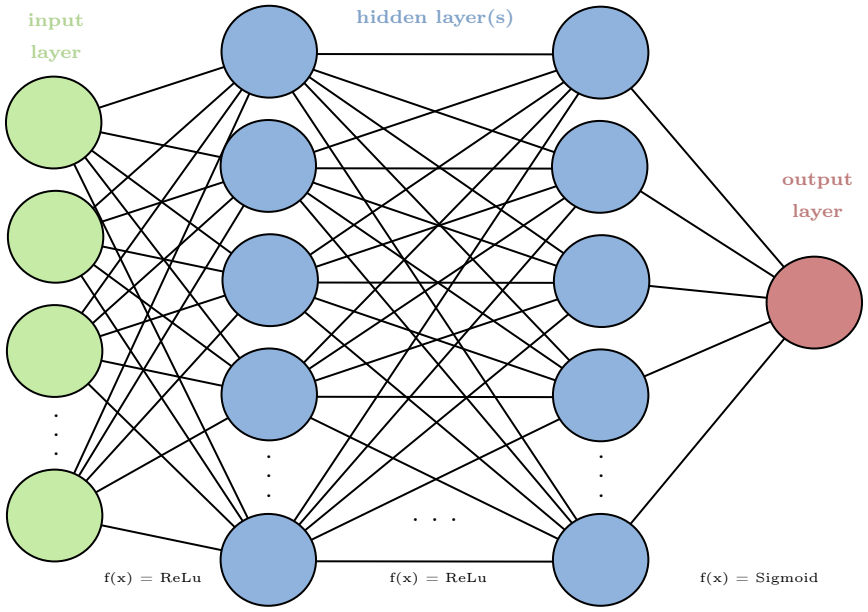


Figure 6.12: Architecture of the NN used on this thesis. The neurons on the hidden layer(s) is a hyperparameter, as well as the number of hidden layers.

Making this type of NN using TensorFlow is easy. An algorithm showing one of the possibilites can be seen in Algorithm 6.1.

Algorithm 6.1: Neural network definition using TensorFlow

```
import tensorflow as tf
from tensorflow.keras import layers

def Neural_Network(inputsize, n_layers, n_neuron, eta, lamda):

    model=tf.keras.Sequential()

    for i in range(n_layers):
        if (i==0):
            model.add(layers.Dense(n_neuron, activation='relu', kernel_regularizer=
             tf.keras.regularizers.l2(lamda), input_dim=inputsize))
        else:
            model.add(layers.Dense(n_neuron, activation='relu', kernel_regularizer=
                    tf.keras.regularizers.l2(lamda)))

    model.add(layers.Dense(1,activation='sigmoid'))

    sgd=tf.optimizers.Adam(learning_rate=eta)

    model.compile(loss=tf.losses.BinaryCrossentropy(),
                optimizer=sgd,
                metrics = [tf.keras.metrics.BinaryAccuracy()])
    return model

```

6.2.6 Grid Search

To get the best performance on our NN, we need to find which hyperparemters helps the network reach highest significance. To do this, we need to do a gridsearch. For our neural network we will mainly focus on four hyperparameters explained on section 4.1:

- The learning rate η
- The L2-regressor variable λ
- The number of neurons on each hidden layer `n_neuron`
- Possibly the number of layers `n_layers`, exluding the output. (NB! Meaning that `n_layers` = 1 means no hidden layer!)

The metrics that will be used to estimate the best hyperparameters are **AUC**, **binary accuracy** and most importantly **expected significance**. The expected significance for this section has been calculated using the low statistics formula Eq. (8.1) just in case there is too few events after the network prediction. The expected significance will also be calculcated when making a cut on 0.85 on the netowrk prediction, meaning only looking at events which the network rates as signal with 85% confidence and above.

The dataset on which I've trained so far is the FULL Z' DM dataset including DH, LV and EFT. The raw number of events on this dataset is roughly 3 million signal events to 70 million background events, I have however split this into a 80% training set and 20% testing set. ***The reason for doing this, is so we get the best hyperparameters for doing a model independant search.***

6.3 Boosted Desicion Tree Training

When working with BDTs we do not run into as many problems as we do with NNs. For example the padding and normalization of data can be completely avoided, making the whole procedure a lot easier when one uses "weird data" as we do in HEPP. There is however one other problem that need to be dealt with when working with this type of ML, this is the weights, which will be discussed in the next section.

For this project we will as mentioned utilize the Extreme Gradient Boosting, or XGBoost for short, package made for the HiggsML whenever we mention BDTs. This project utilized version `1.5.0` without GPU adaptability. XGBoost also helps to avoid padding as it is integrated with a `missing_variable` variable where we can simply write the number of the variable that is missing.

remember to mention it

add source/remove this comment?

6.3.1 Sample weights

For XGBoost there is a different problem when it comes to weights. XGBoost has a variable called `scale_pos_weight` where we can help the network deal with unbalanced data, such as the one we have. Meaning that the whole problem of section **??** completely dissapears, meaning we could use the *real* weights calculated in the MC generators. Sadly it is not that easy, XGBoost does not have to possibility to include negative weights, which this dataset has a few of. Some MC generators generate events with negative weights, such as Sherpa , that take into acount higher order diagrams and needs to add negative weights to "counter" the overcounting of diagrams [38], which are important to correctly scale the simulated events to real data. In the future this might no longer be a problem as future MC generators might only have positive weights.

cite sherpa?

I have no other sources to this other than spoken words at conferenes...

A method to mitigate this problem is to use the absolute value of the weights when training to solve, or rather avoid, this problem. This is however not generally accepted as a solution, and some even say it should be avoided. There are other options however, one is to not include events with negative weights on the training set.

Another one that has been used on a published ATLAS (internal) article (chapter 9.3) [39] is to normalize the weights when using the absolute value with respect to the sum of weights over the sum of absolute weights. The reason behind this is because the sum of weights is obviously not the same when we take the absolute value. Mathemathically speaking, if we have an array of weights W, we can update this like

$$W \to |W| \frac{\sum_i W_i}{\sum_i |W_i|} \tag{6.4}$$

such that the weights are at least in the same scale.

6.3.2 Architecture

Making a BDT for our purposes is fairly easy as well using XGBoost. One way to do it is using the code below in Algorithm 6.2

Algorithm 6.2: Boosted Decision Tree definition using XGBoost

```
import xgboost as xgb

Boosted_Decision_Tree = xgb.XGBClassifier(
                max_depth,
                use_label_encoder=False,
                n_estimators,
                learning_rate,
                reg_lambda,
                predictor = 'cpu_predictor',
                tree_method = 'hist',
                scale_pos_weight=sow_bkg/sow_sig,
                objective='binary:logistic',
                eval_metric='auc',
                missing=10,
                random_state=42,
                verbosity = 1)
```

6.3.3 Grid Search

To get the best performance on our BDT we have to do a grid search here as well. The trainable hyperparameters here are different than for NNs though. With XGBoost we have the following hyperparameters

- Tree depth: how many times we split the data
- Number of estimators: how many estimators we use to do out gradient boosting
- The learning rate η
- L2-regressor λ, to stop overtraning

The data set used for this grid search is the same one used in the NN grid search. That is the FULL Z' DM data set containing DH, LV and EFT models. The metrics used to evaluate the score for this case was again the **AUC** and **expected significance**. One thing to note, for this section I did not include the weighting method showcased in the previous section, I only used the data balancing method, as this is its intended purpose.

At first I set the number of estimators to be 120 and fixed the L2 λ to 10^{-5}. Then conducted a grid search on the tree depth and learning rate only. I used the same range on the learning rate as I did with the NNs, $\eta \in [0.001, 0.01, 0.1, 1]$, and first did a treee depth from 3-6. This showed however a trend hinting to the expected significance only increasing with more depth, because of this I ended up continuing the grid search up to a depth of 30.

Part III

Results

Chapter 7

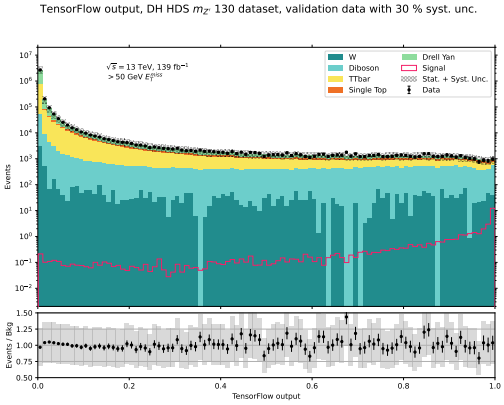
Network opimization

Something something Time to prepare our ML algorithms, statistics etc..

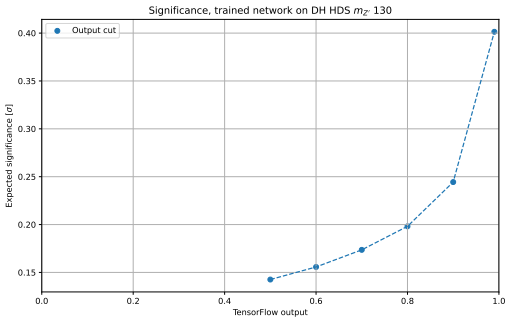
7.1 Neural Network Training

7.1.1 Padding of data

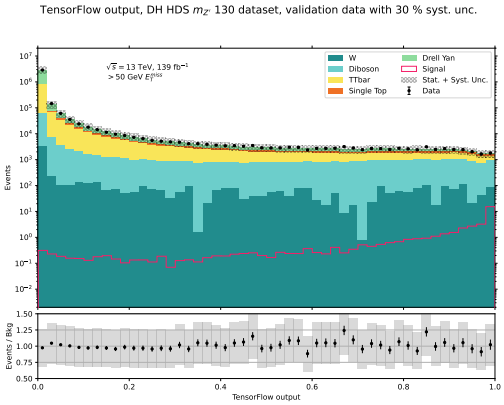
I have however tested this and the difference in a optimized network (more info on this later) is minimal, and depending on our binning choice might make these new features less suited for our task. The resulta using 100 bins and using 50 bins can be seen in Figure 7.1. For the remainder of our NN endevaours we will opt to not use these new features, and just remove the features that have missing variables all togheter.



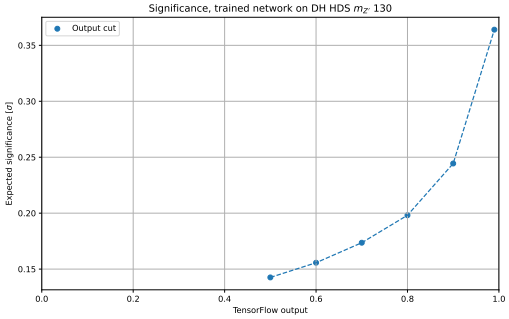
(a) When including new variables and using 100 bins



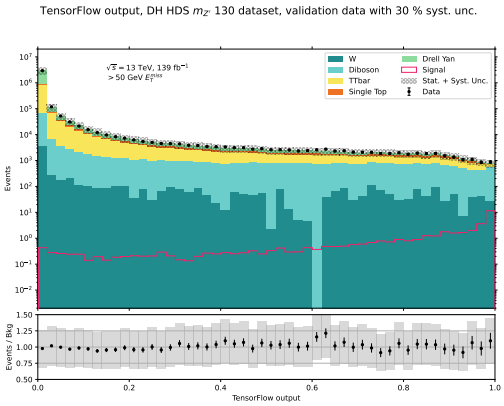
(b) Expected significance of a)



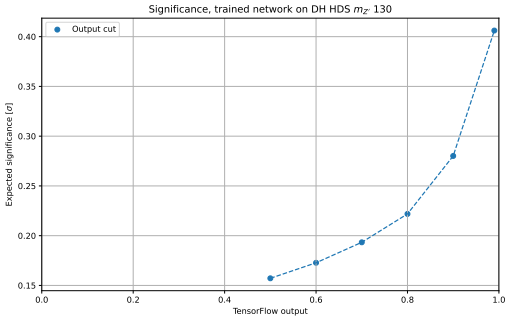
(c) When including new variables and using 50 bins



(d) Expected significance of c)



(e) When excluding new variables and using 50 bins



(f) Expected significance of e)

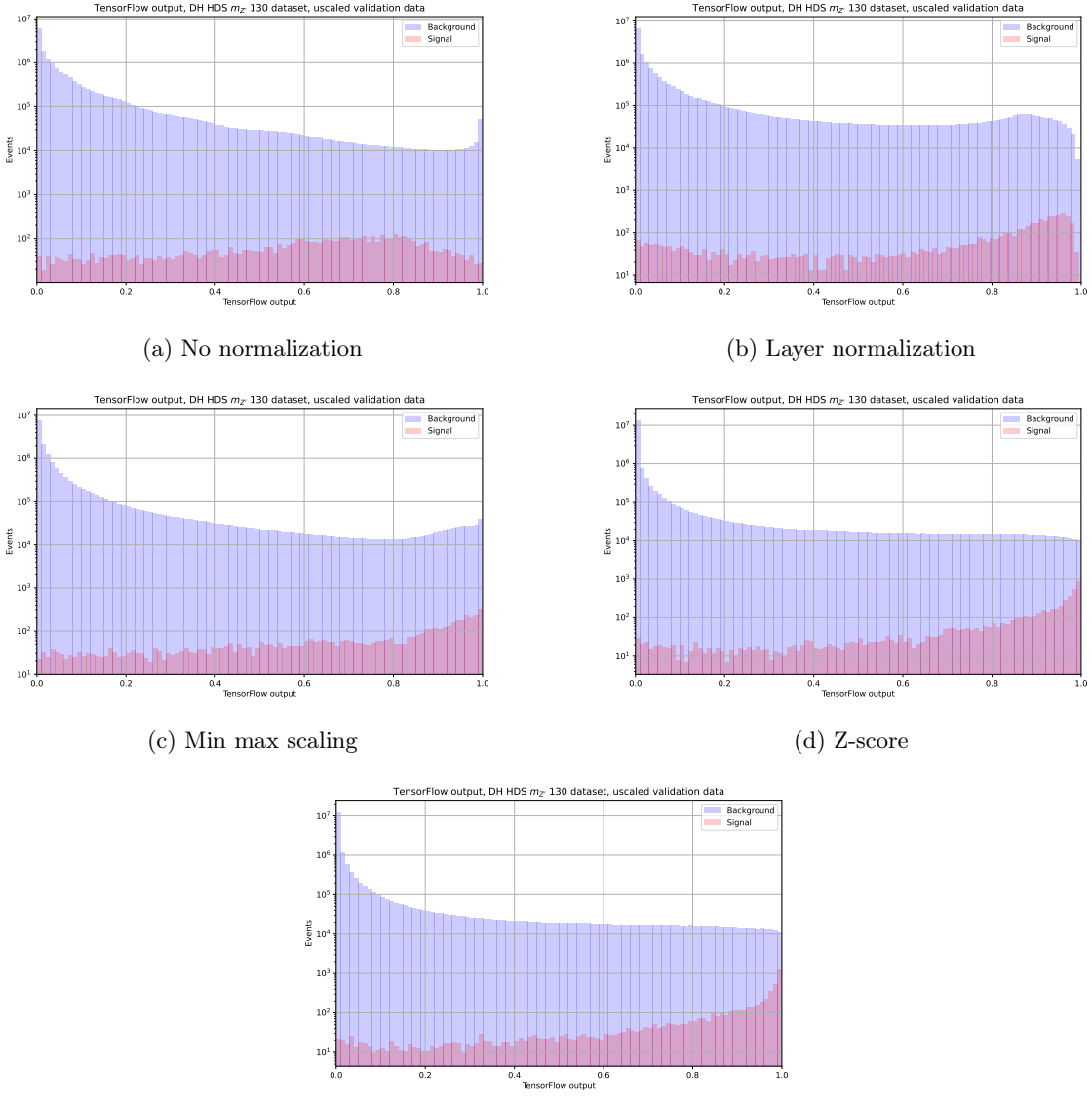
Figure 7.1: NN prediction when using new features to avoid padding. This is testing a dataset with a Z' DM model.

7.1.2 Normalization of data

The different normalization methods have been tested and can be seen in the Figure 7.2. The best results come from Z-score and Batch normalization. The expected significance differs in both cases shown in Figure 7.3.

From these results it is clear that the best normalization method is Batch normalization, but is it reasonable to use this method when one is not using a CNN? The reason batch normalization might work best for our case is because when we divide the data by bacthes it might unevenly represent the SM / signal and their ratio. But by using batch normalization it takes the average of all the batches creating an closer to real distribution. For the following examples in this section I have used the Z-score method, but I will use Batch normaliazation for the signal search.

should I even mention this if I will not use NNs further?



