

Model independent search for Dark Matter using Machine Learning with dilepton and Missing Transverse Energy final states 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 with
dilepton and Missing Transverse Energy
final states with the ATLAS detector at
the LHC

Supervisors:
Professor Farid Ould-Saada
Dr. Eirik Gramstad

Acknowledgements

Thank you everybody<3<3

Abstract

Something something

Contents

I	The theory behind modern particle physics / Background	1
1	The Standard Model of Particle Physics	3
1.1	Quantum Electrodynamics	3
1.2	Quantum Chromodynamics	3
1.3	Electroweak theory and the Brout-Englert-Higgs Mechanism	3
1.4	Adding it all up	3
2	Dark Matter	5
2.1	Observations of existence	5
2.1.1	Cosmology	5
2.1.2	WIMP	5
2.2	Beyond Standard Model candidates	5
2.2.1	New gauge boson	5
2.2.2	Supersymmetry	5
3	Production, Detection and Analysis	7
3.1	Particle production	8
3.1.1	Particle kinematics	8
3.1.2	Proton-proton collisions	10
3.2	The ATLAS detector	11
3.2.1	Inner detector	11
3.2.2	The calorimeters	12
3.2.3	Muon spectrometer	12
3.2.4	Summary of ATLAS	13
3.3	Data analysis	14
3.3.1	Cut and count	14
3.3.2	Statistical analysis	15
3.4	Summary	18
4	Machine Learning	19
4.1	Neural Networks	20
4.1.1	Artificial neurons	20
4.1.2	Stochastic Gradient Descent	20
4.1.3	Feed Forward network	21
4.1.4	Back Propagation algorithm	22
4.1.5	Summary of idea	24
4.2	Boosted Decision Trees	25
4.3	Tools used for both algorithms	26
4.3.1	Activation functions	26
4.3.2	Cost functions	26
4.3.3	Sample weight	27
4.3.4	Network evaluation methods	27
4.4	The declaration of model independence	28

II	Implementation / Methods	29
5	Data Analysis	31
5.1	Standard Model Background Estimation	31
5.2	Kinematic Variables	32
5.3	Dark Matter samples	34
6	Machine Learning	35
6.1	Data Preparation/LOG	35
6.1.1	Zp Dark Matter dataset	35
6.1.2	"Ensemble" dataset / Model indepedence	35
6.2	Neural Network Training	36
6.2.1	Padding of data	36
6.2.2	Normalization of data	39
6.2.3	Weights	42
6.2.4	Balanced weights	45
6.2.5	Architecture	49
6.2.6	Grid Search	50
6.3	Boosted Desicion Tree Training	54
6.3.1	Weights	54
6.3.2	Grid Search	56
III	Results	63
7	Comparison to cut and count	65

This whole section is my reasoning for trying to re-weight the MC events to expected events. But is just plain wrong.

List of Figures

3.1	The ATLAS detector	11
3.2	Illustration of the ATLAS detector layers	12
3.3	The Higgs discovery on the $ZZ^{(*)}$ channel	14
3.4	Confidence Limit on the Higgs discovery	17
4.1	Basic Neural Network Illustration	22
6.1	Network performace when testing new variables to avoid padding	38
6.2	NN prediction when using different normalization methods. This is testing a dataset with a Z' DM model.	40
6.3	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.	41
6.4	Importane of correctly scaling MC events to expected events	42
6.5	Result of the different network training weighting.	43
6.6	Result of the different network training weighting.	44
6.7	NN comparison of trained models in FULL Z' dataset	45
6.8	Correctly scaled plots.	45
6.9	NN prediction when weighting the signal with the SOW ratio.	46
6.10	NN prediction when weighting the signal with the raw event ratio.	46
6.11	NN prediction when weighting the background with the SOW ratio.	47
6.12	NN prediction when weighting the bacgkround with the raw event ratio.	47
6.13	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.	48
6.14	Neural Network Architecture	49
6.15	Grid search significance with $\lambda = 10^{-5}$ and $n_layers = 2$	50
6.16	Grid search AUC with $\lambda = 10^{-5}$ and $n_layers = 2$	51
6.17	Grid search significance with $\lambda = 10^{-5}$ and $\eta = 0.01$	51
6.18	Grid search significance with $\lambda = 10^{-5}$ and $\eta = 0.01$	52
6.19	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.	52
6.20	Difference when using different weighting methods. All networks were trained using the balancing method explained in Section 6.2.3	55
6.21	Grid search expected significance going to a depth of up to 30	56
6.22	Grid search AUC going to a depth of up to 30	57
6.23	Feature importance of depth 30 network trained on FULL Z' DM data set when testing it on DH HDS $m_{Z'} = 130$ GeV model.	58
6.24	Grid search expected significance when setting $\lambda = 10^{-5}$ and $\eta = 0.1$	59
6.25	Grid search AUC when setting $\lambda = 10^{-5}$ and $\eta = 0.1$	59
6.26	Feature importance of depth 30 network trained on FULL Z' DM data set when testing it on DH HDS $m_{Z'} = 130$ GeV model.	60
6.27	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.	61