(a) No normalization

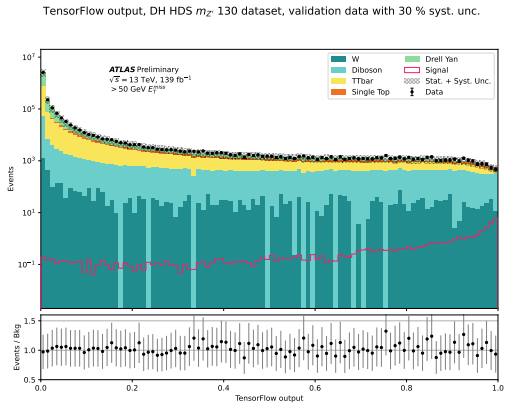
(b) Layer normalization

(c) Min max scaling

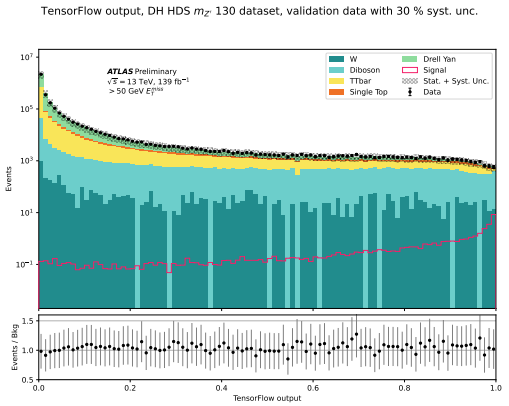
(d) Z-score

(e) Batch normalization

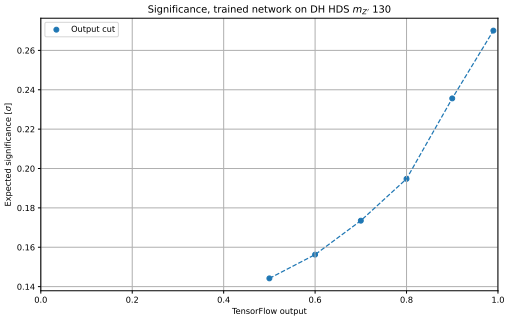
Figure 7.2: NN prediction when using different normalization methods. This is testing a dataset with a Z' DM model.



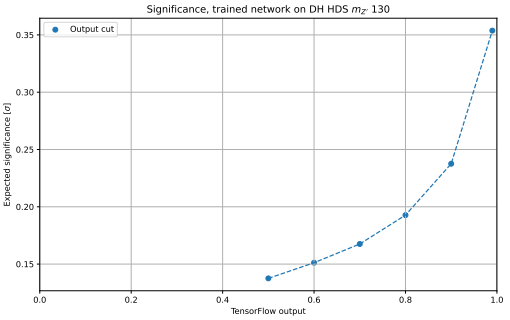
(a) Z-score



(b) Batch normalization



(c) The expected significance of a)



(d) The expected significance of b)

Figure 7.3: Comparison of the best normalization methods. Figure a) and b) show the validation data of both cases, c) and d) show the expected significance of the validation plots when making a cut on the output.

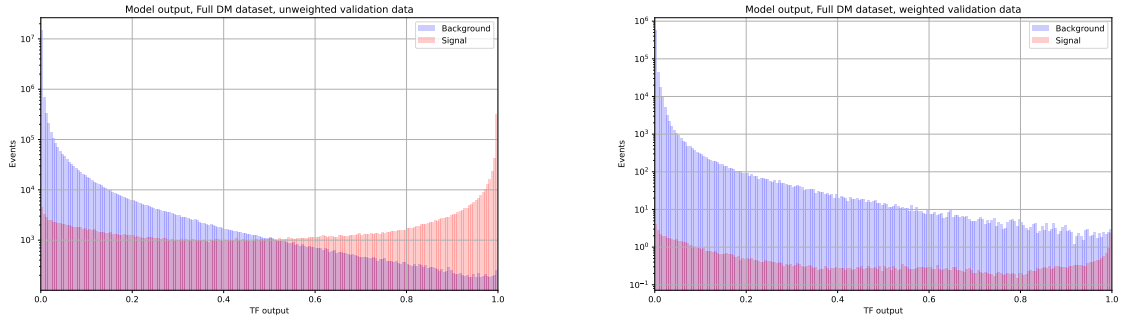
7.1.3 Balancing of signal and background

A big problem that needs to be adressed in this thesis is what we should use as *sample weights* (see Section 4.3.2). If we were to not use any form of sample weights to mitigate the unbalance in our data set it could potentially lead to undertraining where the networks could get "lazy" and guess that everything is background. If we were to split the full Z' data set into a training and test set, where 80% is used for training, and trained a network without any weights then it seems as if the network excels in identifying our signal. This is however misleading as we need to keep in mind two things.

is it called undertraining?

The first one, which **always** will be applied, is that we need to re-weight the MC events the networks trains on to the expected events, this we do by applying the weights explained in Setion ... as well as factoring for the train-test split we made, that means that since our testing set is 20% of the whole data set then we need to multiply this by 5 so we are back at 100%. The difference is shown in Figure 7.4.

will add this later!



(a) Unscaled validation plot.

(b) Scaled validation plot using MC weights.

Figure 7.4: Validation plot of uwweighted network on the first version of the FULL Z' DM dataset.

The second thing, which is more important, is that we are not going to be testing our network on a "full data set", but rather on on one model with fixed parameters. For the purposes of most examples in this chapter, we will conduct our testing on one of the Z' models, more specially it will be a Dark Higgs (DH) with Heavy Dark Sector (HDS) and $m_{Z'} = 130$ GeV. The reason I mention this here, and highlight its importance, is because the data imbalance **will** become a problem when we are studying just one specific model with fixed parameters. To put some numbers, the whole SM background we are studying has roughly 87 million MC events, the whole Z' signal has roughly 3 million MC events, and the DH HDS $m_{Z'} = 130$ GeV model has 40,000 MC events! So if the network were to say that every event is a background event when testing on the small model with fixed parameters, it would be correct over 99.9% of the time. This is obviously something we do not want, as our goal is to make a ML algorithm that actually learns DM signatures.

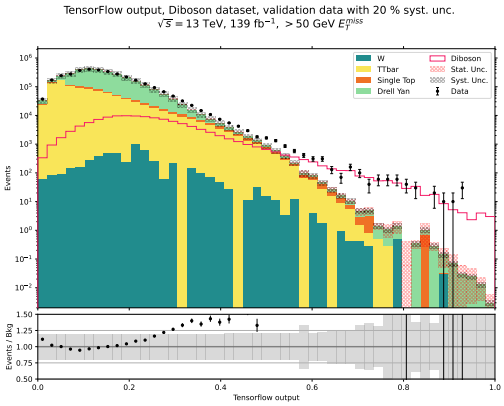
I trained a network to find the Diboson background channel among the other SM backgrounds to find a general solution to what weights will be used to balance using the sample weight feature, as this is something we know exists. The imbalance of this data set can be seen in Table 6.1.

The reason for this choice and not a DM signal, is because I wrongly assumed when first testing the network, that the network *correctly* predicted that there were no DM events. Put in rougher words, I thought that the network completely dismissed the model by claiming it did not exist, and thus excluding the whole model, a very bold, pretentious and wrong claim. As my first attempt to use weights used the same weights used to re-weight MC events to expected events, which included the cross section of every event... this will become important later.

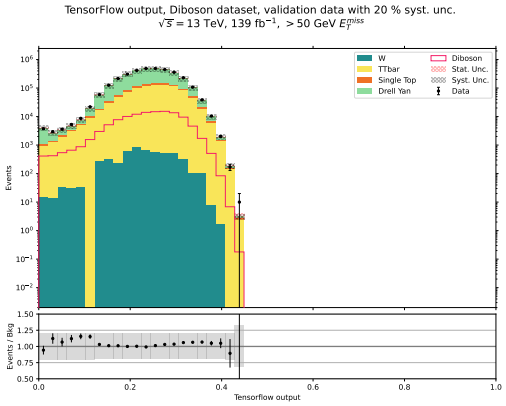
should I write this or skip it entirely?

As expected, my interpretation was wrong, the network trained using the weights to re-weight MC events on the Diboson search also predicted that there were "0 Diboson events", something we can empirically and confidently say is wrong. Moreover, the validation plot of the unweighted network works, but gives poor results. As a method to balance the signal and background, I made a new weight array to be used as sample weight. This array weights down all background events with the ratio of MC signal

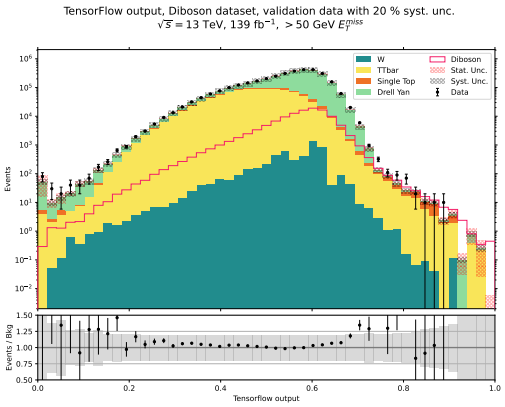
events over MC background events, $\frac{N_{sig}}{N_{bkg}}$. The results are shown in Figure 7.5 and the ROC scores in Figure 7.6.



(a) Validation plot for the unweighted training.



(b) Validation plot for the weighted training using MC weights.



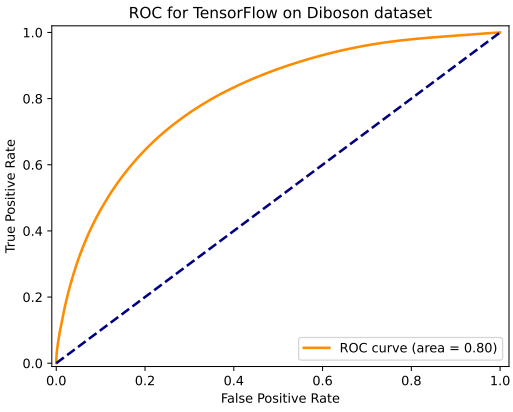
(c) Validation plot for the weighted training using $\frac{N_{sig}}{N_{bkg}}$ as weights on the background.

Figure 7.5: Result of the different network training weighting.

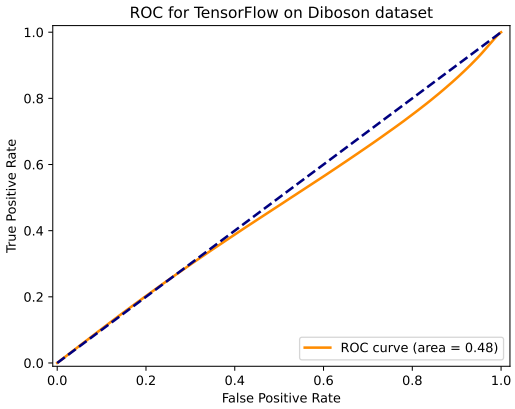
From the results we see that the new weighting method has a poorer result than the unweighted one, and the weighting method that re-weight MC events to expected events just crashes the whole network. The reason for why the latter does not work might be since the network gets punished by guessing "too easily" that an event is signal, and instead of trying harder to learn the patterns and predict with "more confidence" signal events just guesses that everything is a background event and still get a high *accuracy*[1]. Something else to mention, as to why the network does such a poor job at classifying the diboson background, is that the network here was not optimized for the search, and rather just used the defaul NN settings on TensorFlow. And to make matters worse, when testing this I had a bug where I used zero hidden layers by accident, meaning that there was in fact no NN.

mention this or remove completely

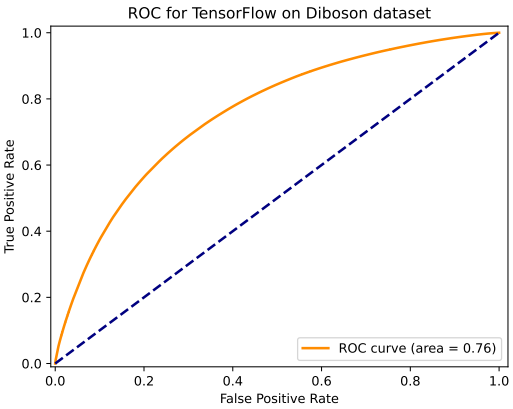
[1] Not to be confused with AUC!



(a) ROC score for the unweighted training.



(b) ROC score for the weighted training using MC weights.

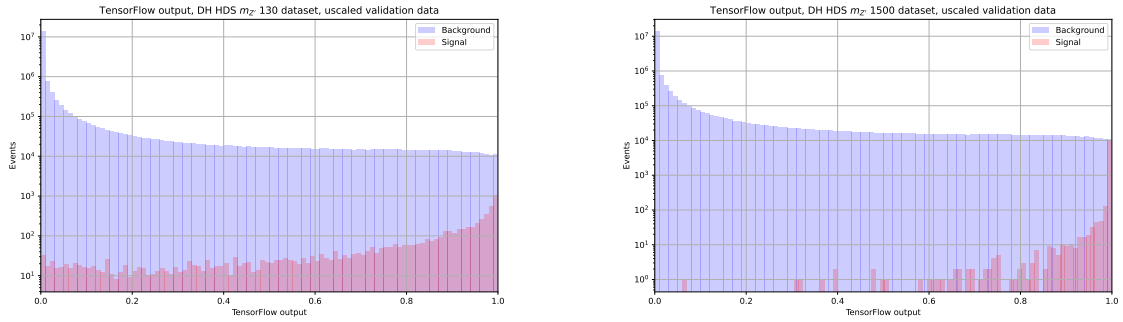


(c) ROC score for the weighted training using $\frac{N_{sig}}{N_{bkg}}$ as weights on the background.

Figure 7.6: Result of the different network training weighting.

7.1.4 Sample weights to get expected events

Even if the weighting method previously described helps the network to give reasonable results. It most likely won't distinguish the importance between signal events that have high resonance and those that do not. As an example, if we look at the Z' DH model in the HDS and compare how the network classifies a model with a $m_{Z'}$ of 130 GeV to one with a $m_{Z'}$ of 1500 GeV we might get an idea of how the network classifies things. We can see how the network predicts wether an event is signal or background using validation plots, this is seen in 7.7.

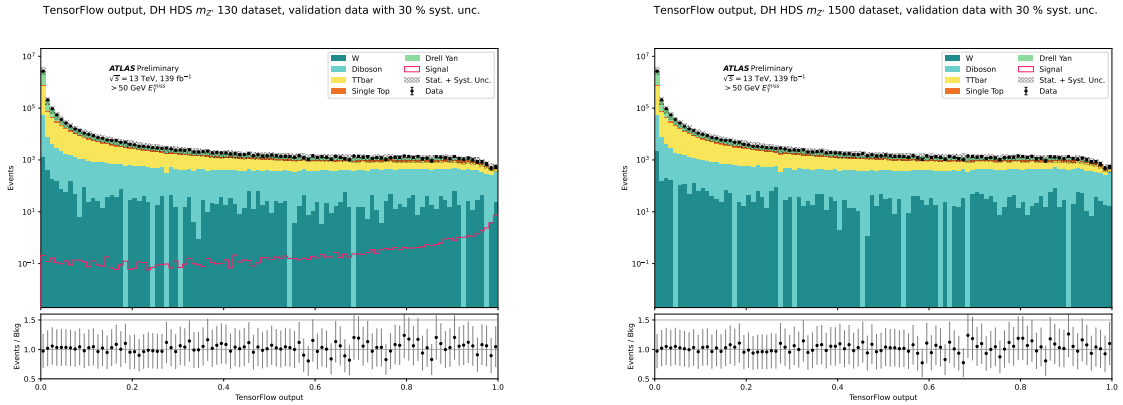


(a) NN prediction of DH HDS $m_{Z'} = 130$ GeV (b) NN prediction of DH HDS $m_{Z'} = 1500$ GeV

Figure 7.7: NN comparison of trained models in FULL Z' dataset

The result purely demonstrate the raw network output, meaning that the validation dataset has not been scaled up using MC weights (i.e. model cross section) or luminosity.

We can see the network is better at classifying the events from the model with $mZ' = 1500$ GeV than the other, even if this model has a much lower cross section, as seen in the correctly scaled plots below.



(a) NN prediction of DH HDS $m_{Z'} = 130$ GeV (b) NN prediction of DH HDS $m_{Z'} = 1500$ GeV

Figure 7.8: Correctly scaled plots.

As we can clearly see, the better predicted model by the network does not even have one event when scaled up correctly. Therefore it is desirable to both take into account the data imbalance between the signal and background as well as the MC weights when training a network. To do this using TensorFlow we could make use of two parameters when training the network: `class_weight` and `sample_weight`.

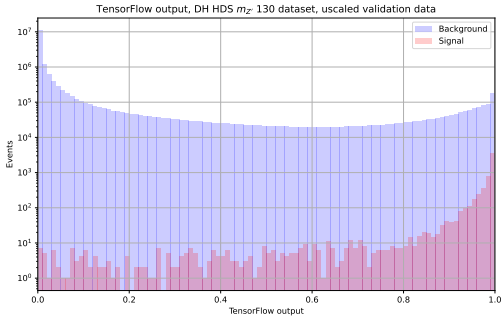
`class_weight` works as a dictionary that weights events that are keys on the dictionary. For our purposes we can make a dictionary where we weight signal and background events differently, this is the same type of scaling that was done in the previous section. `sample_weight` takes in individual weights for every single event that goes into the netwrok, meaning that it is crucial that we know that the desired weight matches the desired event. Ideally we would use both weighting methods, `class_weight` to balance the signal to background ratio and `sample_weight` with the MC weights. However there is

a bug in TensorFlow (up to version `GPU 2.7.1`) that makes it so the program doesn't run when using both parameters. This is not a big problem though, as when looking at the source code one can see that what TensorFlow does with both weights is multiply them togheter. Thus we could try to make use of this "balaned weighting" method to see if the network learns the different models better.

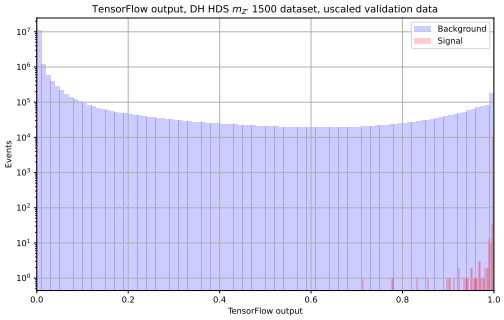
We can see in Table 7.1 how uneven the data (FULL Z') is. To balance the data we have four options. We can either *weigh up the signal* events by the ratio of background over signal. Or *weigh down the bacgkround* with the ratio of signal over background. However this ratio might differ a lot when using the MC raw event ratio or the SOW events ratio. I have tested all four possibilities and these can be see below.

Table 7.1: Table showcasing the data imbalance between background and signal in both raw MC events and actual events. By actual events it is meant weighted events. This is for the training dataset on the FULL Z' dataset.

	Actual events	MC events
Background	2,715,280.4	69,664,290
Signal	388.4	2,991,598

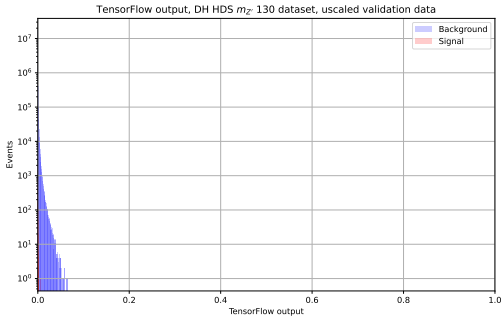


(a) NN prediction of DH HDS $m_{Z'} = 130$ GeV

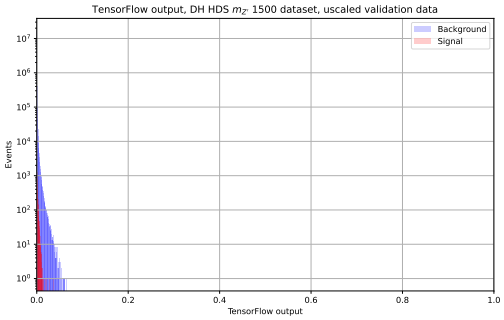


(b) NN prediction of DH HDS $m_{Z'} = 1500$ GeV

Figure 7.9: NN prediction when weighting the signal with the SOW ratio.

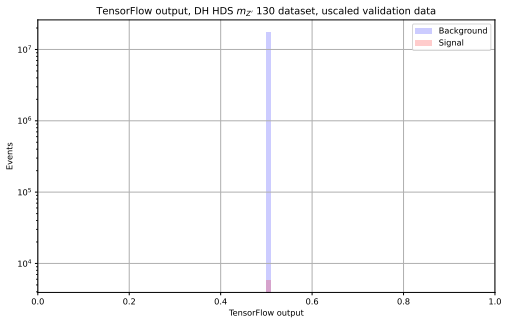


(a) NN prediction of DH HDS $m_{Z'} = 130$ GeV

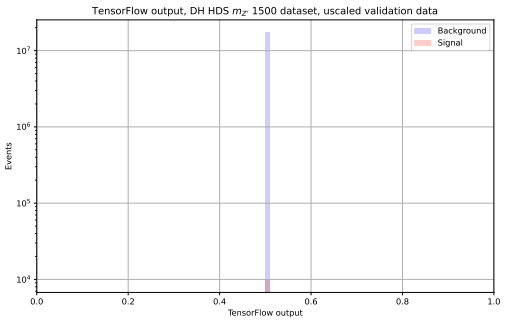


(b) NN prediction of DH HDS $m_{Z'} = 1500$ GeV

Figure 7.10: NN prediction when weighting the signal with the raw event ratio.

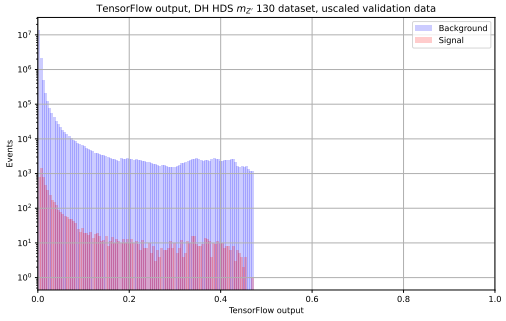


(a) NN prediction of DH HDS $m_{Z'} = 130$ GeV

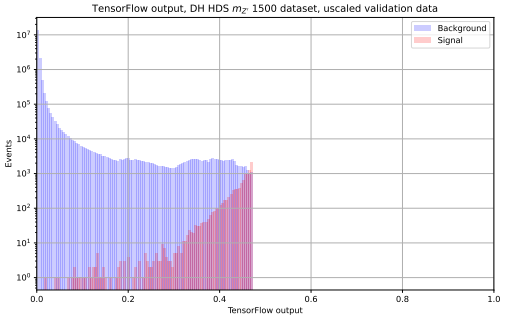


(b) NN prediction of DH HDS $m_{Z'} = 1500$ GeV

Figure 7.11: NN prediction when weighting the background with the SOW ratio.



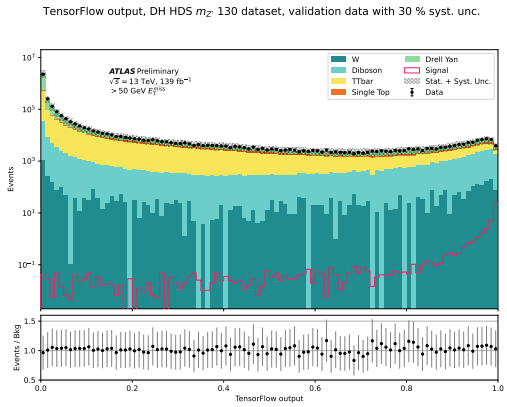
(a) NN prediction of DH HDS $m_{Z'} = 130$ GeV



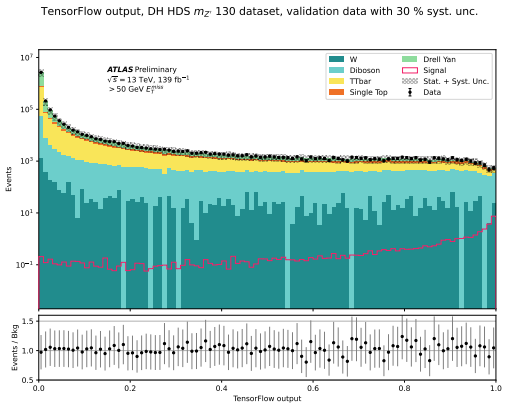
(b) NN prediction of DH HDS $m_{Z'} = 1500$ GeV

Figure 7.12: NN prediction when weighting the bacgkround with the raw event ratio.

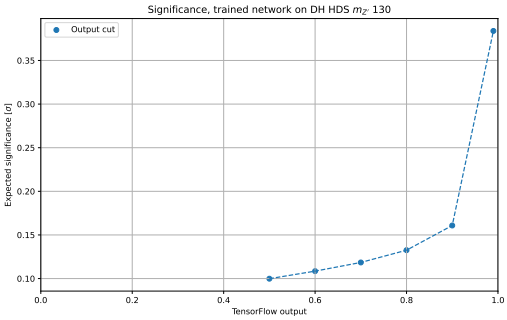
As we can see almost every way of doing this balanced weighting gives poor results. The only one that seems to work is when we we scale up the MC weights with the ratio between the SOW of the bacgkround over signal, however it does not seem to learn the model with lower mass better compared to how it learned the model with higher mass. It seems to have generally learned both models better, which might be a hint that it learns other models worse, i.e. EFT models with much lower cross section. If we now compare the correctly scaled validation plots of DH HDS $m_{Z'} = 130$ GeV.



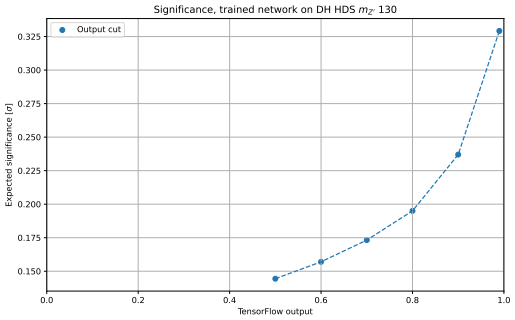
(a) MC weights + scaling signal up with SOW ratio



(b) Only using raw MC events weigh down bacgkround



(c) The expected significance of a)



(d) The expected significance of b)

Figure 7.13: Comparison of the best balanced weighting method to the weighting method of the previous section. Figure a) and b) show the validation data of both cases, c) and d) show the expected significance of the validation plots when making a cut on the output.

From this we can see that the the network that trained when balancing the data is better at classifying the DH HDS $m_{Z'} = 130$ GeV model correctly as signal than the other network. We also observe that it wrongly classifies more background as signal than the other network, but even if this is the case the expected significance is greater in the new network. The reason that the new network wrongly classifies more background as signal might be becuse I utilized the same hyperparameters when training both newtworks, when it might be better to use different hyperparameters on each. Another thing that might be of significance is that I used the ratio of the training set, it might make a difference (better or worse) to use the ratio of the full dataset when training as this is more general. Since the ratio most likely differs for the training and testing set.

Should I add here that this will not be pursued further? Or rather include it on the "results" part of the thesis

7.1.5 Grid Search

The full results of the gridsearch when setting `n_layers` $= 2$ (one hidden layer) and $\eta \in [0.001, 0.01, 0.1, 1]$, $\lambda \in [10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}]$ and `n_neuron`$\in [1, 10, 50, 100]$ can be found in my GitHub under `Plots/NeuralNetwork/FULL/GRID_lamda_eta_neurons`, but for the sake of this thesis not being too long I will only show the significance plot as well as the AUC for the testing and training set when setting $\lambda = 10^{-5}$. The significance is seen in Figure 7.14, while the AUC is in Figure 7.15.

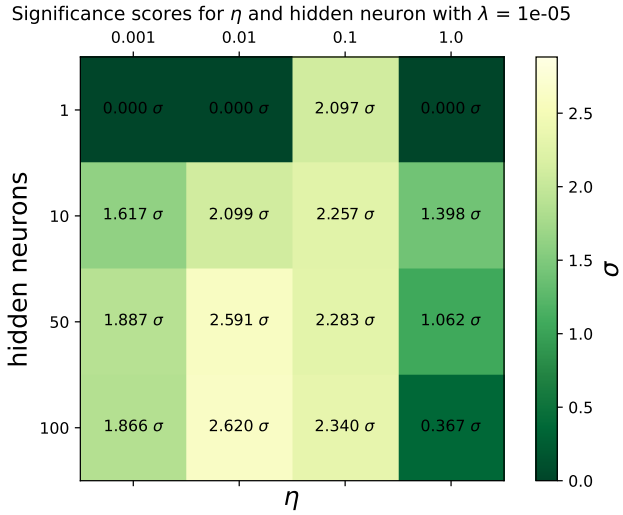
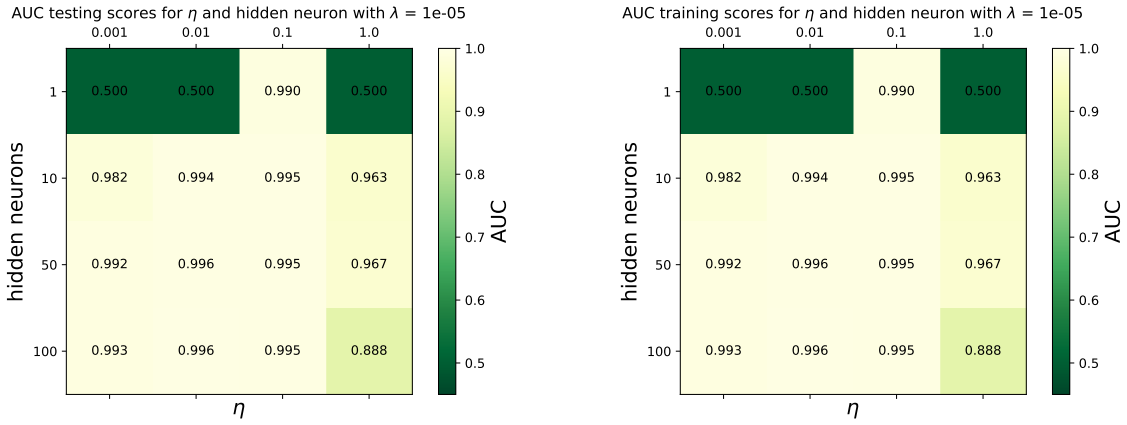


Figure 7.14: Grid search significance with $\lambda = 10^{-5}$ and n_layers $= 2$



(a) Testing AUC

(b) Training AUC

Figure 7.15: Grid search AUC with $\lambda = 10^{-5}$ and n_layers $= 2$

Doing the same but with more hidden layers and setting $\lambda = 10^{-5}$ we get the results shown in GitHub under `Plots/NeuralNetwork/FULL/GRID_layers_eta_neurons`, for the sake of this thesis not being too long I will again only show the significance plot as well as the AUC for the testing and training set but this time when setting $\eta = 0.01$. The significance is seen in Figure 7.16, while the AUC is in Figure 7.17.