7.1	Expected significance of XGBoost when trained on the Full DM dataset for the DH HDS $m_{Z'} = 130$ GeV muon model.	66
7.2	Expected significance of the Neural Network when trained on the Full DM dataset for the DH HDS $m_{Z'} = 130$ GeV muon model.	67

List of Tables

5.1	Cuts for model-independent search	31
5.2	Kinematic variables used as features	33
5.3	Kinematic variables that need padding	34
6.1	New kinematic variables that need no padding	37
6.2	Unbalanced Diboson training dataset	42
6.3	Imbalance raw events and SOW	46
7.1	Cut and count cuts	65
7.2	Cut and count significance ee	65
7.3	Cut and count significance uu	66
7.4	Cut and count significance ee	69
7.5	Unbalanced DM training dataset	69

List of Acronyms

SM Standard Model

DM Dark Matter

LHC Large Hadron Collider

MET Missing Transverse Energy

ML Machine Learning

NN Neural Network

BDT Boosted 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 combining all these groups we get spontaneous symmetry breaking resulting in the Brout-Englert-Higgs Mechanism. The whole lagrangian is of the form

$$U(1)_Y \otimes SU(2)_L \otimes SU(3)_C \Rightarrow$$
$$\mathcal{L}_{SM} = -\frac{1}{4}F_{\mu\nu}F^{\mu\nu} + i\bar{\Psi}\not{D}\Psi + \psi_i y_{ij} \psi_j \phi + h.c. + |D_\mu \phi|^2 - V(\phi) \quad (1.1)$$

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

2.1.1 Cosmology

2.1.2 WIMP

2.2 Beyond Standard Model candidates

2.2.1 New gauge boson

2.2.2 Supersymmetry

Chapter 3

Production, Detection and Analysis

Now that we have established the necessary theoretical groundwork of particle physics, 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?

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.

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 [1], Jacksons "Kinematics" [2]. and Vadlas PhD. thesis [3].

3.1.1 Particle kinematics

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 detectors, that means we will use the polar angle, θ , and the azimuthal angle, ϕ , such that we have

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

where p_T is the *transverse momentum* expressed as

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

The energy and momentum can be expressed in relativistic 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 contracting² 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

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

As this thesis will focus on a dilepton final (and missing transverse energy) final state, which is of the type $2 \rightarrow 2 (+MET)$ 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 directly from the same equation

$$E_T = \sqrt{m^2 + p_T^2} \quad (3.4)$$

The invariant mass is what we measure in the final state only. But as we are going to use data³ from the LHC, from which the initial state is made controlled by us, it will be of interest to see what the total energy and momentum of the two protons is. 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 being that we look at the initial particles. For this thesis we will look at data and simulations of Run II from the LHC, which had $\sqrt{s} = 13$ TeV.

As this thesis aims to search for DM, which we know does not interact with matter in the same way as neutrinos⁴, 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

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

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.

¹As this is a particle physics thesis I will convert to Natural Units where we set $c = 1$

²Using the particle physicists convention of the minkowsky metric tensor $\eta_{\mu\nu}$, (+, -, -, -)

³And mostly simulations mimicking the ATLAS detector

⁴Also called *missing momentum*

Is this comment unnecessary?

add sources

Meaning that significant MET is therefore indicative of the presence of an undetected particle.

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

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

this gives a measurement of the hadronic energy scale of an event. Another handy trick comes from the realization that since the centre-of-mass frame is between the hadrons, where the total momentum is given as a function of the energy of the hadron. Meaning that the final state particles are boosted along the beam axis. With this realization we can now introduce a Lorentz invariant⁵ kinematic property known as the *rapidity*, y used to express the lepton angles

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

where we can use $p_z = E \cos \theta$ as we neglect the mass in the high-energy limit. In this limit we can use the *pseudorapidity*, η , defined By

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

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 circle in (η, ϕ) -space surrounding the object of interest. It is defined as

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

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

$$m_T^2 = m^2 + p_T^2 \quad (3.10)$$

where m^2 is the invariant 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 q_T . This is called the *stransverse mass* and is defined by

$$m_{T2}^2(\chi) = \min_{\mathbf{q}_T^{(1)} + \mathbf{q}_T^{(2)} = \mathbf{p}_T} \left[\max \left\{ m_T^2 \left(\mathbf{p}_T^{\ell_1}, \mathbf{q}_T^{(1)}; \chi \right), m_T^2 \left(\mathbf{p}_T^{\ell_2}, \mathbf{q}_T^{(2)}; \chi \right) \right\} \right] \quad (3.11)$$

where \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. [4]. 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 $W \rightarrow l + \nu_l$ 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 $m_{T2} > m_W$.

This sentence is inspired from Elizabeth Christensen, but I don't want to repeat what she said...

⁵Under 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 produced still remains. The answer to that is protons. The way we produce elementary particles to study is by colliding two protons together. The reason as to why this works is because protons are also made of elementary particles, two *up* and one *down* to be specific⁶. Because of this it is not hard to realize that the Feynman rules acquired from the SM also apply here.

⁶As well as gluons and partons existing inside of it

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 known as the ATLAS detector. Figure 3.1 showcases the detector and its size. The information of this section is largely based on the original ATLAS Collaboration introduction paper presented to CERN [5]. (and Knuts thesis [3] that described it in a more understandable way).

from my understanding, correct me if wrong!

The ATLAS (A Toroidal LHC ApparatuS) detector is a general multi-purpose⁷ detector located at the LHC and covers nearly the entire solid angle around the collision point, as described with (η, ϕ) -coordinates in Section 3.1.1. The ATLAS detector consists of four main subdetectors; (i) an inner tracking detector (ID), an (ii) electromagnetic calorimeter (ECAL), and a (iii) hadronic calorimeter (HCAL), and lastly (iv) a muon spectrometer (MS). Figure 3.2 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 through Section 3.2.1-3.2.3.

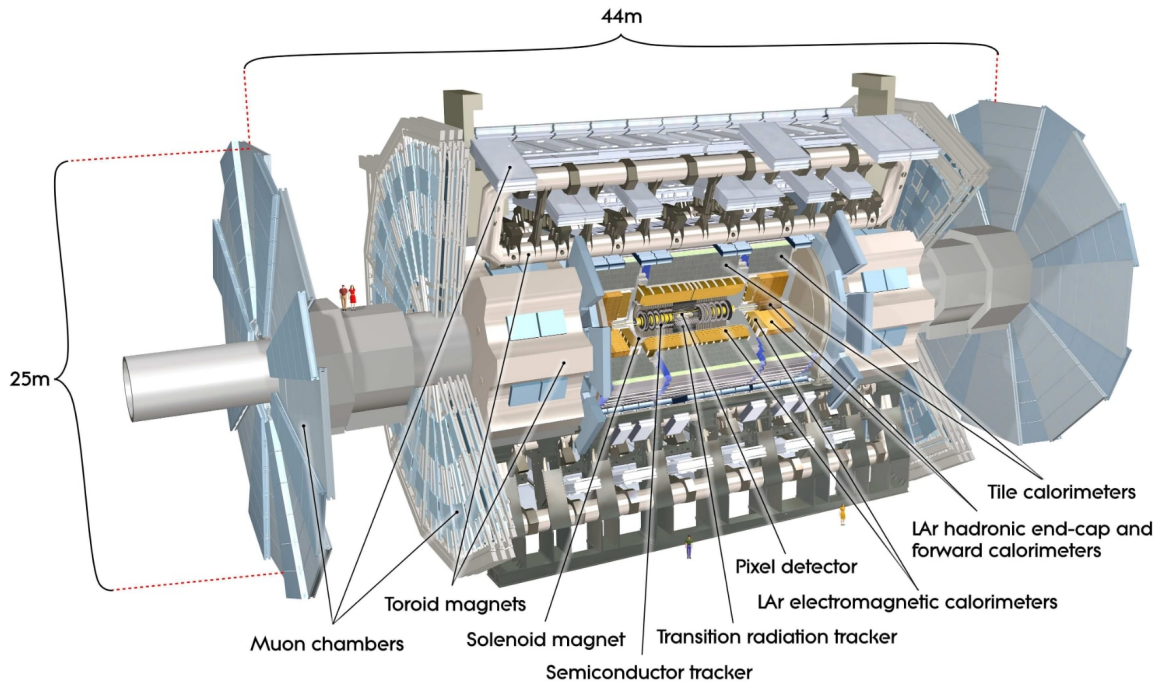


Figure 3.1: Cut-away view of the ATLAS detector, image taken from Ref. [5]

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 silicon 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 silicon 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 $|\eta| = 2.0$.