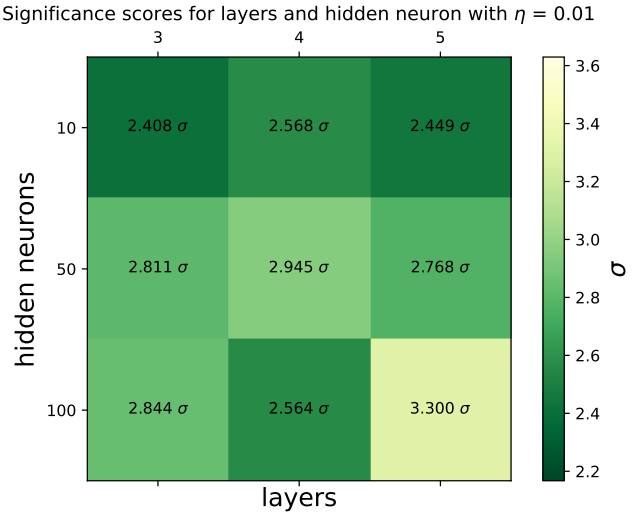
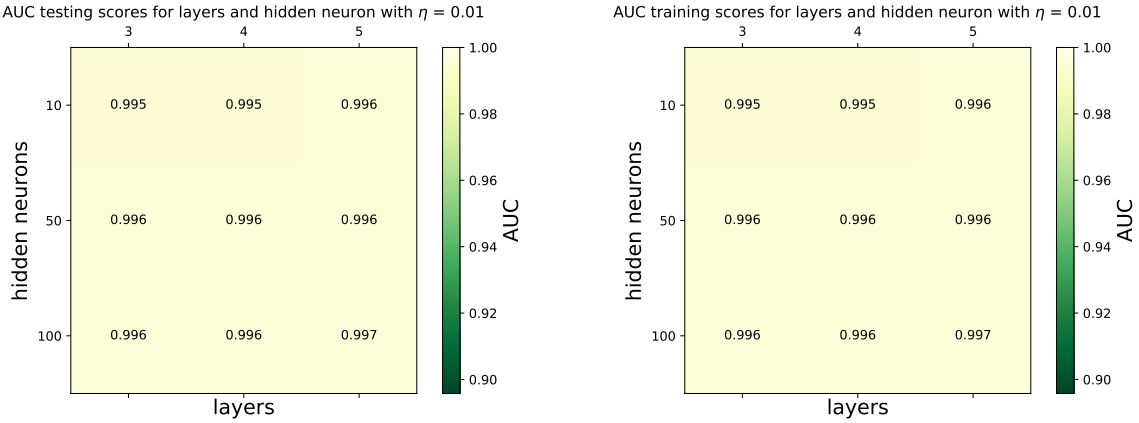


Figure 7.16: Grid search significance with $\lambda = 10^{-5}$ and and $\eta = 0.01$

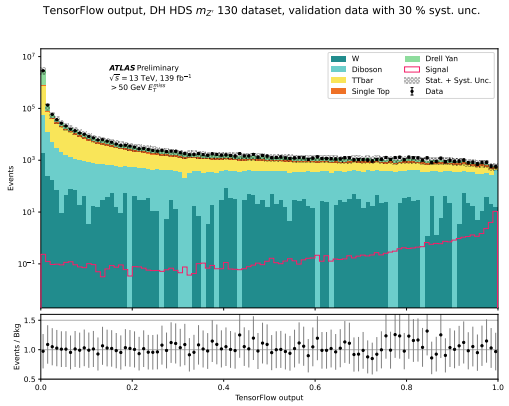


(a) Testing AUC

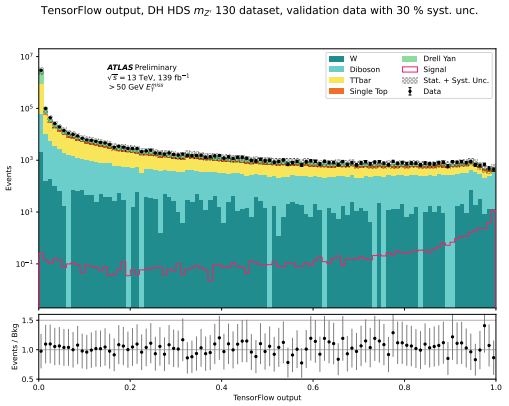
(b) Training AUC

Figure 7.17: Grid search significance with $\lambda = 10^{-5}$ and $\eta = 0.01$

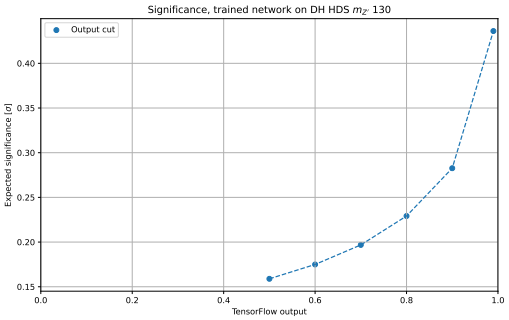
I also made a test network with the same hyperparameters as the best one in Figure 7.16, the only difference being that it has 10 hidden layers. I plotted the validation data to see how different the networks were at predicting the DH HDS $m_{Z'} = 130$ GeV model. These were trained and tested when using the Z-score normalization, Eq. (6.3), and the weighting method explained in section **??**. The results are shown in the figure below.



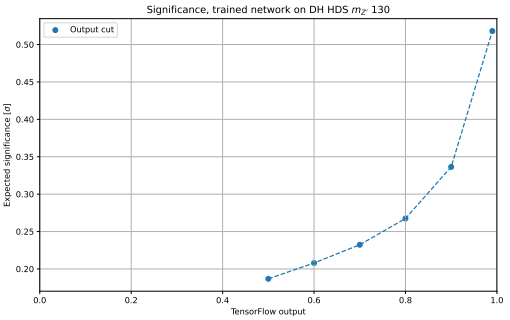
(a) Four hidden layers



(b) Ten hidden layers



(c) The expected significance of a)



(d) The expected significance of b)

Figure 7.18: Comparison of the network performance when having four and ten hidden layers. Figure a) and b) show the validation data of both cases, c) and d) show the expected significance of the validation plots when making a cut on the output.

This hints that we could be able to make a DNN with more hidden layers and get better results. However there are a few things that need to be noted when doing this, aside from the padding which is an even greater problem. The first and smallest one is that we could have used batch normalization instead of Z-score, but this is again something to be further discussed.

The second being that I have not used the "balanced weighting" method when training the network, which might be for better or worse if the network really does ignore all EFT models...

The third one which is more technical is that since more complex networks require more computational power, then this leads to us decreasing the batch size. Which lowers the statistics of signal, and might even lead to the network training on batches without any signal sample at all. So the trade off is also something to be discussed.

The last thing to be noted is that having a DNN completely removes the possibility of combining the results of multiple networks trained on a single model, as the imbalance becomes too much for the network to see anything. A solution to this however, is that instead of combining the results of multiple networks trained on a singular model, one could try the Parametrized NN approach used by Baldi et. al. [19], which could potentially avoid the imbalance problem, but this is proposed as a plausible new research project due to time constrain on this thesis.

7.2 Boosted Desicion Tree Training

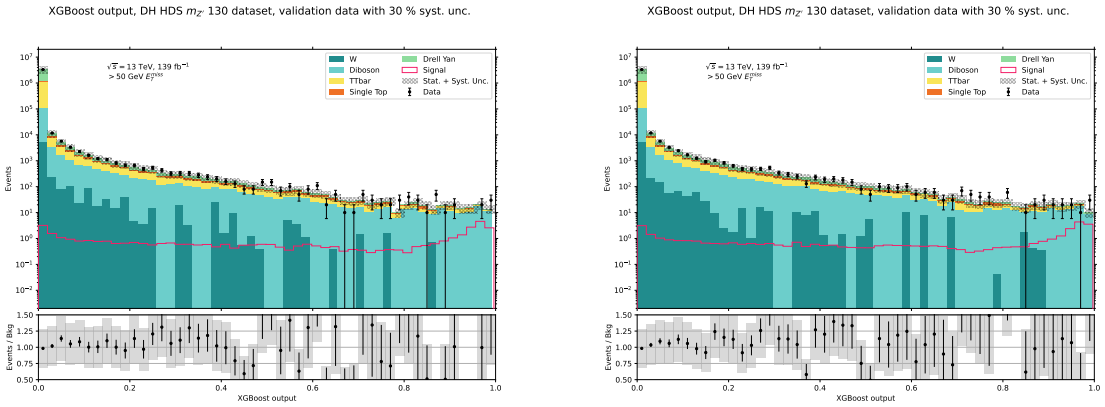
7.2.1 Weights

I have tried all of these options and the results can be seen in Figure 7.19.

As we can see it makes a significant difference whether we use the weights to re-weight MC events to expected events. But there is no mathematical reason as to why we should include this re-weighting weight as sample weights, as the reason to use sample weights is to *only* balance signal and bacgkround. In theory it makes no sense whatsoevert to include these weights either as the network doesn't really care for cross sections or luminosity, and what we are doing in principle is making it harder for the network to learn anything. But as seen on the results, the BDT learns the background extremely well when using the weights, while it also does a poorer job in learning the signal we are testing.

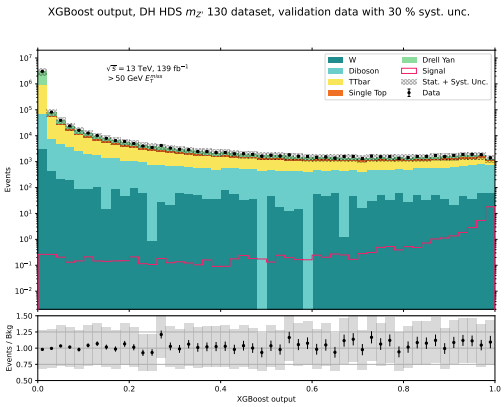
In a sense this is not a negative thing for our purposes, but stricly speaking we are invoking a semi-unsupervised learning method by punishing the network if it learns the signal too quickly. To get to the point as why this is good for our purposes, we are indirectly making our network more model indepentent! Because of this, the method that will be pursued further in this thesis will be to take the positive weights and balance the data.

Add that ATLAS used it, so because of that so can I?



(a) Using the scaled absolute value of the weights

(b) Using only positive weights



(c) Using no weights

Figure 7.19: Difference when using different weighting methods. All networks were trained using the balancing method explained in Section **??**

7.2.2 Grid Search

The results can be shown in Figure 7.20. This is however highly radical as the convention is to normally not have a depth greater than 7, the reason being that the network is highly likely to overtrain and give wrong predictions. However this was not the case for me as seen for example in Figure 7.21.

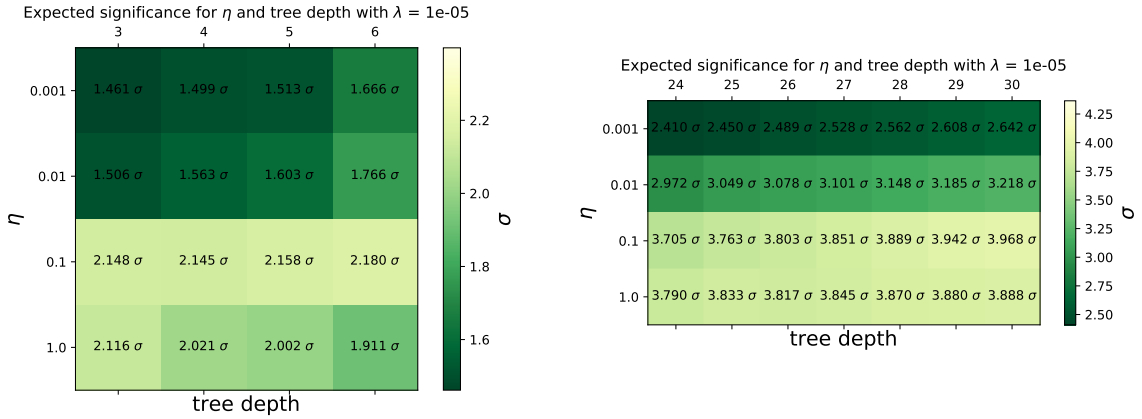
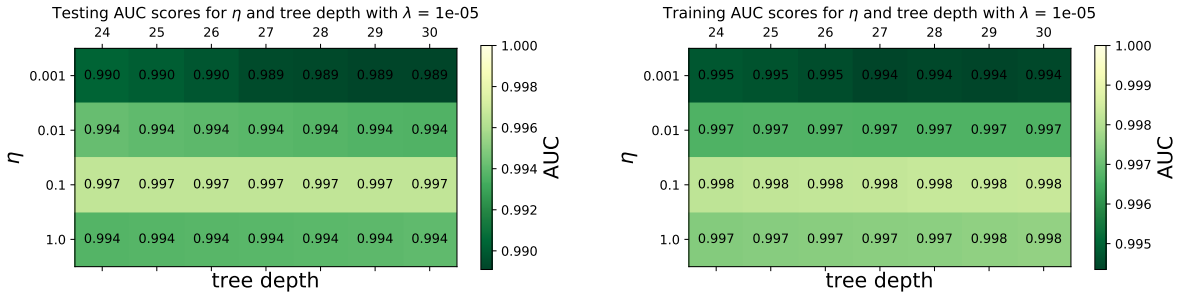


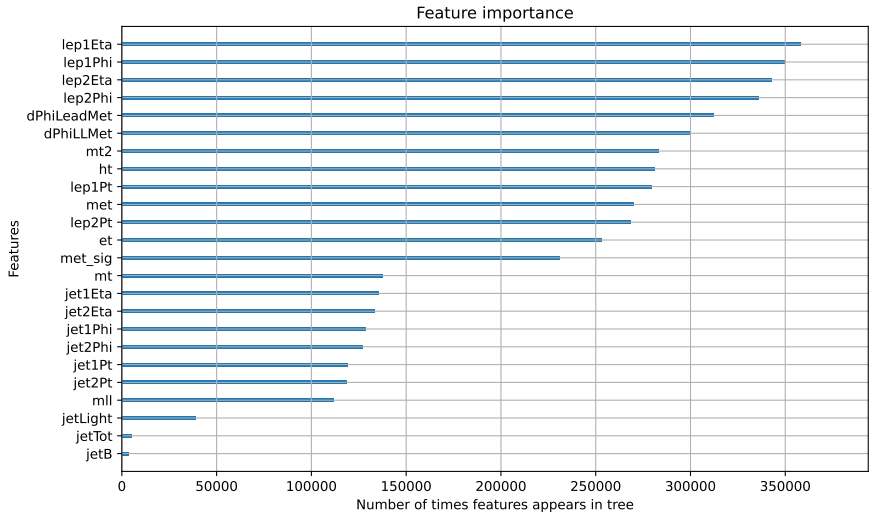
Figure 7.20: Grid search expected significance going to a depth of up to 30



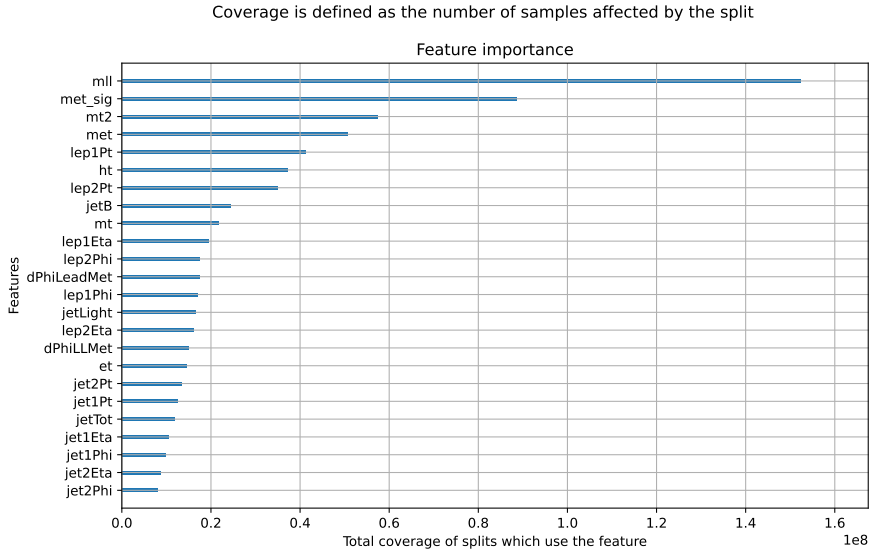
(a) Testing AUC (b) Training AUC

Figure 7.21: Grid search AUC going to a depth of up to 30

I understand however that this is controversial since we are splitting a data set, that is at best of size 2^{27}, 30 times. That means that after a depth of 27 there is exaclty one event pr branch. So how does a depth of 30 make sense? To help with this we could use a feature in XGBoost to see which features are most important when evaluating a signal. When testing the network trained on the FULL Z' DM data set on a DH HDS $m_{Z'} = 130$ GeV model we get the features shown in Figure 7.22 as most important.



(a) Using "weight" metric



(b) Using "coverage" metric

Figure 7.22: Feature importance of depth 30 network trained on FULL Z' DM data set when testing it on DH HDS $m_{Z'} = 130$ GeV model.

As we can see these features vary a lot depending on which metric we use to evaluate the importance. When using the "coverage" metric, which as stated is defined as the number of samples affected by the split, we get the features we physically expect to be important when trying to single out a DM model. And this metric is arguably the one we need to use to define what features are important. Since the more samples a feature split, the more powerful it is to separate signal from background.

We can see however that when we use "weight" as a metric, which is the XGBoost standard metric, we get completely unexpected features that we physically don't expect to be important when trying to single out a DM model. But as described by the metric, the "weight" is the number of times a feature appears in a tree. Which might explain that the reason the pseudorapidity and ϕ range so high on this list, is simply because the tree is struggling to find a pattern here and is trying extra hard to single out DM from SM.

As the previous results are to be taken with a heavy grain of salt, I conducted another grid search. On the second grid search I set the values of $\eta = 0.1$ as the trend showed this giving the best results with less overtraining, and $\lambda = 10^{-5}$. This grid search had `n_estimators` $\in [10, 100, 500, 1000]$ and depth $\in [3, 4, 5, 6]$. The expected significance is shown in Figure 7.23. The testing and training AUC can be seen in Figure 7.24.

should I conduct a new grid search with different λ and loss functions?

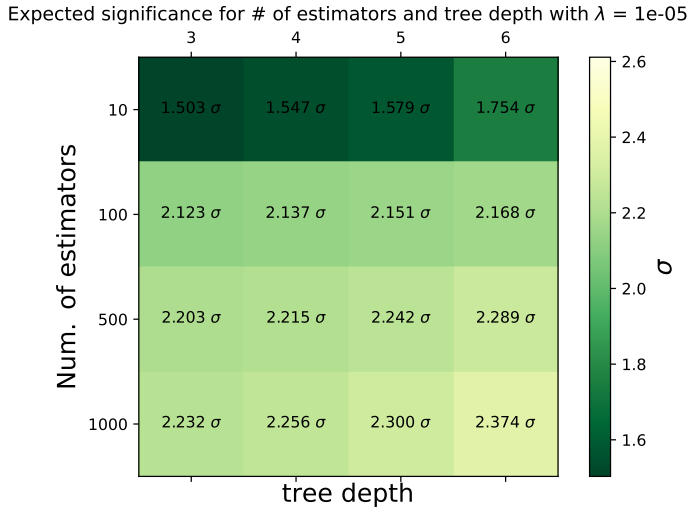
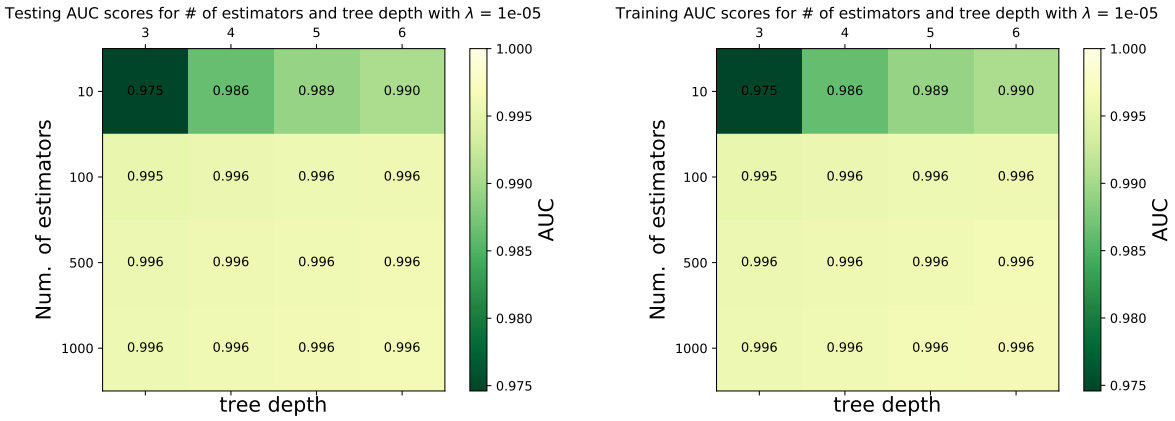


Figure 7.23: Grid search expected significance when setting $\lambda = 10^{-5}$ and $\eta = 0.1$

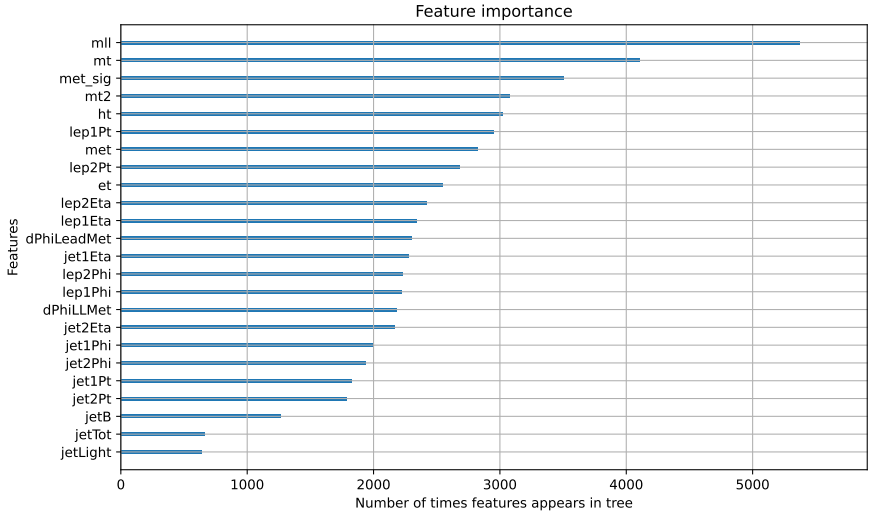


(a) Testing AUC

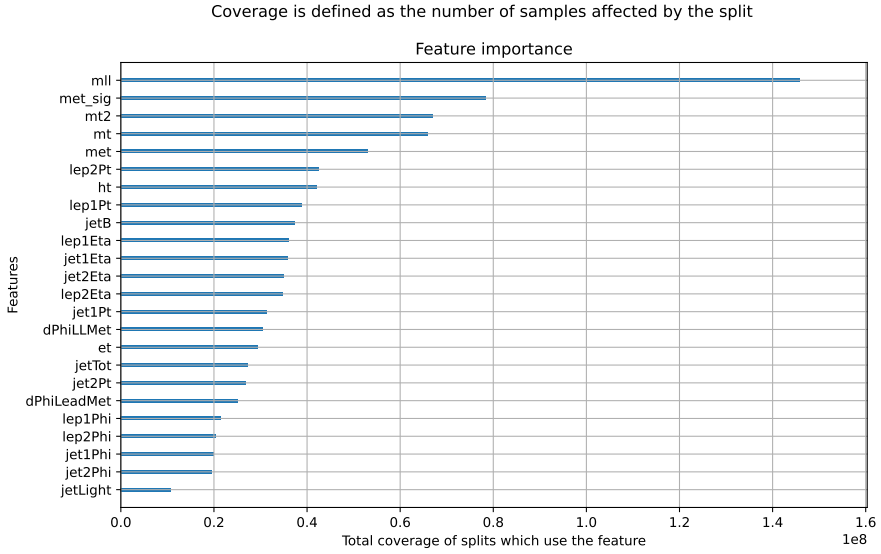
(b) Training AUC

Figure 7.24: Grid search AUC when setting $\lambda = 10^{-5}$ and $\eta = 0.1$

When testing the best network with a depth of 6 and 1000 estimators on the same DH HDS $m_{Z'} = 130$ GeV model we get the feature importance plots shown in Figure 7.25. Here we see that the "weight" metric gives us the expected features as most important. But the "cover" metric seems to be less of what we expect since the jet kinematic variables score higher, this might just be a curiosity rather than something to be suspect of.



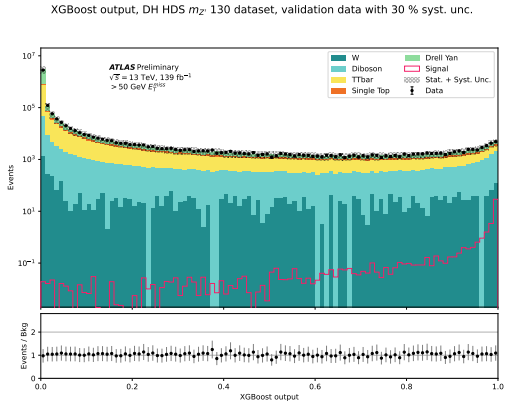
(a) Using "weight" metric



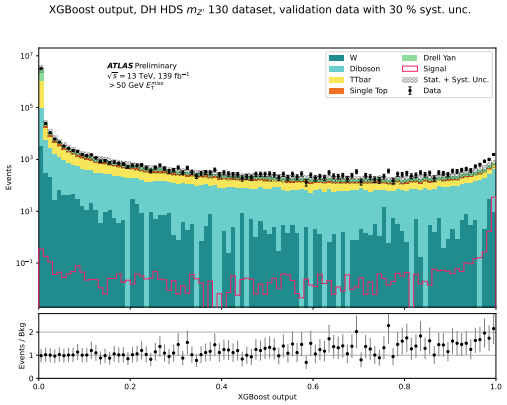
(b) Using "coverage" metric

Figure 7.25: Feature importance of depth 30 network trained on FULL Z' DM data set when testing it on DH HDS $m_{Z'} = 130$ GeV model.

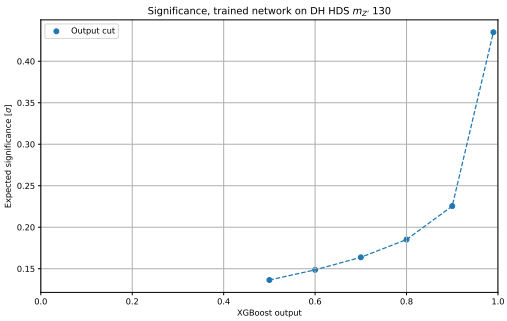
To showcase the difference in signal recognition between the monstrous 30 depth BDT to the more sensible 6 depth BDT, I again tested the networks on the good old DH HDS $m_{Z'} = 130$ GeV model. The results as well as their expected significance can be seen below.



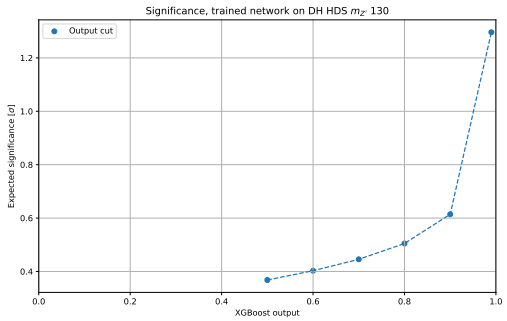
(a) Depth of 6



(b) Depth of 30



(c) The expected significance of a)



(d) The expected significance of b)

Figure 7.26: Comparison of the network performance when having a depth of 6 and 30. Figure a) and b) show the validation data of both cases, c) and d) show the expected significance of the validation plots when making a cut on the output.

The difference is extreme, when looking at the monster of depth 30 we can get an expected significance of 1.2 σ (without uncertainties) on our model of max 15 events, only having made a cut of 50 GeV on the missing transverse energy. We can however see that the data and background do not agree to the same degree of the network with depth 6. Using purely statistical uncertainty and assuming a systematic uncertainty of 30%, we see that a few data ponts do not agree with the MC background. These data points are points the network classified as signal, so if we completely trusted the network this would be a hint of new physics! However this is the last thing we should assume, and rather take this as a hint that the network is doing something fishy.

when should I use them?

If I had access to XGBoost with built GPU support, I would increase the number of estimators even more to check if this increases the significance while still having a depth of maximum 6. However as of now this is not possible. As the weighting method explained in the previous section was not included here, we will drop going to a tree depth of 30, and have a maximum of 6.

Chapter 8

Comparison to cut and count

Testing three models using the classical data analysis way we apply cuts to kinematic variables and try to isolate the signal from the background to then calculated the expected significance. The three models I chose to test are all High Dark Sector models with $m_{Z'} = 130$GeV. They are a Light Vector (LV), Dark Higgs (DH) and Effective Field Theory (EFT) models. The cuts I made on these are shown in Table 8.1.

Table 8.1: Table showcasing the cuts used when doing the cut and count method.

	Cut
E_T^{miss}/σ	> 10
m_T	> 160 GeV
m_{ll}	> 120 GeV
Number of B-jets	0
m_{T2}	> 110 GeV

Since the cross section to find Dark Matter is really small we have to use the low-statistics expected significance formula to find the closest to correct significance. The formula is

$$Z = \sqrt{2\left[(s+b)\ln\left(1+\frac{s}{b}\right) - s\right]} \tag{8.1}$$

Where s is the number of signal events and b is the number of background events. Using this we get the results shown in Table 8.2 for the electron channel and Table 8.3 for the muon channel. Also included on the tables are the number of events. One thing worth mentioning is that when adding another cut on the maximum invariant mass increases the significance. The significance for LV on the electron channel was at 1.2σ when adding a cut stating that $m_{ll} < 150$ GeV. This makes sense since the models in question all have a $m_{Z'} = 130$GeV. This cut was not added since we do not want to put a cap on the mass of the propagator, as we don't know what the real mass is.

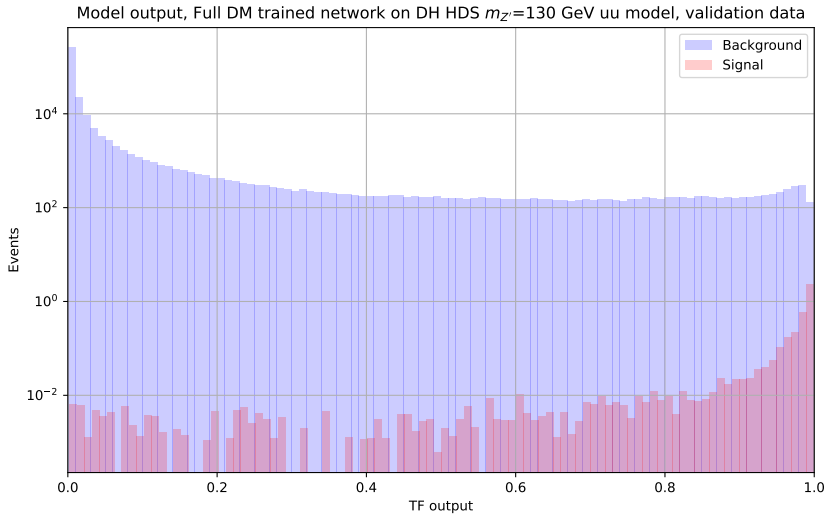
Table 8.2: Table showcasing the result of the cut and count method for the electron channel.

	LV	DH	EFT	Background
Events before cuts	15	20	0	1,256,624
Events after cuts	4	6	0	117
Expected significance [σ]	0.4	0.6	0	

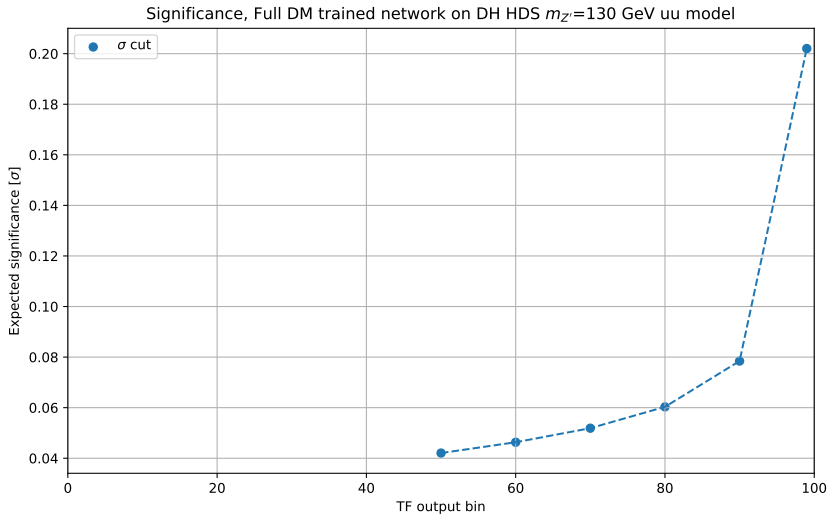
If we were to compare these results with what our NN and BDT that trained on the full dataset we see that we can calculate the expected significance in different locations for the validation plots. Testing on the networks that trained using the data scientist method on the full DM dataset we get the results shown in Figure 8.1 for XGBoost and Figure 8.2 for the Neural Network.

Table 8.3: Table showcasing the result of the cut and count method for the muon channel.

	LV	DH	EFT	Background
Events before cuts	14	19	0	1,626,098
Events after cuts	3	5	0	108
Expected significance [σ]	0.36	0.51	0	

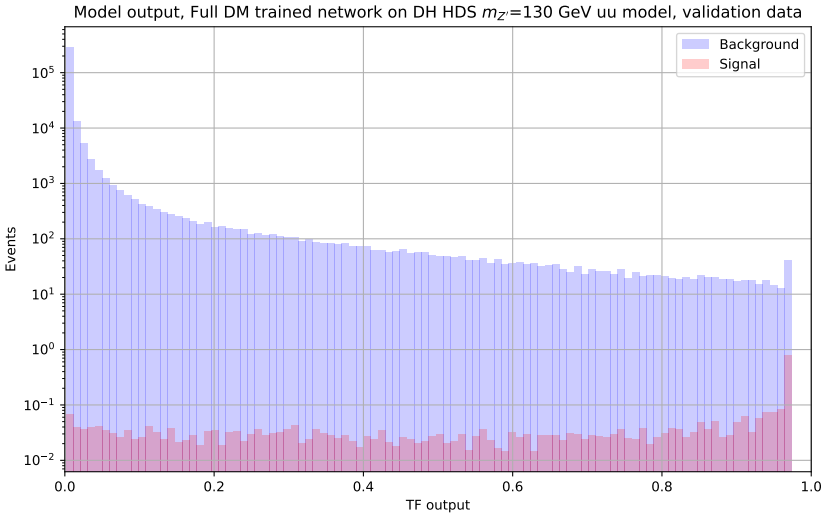


(a) Validation plot.

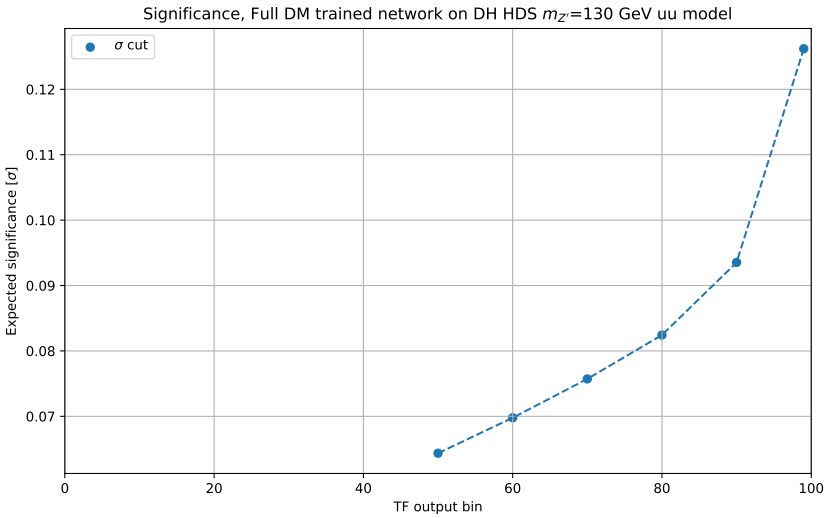


(b) Expected significance when looking at bins and forth.

Figure 8.1: Expected significance of XGBoost when trained on the Full DM dataset for the DH HDS $m_{Z'} = 130$ GeV muon model.



(a) Validation plot.



(b) Expected significance when looking at bins and forth.

Figure 8.2: Expected significance of the Neural Network when trained on the Full DM dataset for the DH HDS $m_{Z'} = 130$ GeV muon model.

As we can see the expected significance is lower using ML than a rough cut and count. My theory for why this is the case is because we are testing just ***one*** sample out of 154 different ones that are included for the three different theories I have acquired so far. And the ML networks shown above have both trained on a dataset including all 154 DM samples. The models that I tested might also not have been one of the "important" models the network learned from. Thus if I were to train the network individually based on the theory it might give better results.

Table 8.4: Table showcasing the result of the cut and count method for the electron channel.

	Signal	Background
MC events	2,990,986	69,664,902
Sum of "Weight"	388.75	2,714,091.3
Sum of generator weights	236.3	55,446,228,776,354.8
Sum of (generator weights*lumi / SOW)	9,167.1	61.1
Sum of (generator weights/SOW)	199.21	1.52

Table 8.5: Table Showcasing how uneven the training dataset is between signal and background. This is on the dataset which incorporates all the different DM MC samples

	Number of events	Sum of weights	Events × SOW [10^{13}]
Signal	2,991,543	36,327,943.99	1.08
Background	69,664,345	36,327,944.03	25.3

in spacetime. To find efficiency of signal: you have cross section for each MC period... so to find signal events before "cuts" multiple cross section with luminosity period! mll regions n met?

Appendix A

Dataset IDs for MC samples

Table A.1: Drell Yan background MC samples

Dataset ID	Process
700320	Zee_maxHTpTV2_BFilter
700321	Zee_maxHTpTV2_CFilterBVeto
700322	Zee_maxHTpTV2_CVetoBVeto
700323	Zmumu_maxHTpTV2_BFilter
700324	Zmumu_maxHTpTV2_CFilterBVeto
700325	Zmumu_maxHTpTV2_CVetoBVeto
700326	Ztautau_LL_maxHTpTV2_BFilter
700327	Ztautau_LL_maxHTpTV2_CFilterBVeto
700328	Ztautau_LL_maxHTpTV2_CVetoBVeto
700452	Zee_mZ_120_ECMS_BFilter
700453	Zee_mZ_120_ECMS_CFilterBVeto
700454	Zee_mZ_120_ECMS_CVetoBVeto
700455	Zmumu_mZ_120_ECMS_BFilter
700456	Zmumu_mZ_120_ECMS_CFilterBVeto
700457	Zmumu_mZ_120_ECMS_CVetoBVeto
700458	Ztautau_mZ_120_ECMS_BFilter
700459	Ztautau_mZ_120_ECMS_CFilterBVeto
700460	Ztautau_mZ_120_ECMS_CVetoBVeto

Table A.2: Single top and TTbar background MC samples

Dataset ID	Process
410472	$t\bar{t}$ dilepton filtered
410644	Single top s-channel (top)
410645	Single top s-channel (anti-top)
410648	$W + t$ associated production dilepton filtered (top)
410649	$W + t$ associated production dilepton filtered (anti-top)
410658	Single top t-channel (top)
410659	Single top t-channel (anti-top)

Table A.3: Diboson background MC samples

Dataset ID	Process
363356	$Z \to qq + Z \to ll$
363358	$W \to qq + Z \to ll$
363359	$W^+ \to qq + W^- \to l\nu$
363360	$W^+ \to l\nu + W^- \to qq$
363489	$W \to l\nu + Z \to qq$
364250	$llll$
364253	$lll\nu$
364254	$ll\nu\nu$
364255	$l\nu\nu\nu$

Table A.4: W background MC samples

Dataset ID	Process
364156	$W \to \mu\nu$ maxHTpTV0_70_CVetoBVeto
364157	$W \to \mu\nu$ maxHTpTV0_70_CFilterBVeto
364158	$W \to \mu\nu$ maxHTpTV0_70_BFilter
364159	$W \to \mu\nu$ maxHTpTV70_140_CVetoBVeto
364160	$W \to \mu\nu$ maxHTpTV70_140_CFilterBVeto
364161	$W \to \mu\nu$ maxHTpTV70_140_BFilter
364162	$W \to \mu\nu$ maxHTpTV140_280_CVetoBVeto
364163	$W \to \mu\nu$ maxHTpTV140_280_CFilterBVeto
364164	$W \to \mu\nu$ maxHTpTV140_280_BFilter
364165	$W \to \mu\nu$ maxHTpTV280_500_CVetoBVeto
364166	$W \to \mu\nu$ maxHTpTV280_500_CFilterBVeto
364167	$W \to \mu\nu$ maxHTpTV280_500_BFilter
364168	$W \to \mu\nu$ maxHTpTV500_1000
364169	$W \to \mu\nu$ maxHTpTV1000_E_CMS
364170	$W \to e\nu$ maxHTpTV0_70_CVetoBVeto
364171	$W \to e\nu$ maxHTpTV0_70_CFilterBVeto
364172	$W \to e\nu$ maxHTpTV0_70_BFilter
364173	$W \to e\nu$ maxHTpTV70_140_CVetoBVeto
364174	$W \to e\nu$ maxHTpTV70_140_CFilterBVeto
364175	$W \to e\nu$ maxHTpTV70_140_BFilter
364176	$W \to e\nu$ maxHTpTV140_280_CVetoBVeto
364177	$W \to e\nu$ maxHTpTV140_280_CFilterBVeto
364178	$W \to e\nu$ maxHTpTV140_280_BFilter
364179	$W \to e\nu$ maxHTpTV280_500_CVetoBVeto
364180	$W \to e\nu$ maxHTpTV280_500_CFilterBVeto
364181	$W \to e\nu$ maxHTpTV280_500_BFilter
364182	$W \to e\nu$ maxHTpTV500_1000
364183	$W \to e\nu$ maxHTpTV1000_E_CMS
364184	$W \to \tau\nu$ maxHTpTV0_70_CVetoBVeto
364185	$W \to \tau\nu$ maxHTpTV0_70_CFilterBVeto
364186	$W \to \tau\nu$ maxHTpTV0_70_BFilter
364187	$W \to \tau\nu$ maxHTpTV70_140_CVetoBVeto
364188	$W \to \tau\nu$ maxHTpTV70_140_CFilterBVeto
364189	$W \to \tau\nu$ maxHTpTV70_140_BFilter
364190	$W \to \tau\nu$ maxHTpTV140_280_CVetoBVeto
364191	$W \to \tau\nu$ maxHTpTV140_280_CFilterBVeto
364192	$W \to \tau\nu$ maxHTpTV140_280_BFilter
364193	$W \to \tau\nu$ maxHTpTV280_500_CVetoBVeto
364194	$W \to \tau\nu$ maxHTpTV280_500_CFilterBVeto
364195	$W \to \tau\nu$ maxHTpTV280_500_BFilter
364196	$W \to \tau\nu$ maxHTpTV500_1000
364197	$W \to \tau\nu$ maxHTpTV1000_E_CMS

Appendix B

Kinematical variables distribution in control region

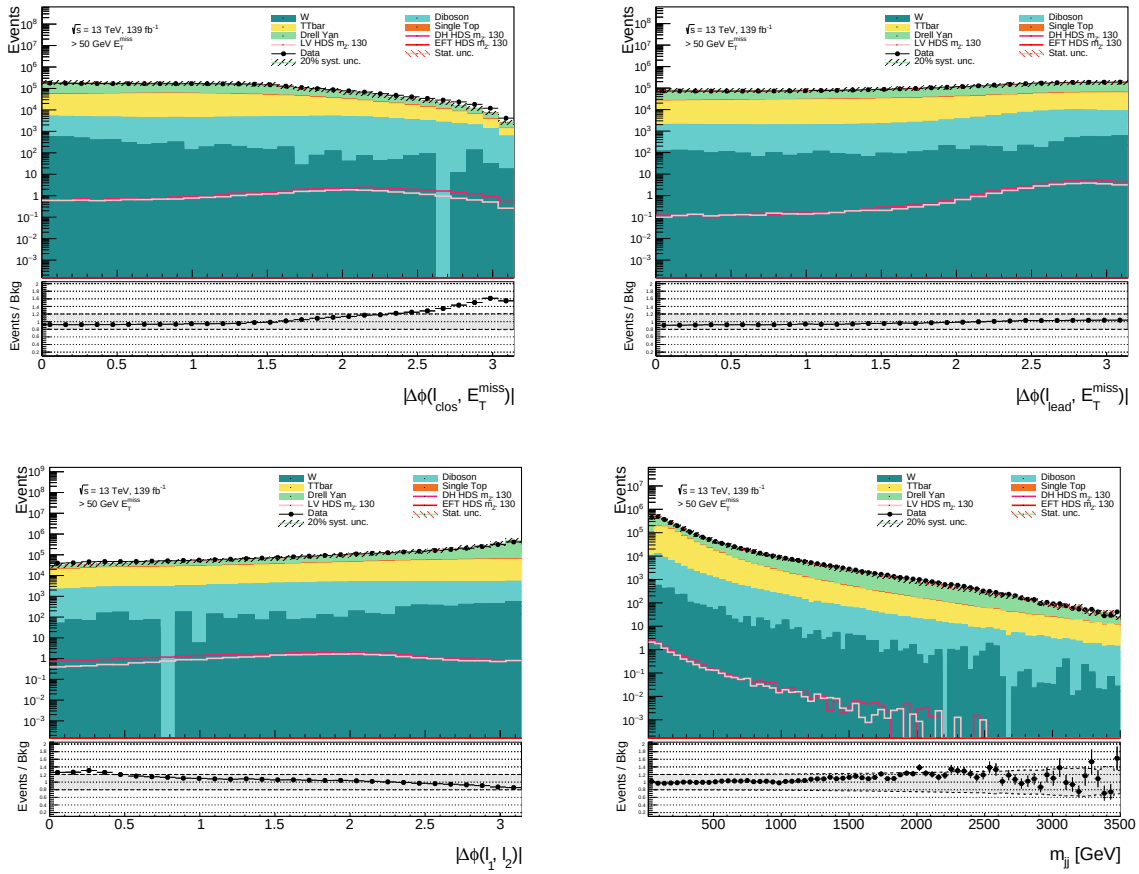


Figure B.1: $\Delta\phi(l_c, E_T^{miss})$, $\Delta\phi(l_l, E_T^{miss})$, $\Delta\phi(l_1, l_2)$ and m_{jj} distribution in the control region.

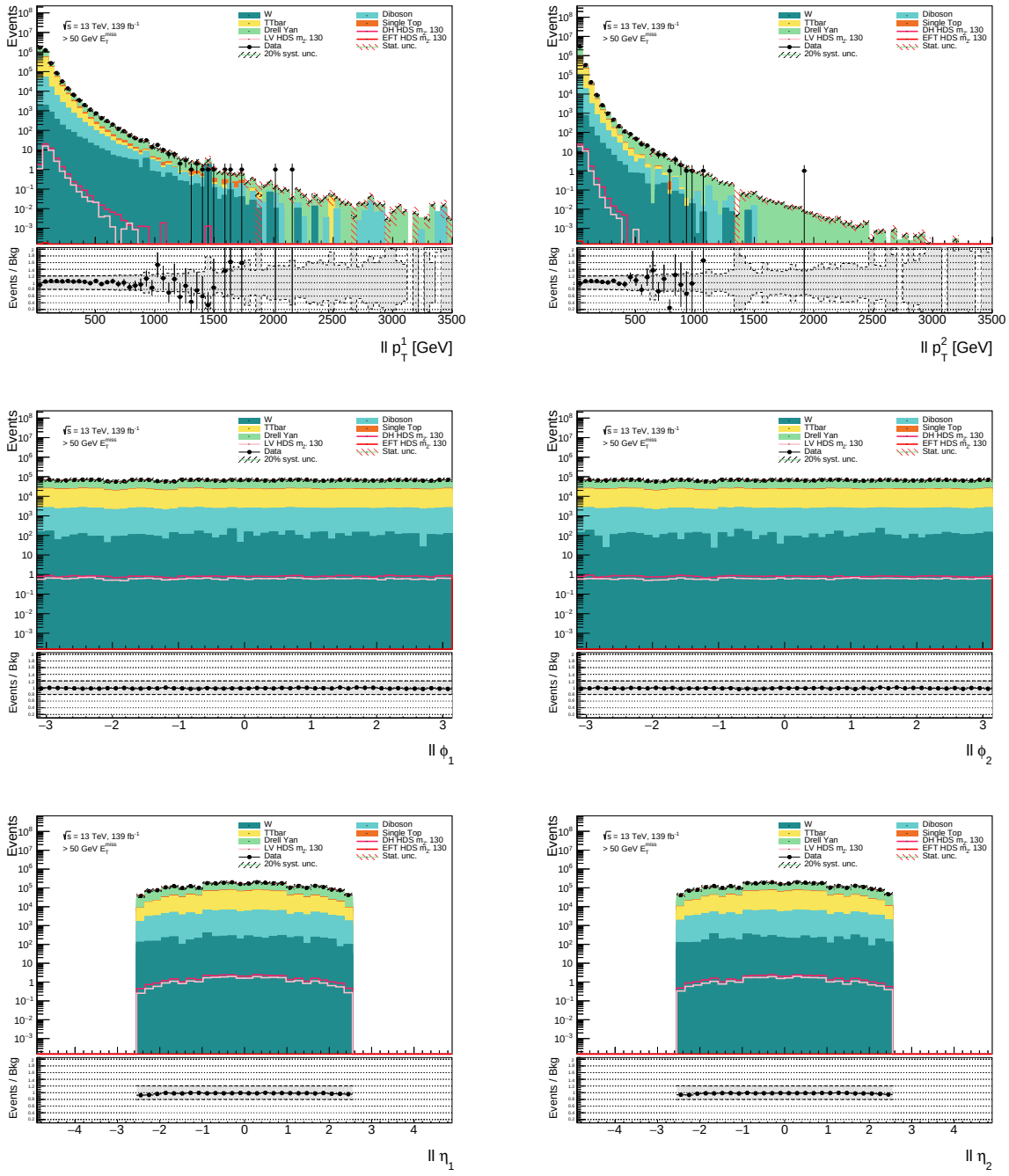


Figure B.2: Leptons p_T, ϕ and η distribution in the control region.

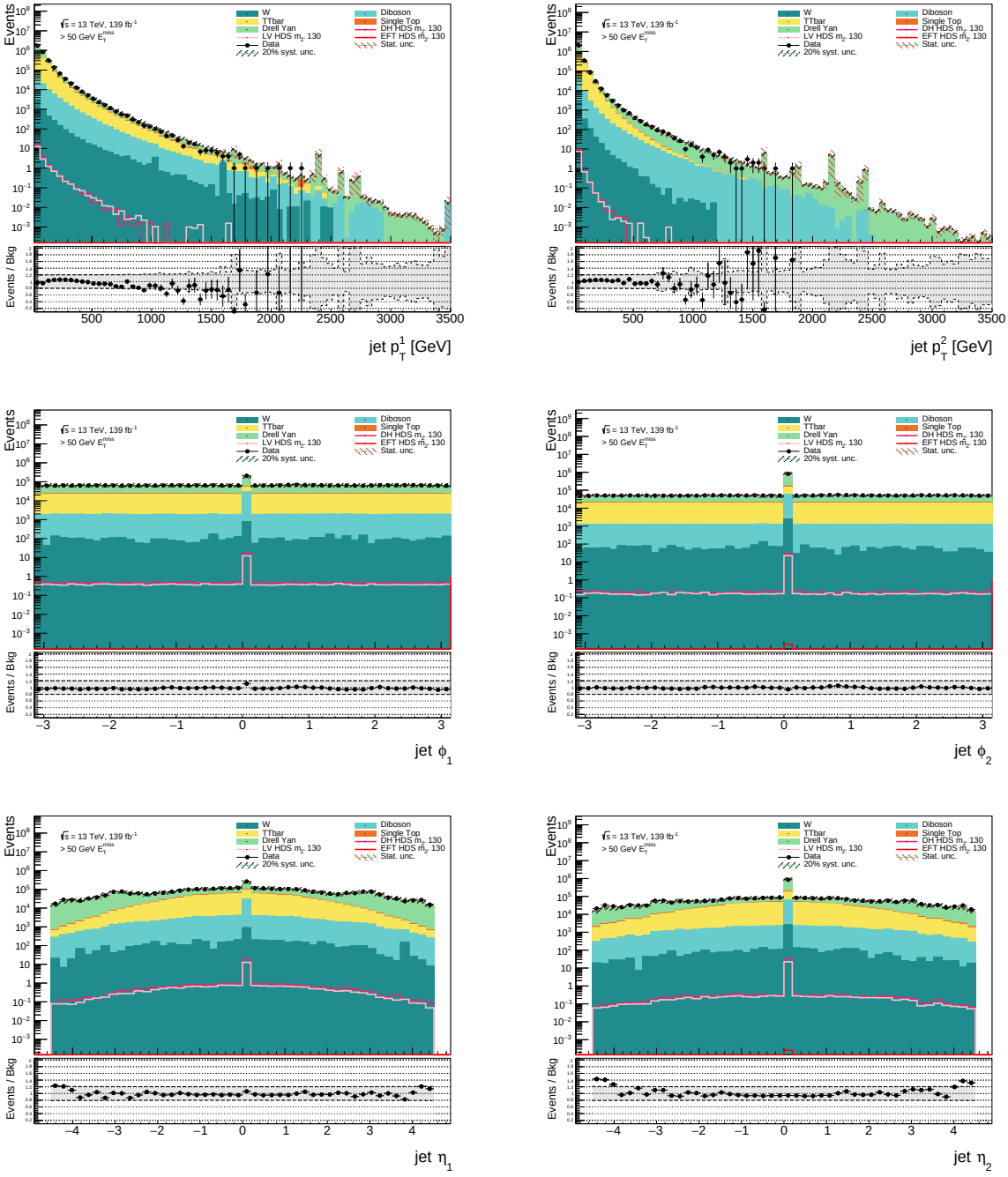


Figure B.3: Leading jets p_T, ϕ and η distribution in the control region.

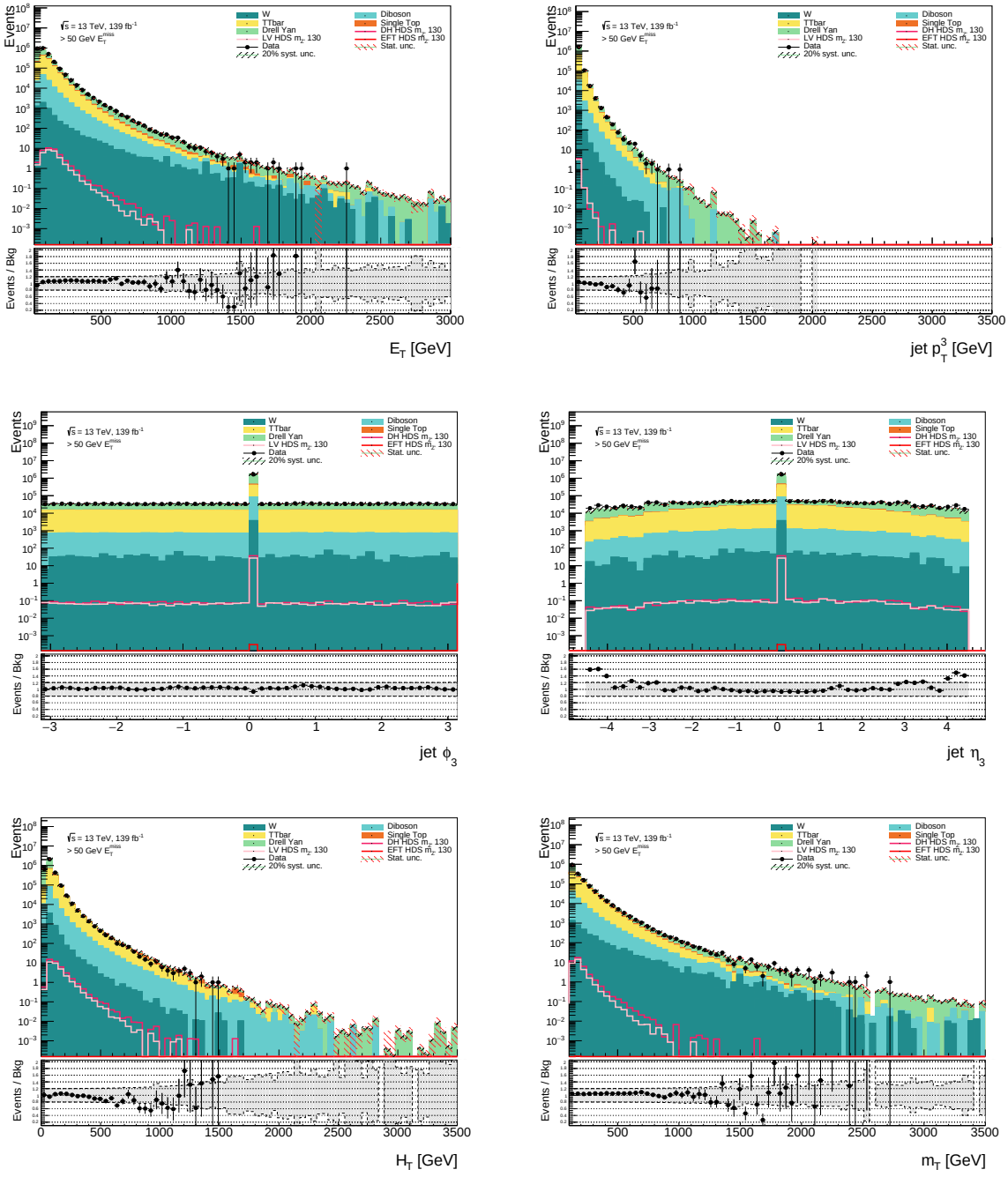


Figure B.4: p_T, ϕ and η of third leading jet and the dilepton pairs E_T, H_T and m_T distribution in the control region.

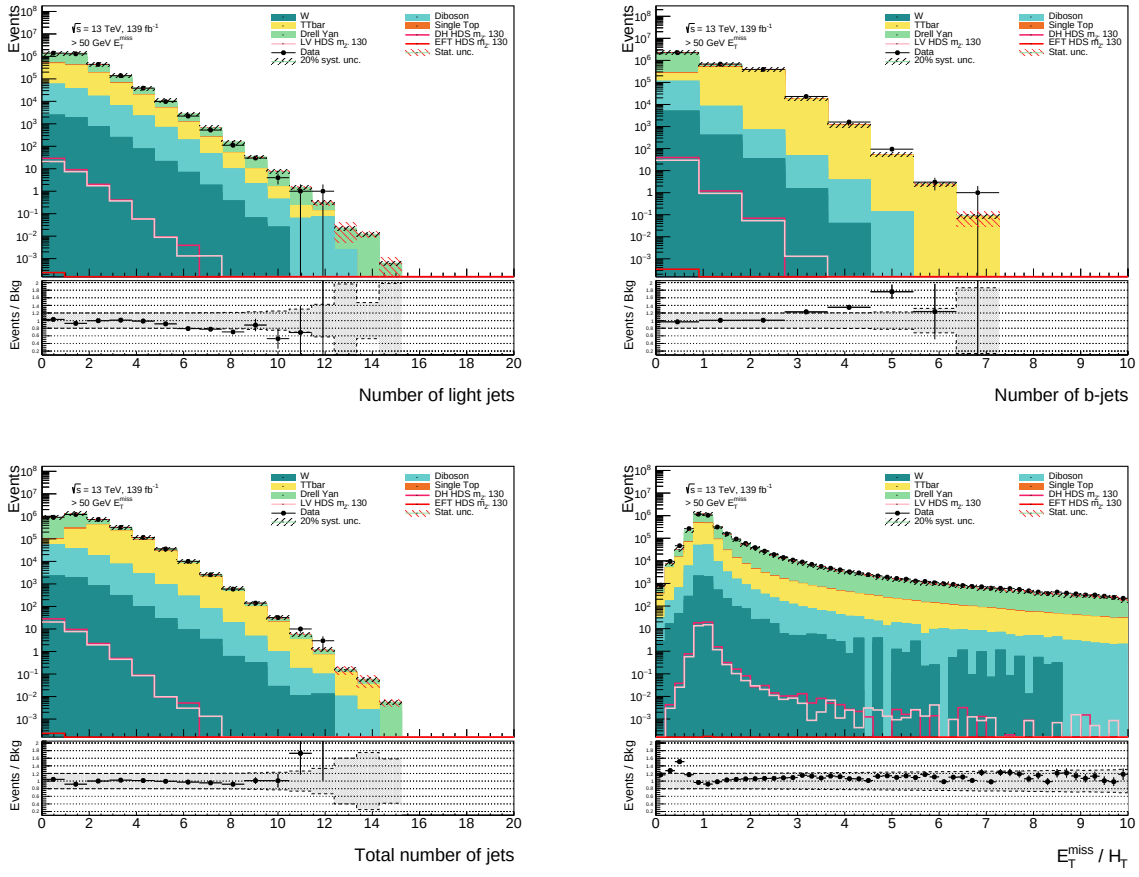


Figure B.5: Number of light, b- and total jets and E_T^{miss}/H_T distribution in the control region.

Appendix C

Data and MC agreement jets CR

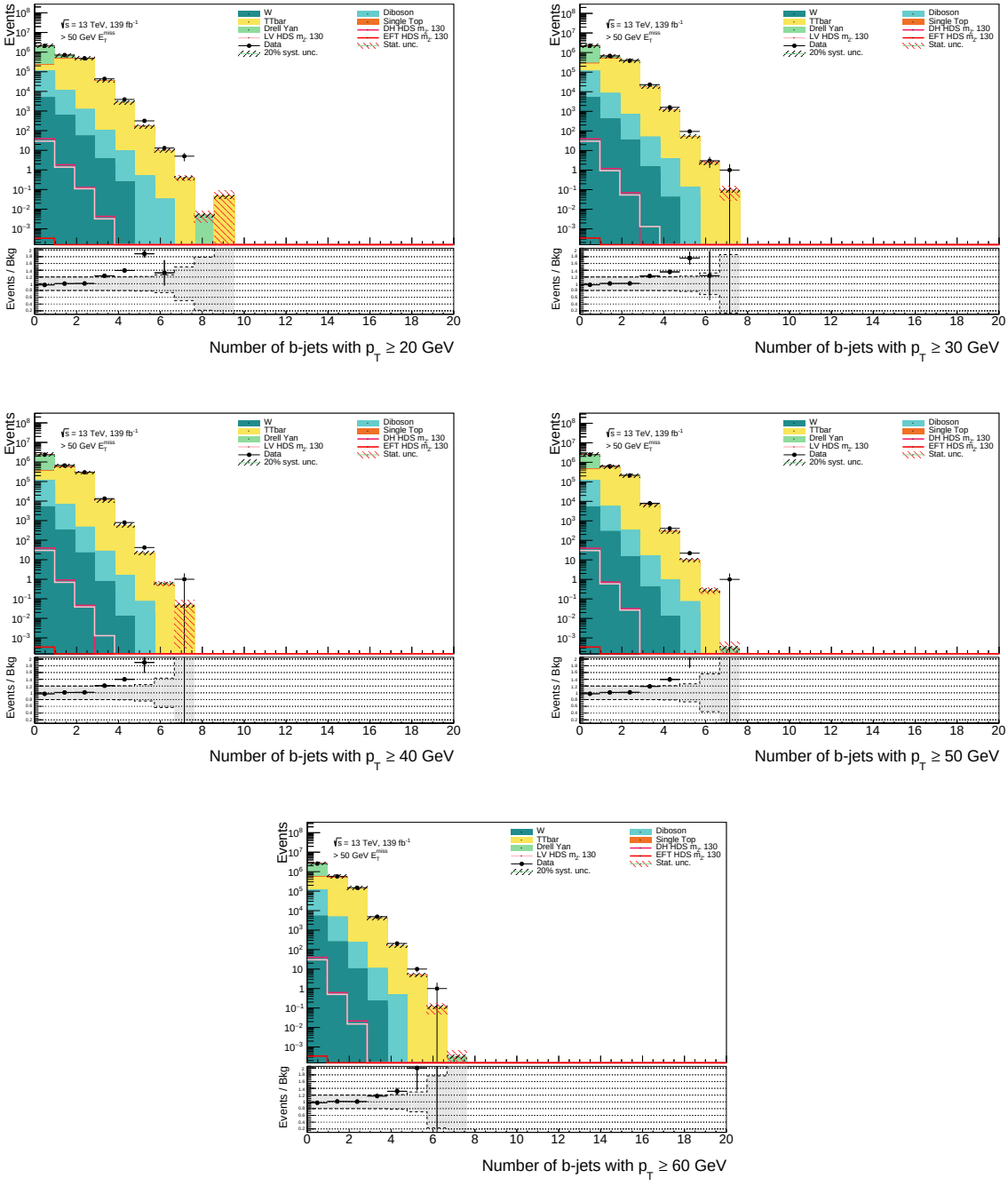


Figure C.1: Data and MC agreement on number of b- jets with different p_T cuts in CR.

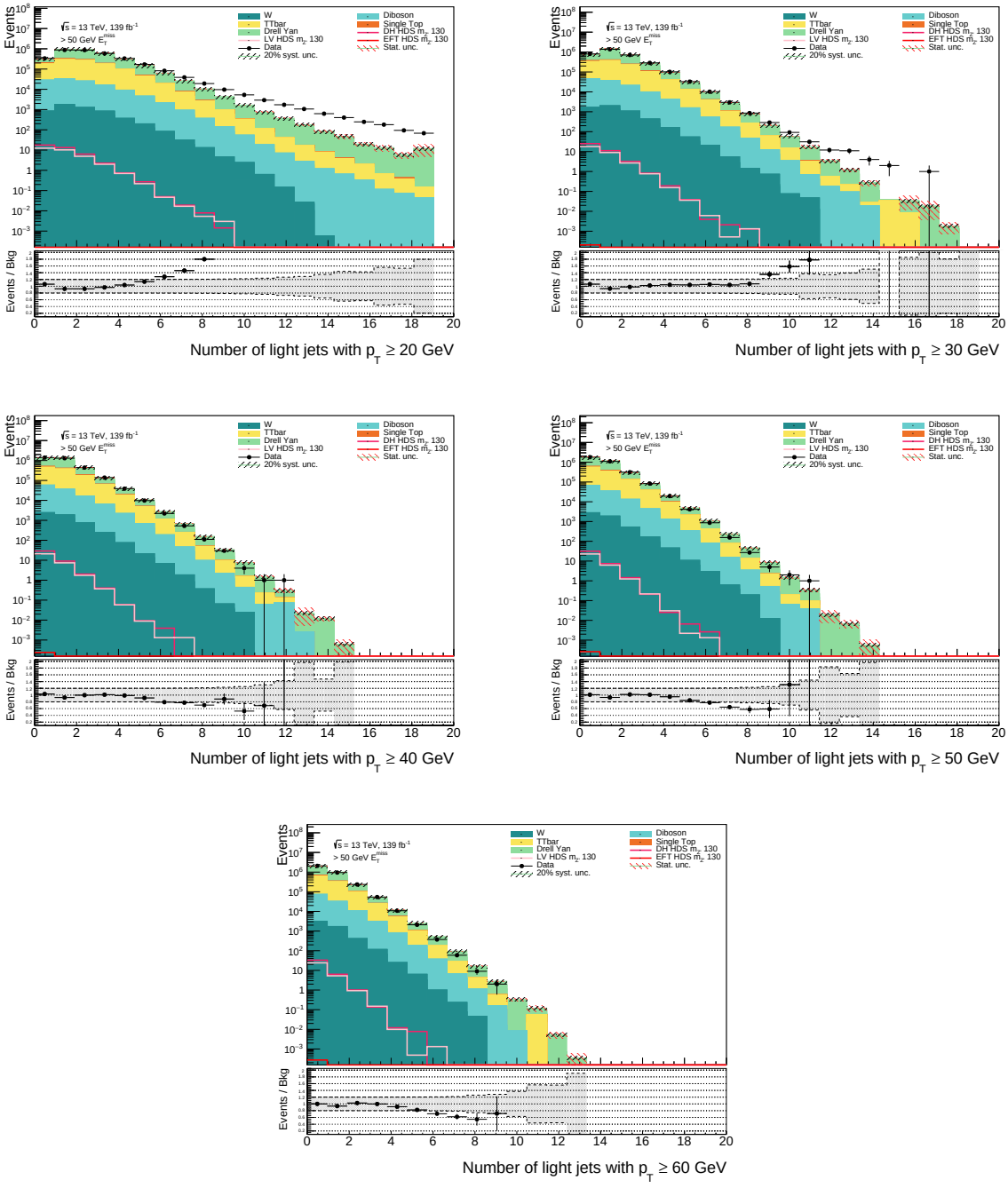


Figure C.2: Data and MC agreement on number of light jets with different p_T cuts in CR.

Appendix D

Data and MC agreement jets CR

to come...

Logbook

We are dropping $\Delta\phi(l_1, l_2)$ and $\Delta\phi(l_c, E_T^{miss})$ due the poor agreement between MC and data. The first one is most likely due us not including fake leptons (and also for all non SFOS final states). The latter is a problem that PhD. Even is being haunted by. There is also the problem of missing variables. For this theis, as shown in Table **??**, we will record events with up to three jets in the final state. As mentioned earlier, there isn't always three jets in the final state, to mediate this problem I chose to set the p_T to zero for the missing jets and m_{jj} to zero if there are less than two jets, this is something that is physically reasonable as it doesn't violate any conservation laws. More problematic however is the η and ϕ when there aren't jets. To mediate this I've set the values to -999, which has no physical meaning and is impossible to achieve, this I did so it becomes easier for us to identify the jagged arrays further in the network preparation. The missing variables is not a problem when making BDTs with XGBoost. There is also another problem, albeit less problematic than the previous ones, with the final states that are not SFOS, as the MC generated background on these tend to be lower than the recorded data. The number of events that are not SFOS are minimal though, and we think the reason it doesn't fit the data is because we are not including fake leptons.

I will remove this sentence, but its just so we know

something more to add?

Zp Dark Matter dataset

To train the networks I will utilize two methods. The first one being this where the dataset being sent into the ML network will contain every single DM MC sample available. So far there are 154 different MC samples, these are based on three theories. A Light Vector (LV), Dark Higgs (DH) and Effective Field Theory (EFT) which produces the WIMP DM particles, and a new theoretical particle, Z', that decays into a lepton pair. The three theories are divided further into MC samples with a Light Dark Sector (LDS) and High Dark Sector (HDS) which tells us therange of the Dark Matter candidate mass. And lastly it is divided further into more MC samples with different masses for Z'. This dataset includes all of these samples such that the network learns Dark Matter in a model independent way.

see next comment for the two methods

"Ensemble" dataset / Model indepedence

Another approach is to make multiple datasets and combine the results of every network into a "big network". This is the second approach which I call ensemble modelling. The thought behind this is that when training a network using the full dataset it might only focus on a the more resonant models with fixed masses. Also, every different DM sample has different phenomenology, specially in the future when I will be receiving SUSY samples, meaning that it also might not train the network physics. Thus if we were to train a network one one sample at a time it would be the perfect scenario. However as will become apparent in Section **??**, the datasets (even the full DM dataset) are extremely unbalanced. To put some numbers, on each DM MC sample there are roughly 40,000 MC events, and for the SM background (with a massive MET $>$ 50GeV cut!) there are roughly 87,000,000 MC events. Factoring the weights to re-weight the MC events to expected events gives us an extremely low statistics dataset, which punishes the network for guessing correctly.

This subsection and is outdated with our current plan, and this will change when we discuss further how to implement the model-independent aspect.

THIS IS OUTDATED AND WILL MOST LIKELY BE CHANGED TO THE SIGNAL REGION APPROACH, WHERE WE STATISTICALLY COMBINE WHAT THE NETWORKS LEARN IN A MODEL DEPENDENT WAY Thus making the approach to teach the networks one MC sample at a time is impossible. So far I have tried dividing the the MC samples into 18 different categories. First into their respective theory. Then into LDS or HDS. Then into three $m_{Z'}$ regions, where I've defined the *low mass region* to be $\leq$ 600 GeV, the *middle mass region* to be $>$ 600 $\cap \leq$ 1100 GeV and the *high mass region* to be $\geq$ 1100 GeV. Using a NN with three hidden layers I get

poor results, but changing this into one works! Will repeat with real weights, it didn't work.

Interesting articles

- From "Search for a charged Higgs boson decaying into a top and a bottom quark at $\sqrt{s} = 13$ TeV" [40] they say (chap 4.2): "The signal samples are trained against all backgrounds, which are weighted according to their cross-sections."
-

Bibliography

1. Bertone G, Hooper D, and Silk J. Particle dark matter: Evidence, candidates and constraints. Phys. Rept. 2005; 405:279–390. DOI: 10.1016/j.physrep.2004.08.031. arXiv: hep-ph/0404175
2. Einasto J. Dark Matter. 2010. arXiv: 0901.0632 [astro-ph.CO]
3. Elgammal S, Louka M, Ellithi AY, and Hussein MT. Search for the production of dark matter candidates in association with heavy dimuon resonance using the CMS open data for pp collisions at $\sqrt{s} = 8$ TeV. 2021 Sep. arXiv: 2109.11274 [hep-ex]
4. Autran M, Bauer K, Lin T, and Whiteson D. Searches for dark matter in events with a resonance and missing transverse energy. Phys. Rev. D 2015 Aug; 92(3):035007. DOI: 10.1103/PhysRevD.92.035007. Available from: https://link.aps.org/doi/10.1103/PhysRevD.92.035007
5. Aaboud M et al. Search for dark matter in events with a hadronically decaying vector boson and missing transverse momentum in pp collisions at $\sqrt{s} = 13$ TeV with the ATLAS detector. JHEP 2018; 10:180. DOI: 10.1007/JHEP10(2018)180. arXiv: 1807.11471 [hep-ex]
6. Thomson M. Modern Particle Physics. Cambridge University Press, 2013
7. Jackson JD. "Kinematics". Particle Data Group 2008. Available from: https://pdg.lbl.gov/2008/reviews/kinemarpp.pdf
8. Vadla KOH. Search for production of charginos and neutralinos in dilepton final states with the ATLAS detector at the LHC. PhD thesis. University of Oslo, 2022. Available from: https://www.duo.uio.no/handle/10852/97304
9. Barr A, Lester C, and Stephens P. A variable for measuring masses at hadron colliders when missing energy is expected; m_{T2}: the truth behind the glamour. Journal of Physics G: Nuclear and Particle Physics 2003 Sep; 29:2343–63. DOI: 10.1088/0954-3899/29/10/304. Available from: https://doi.org/10.1088%2F0954-3899%2F29%2F10%2F304
10. Martin AD, Stirling WJ, Thorne RS, and Watt G. Parton distributions for the LHC. The European Physical Journal C 2009 Jul; 63:189–285. DOI: 10.1140/epjc/s10052-009-1072-5. Available from: https://doi.org/10.1140%2Fepjc%2Fs10052-009-1072-5
11. Jadach S, Płaczek W, Sapeta S, Siodmok A, and Skrzypek M. Parton distribution functions in Monte Carlo factorisation scheme. Eur. Phys. J. C 2016; 76. Comments: 25 pages, 6 figures:649. DOI: 10.1140/epjc/s10052-016-4508-8. arXiv: 1606.00355. Available from: https://cds.cern.ch/record/2157419
12. The ATLAS Collaboration. The ATLAS Experiment at the CERN Large Hadron Collider. JINST 2008; 3. Also published by CERN Geneva in 2010:S08003. DOI: 10.1088/1748-0221/3/08/S08003. Available from: https://cds.cern.ch/record/1129811
13. Pequenao J and Schaffner P. How ATLAS detects particles: diagram of particle paths in the detector. 2013. Available from: https://cds.cern.ch/record/1505342
14. ATLAS Collaboration. The ATLAS inner detector trigger performance in pp collisions at 13 TeV during LHC Run 2. Eur. Phys. J. C 2022; 82:206. DOI: 10.1140/epjc/s10052-021-09920-0. arXiv: 2107.02485 [hep-ex]
15. Guevara R. The HiggsML and a simple Quantum-enhanced Machine Learning algorithm. 2022. Available from: https://github.com/rubenguevara/QuantumMachineLearning
16. ATLAS-Collaboration. Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC. Physics Letters B 2012 Sep; 716:1–29. DOI: 10.1016/j.physletb.2012.08.020. Available from: https://doi.org/10.1016%2Fj.physletb.2012.08.020

17. ATLAS Collaboration. Search for new particles in events with one lepton and missing transverse momentum in pp collisions at $\sqrt{s} = 8$ TeV with the ATLAS detector. Journal of High Energy Physics 2014 Sep; 2014. DOI: 10.1007/jhep09(2014)037. Available from: https://doi.org/10.1007%5C%2Fjhep09%5C%282014%5C%29037

18. Cowan G. Discovery sensitivity for a counting experiment with background uncertainty. Tech. rep. http://www.pp.rhul.ac.uk/∼cowan/stat/medsig/medsigNote.pdf. London: Royal Holloway, 2012. Available from: http://www.pp.rhul.ac.uk/~cowan/stat/medsig/medsigNote.pdf

19. Baldi P, Cranmer K, Faucett T, Sadowski P, and Whiteson D. Parameterized neural networks for high-energy physics. The European Physical Journal C 2016 Apr; 76. DOI: 10.1140/epjc/s10052-016-4099-4. Available from: https://doi.org/10.1140%2Fepjc%2Fs10052-016-4099-4

20. Baldi P, Sadowski P, and Whiteson D. Searching for exotic particles in high-energy physics with deep learning. Nature Communications 2014 Jul; 5. DOI: 10.1038/ncomms5308. Available from: https://doi.org/10.1038%2Fncomms5308

21. Hjorth-Jensen M. Week 40: From Stochastic Gradient Descent to Neural networks. 2021 Nov. Available from: https://compphysics.github.io/MachineLearning/doc/pub/week40/html/week40.html

22. Hjorth-Jensen M. Week 41 Constructing a Neural Network code, Tensor flow and start Convolutional Neural Networks. 2022 Aug. Available from: https://compphysics.github.io/MachineLearning/doc/pub/week41/html/week41.html

23. Hjorth-Jensen M. Applied Data Analysis and Machine Learning, 6. Logistic Regression. 2021 Aug. Available from: https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/chapter4.html

24. Morgan JN and Sonquist JA. Problems in the Analysis of Survey Data, and a Proposal. Journal of the American Statistical Association 1963; 58:415–34

25. Quinlan JR. Induction of Decision Trees. Mach Learning 1986; 1:81–106. DOI: https://doi.org/10.1007/BF00116251

26. Friedman JH. Greedy function approximation: A gradient boosting machine. The Annals of Statistics 2001; 29:1189–232. DOI: 10.1214/aos/1013203451. Available from: https://doi.org/10.1214/aos/1013203451

27. Adam-Bourdarios C, Cowan G, Germain C, Guyon I, Kégl B, and Rousseau D. The Higgs boson machine learning challenge. *Proceedings of the NIPS 2014 Workshop on High-energy Physics and Machine Learning*. Ed. by Cowan G, Germain C, Guyon I, Kégl B, and Rousseau D. Vol. 42. Proceedings of Machine Learning Research. Montreal, Canada: PMLR, 2015 Dec :19–55. Available from: https://proceedings.mlr.press/v42/cowa14.html

28. Chen T and Guestrin C. XGBoost: A Scalable Tree Boosting System. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA: ACM, 2016 :785–94. DOI: 10.1145/2939672.2939785. Available from: http://doi.acm.org/10.1145/2939672.2939785

29. Hjorth-Jensen M. Applied Data Analysis and Machine Learning, 9. Decision trees, overarching aims. 2021 Aug. Available from: https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/chapter6.html#

30. Hjorth-Jensen M. Applied Data Analysis and Machine Learning, 10. Ensemble Methods: From a Single Tree to Many Trees and Extreme Boosting, Meet the Jungle of Methods. 2021 Aug. Available from: https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/chapter7.html

31. Gleisberg T, Höche S, Krauss F, Schönherr M, Schumann S, Siegert F, and Winter J. Event generation with SHERPA 1.1. Journal of High Energy Physics 2009 Feb; 2009:7–7. DOI: 10.1088/1126-6708/2009/02/007. Available from: https://doi.org/10.1088%2F1126-6708%2F2009%2F02%2F007

32. Alioli S, Nason P, Oleari C, and Re E. A general framework for implementing NLO calculations in shower Monte Carlo programs: the POWHEG BOX. Journal of High Energy Physics 2010 Jun; 2010. DOI: 10.1007/jhep06(2010)043. Available from: https://doi.org/10.1007%2Fjhep06%282010%29043

33. Sjöstrand T, Ask S, Christiansen JR, Corke R, Desai N, Ilten P, Mrenna S, Prestel S, Rasmussen CO, and Skands PZ. An introduction to PYTHIA 8.2. Computer Physics Communications 2015 Jun; 191:159–77. DOI: 10.1016/j.cpc.2015.01.024. Available from: https://doi.org/10.1016%2Fj.cpc.2015.01.024

34. Brun R and Rademakers F. ROOT: An object oriented data analysis framework. Nucl. Instrum. Meth. A 1997; 389. Ed. by Werlen M and Perret-Gallix D:81–6. DOI: 10.1016/S0168-9002(97)00048-X

35. McKinney W. Data Structures for Statistical Computing in Python. *Proceedings of the 9th Python in Science Conference*. Ed. by Walt S van der and Millman J. 2010 :56–61. DOI: 10.25080/Majora-92bf1922-00a

36. Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Jia Y, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. Software available from tensorflow.org. 2015. Available from: https://www.tensorflow.org/

37. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, and Duchesnay E. Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research 2011; 12:2825–30

38. Danziger K, Höche S, and Siegert F. Reducing negative weights in Monte Carlo event generation with Sherpa. 2021. DOI: 10.48550/ARXIV.2110.15211. Available from: https://arxiv.org/abs/2110.15211

39. Abbott B, Alhroob M, Bhopatkar VS, Hopkins WH, Lambert JE, Metcalfe J, Serkin L, Xu W, and Zhu J. Search for Triboson $W^{\pm}W^{\mp}W^{\mp}$ Production Using 13 TeV pp Collision Data at ATLAS. Tech. rep. Geneva: CERN, 2020. Available from: https://cds.cern.ch/record/2714377

40. Benchekroun D, Bosman M, Chadi Z, Mir LM, Riu I, Salvador Salas A, Sato K, Yamauchi H, and Glatzer JMV. Search for a charged Higgs boson decaying into a top and a bottom quarks at $\sqrt{s}$=13 TeV. Tech. rep. Geneva: CERN, 2020. Available from: https://cds.cern.ch/record/2710015