⁷Probing p - p and A - A (heavy ions) collisions.

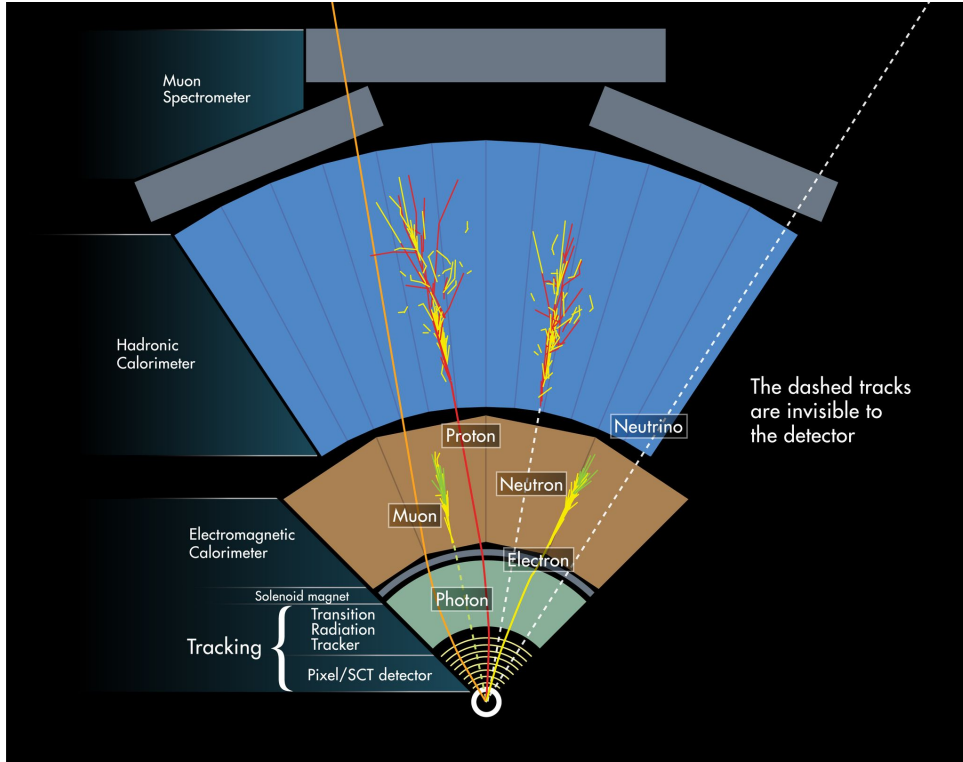


Figure 3.2: Illustration of the ATLAS detector layers, image taken from Ref. [6]

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 surrounding 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 lengths with lead.

The HCAL is immediately surrounding 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 maximize the interaction 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 spectrometer* (MS), dedicated to the measurement of the muons momenta. The MS, similar to the ID, consists of multiple layers of detector material, and is immersen 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 triggers. 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

We can now dive into how particles are identified now that the main four parts of the ATLAS detector; *inner-detector* (ID), *electromagnetic calorimeter* (ECAL), *hadronic calorimeter* (HCAL), and the *muon spectrometer* (MS), have briefly been explained. For an easier understanding on the different parts I refer the reader again to Figure 3.2 which shows visually how different particles interact with the detector. We can start by noting tht 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)⁸. 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 now interacts with the neutrons, giving us information about their energies (and location for neutrons). Lastly we see that the muon flies 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 3.2, 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 however, 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.

Question to Farid: Is it okay for me to do this and move on? Or should I write about this?

find sources

⁸So as we know that the proton is positively charged, then we know that the muon and electron in Figure 3.2 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 physics 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 [7]. To give a short description of the cut and count method, it makes kinematical *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 experimental physics discovery without making a statistical analysis of the *significance* of the *cut signal region* we will also explain how we measure these. To guide us through this process I will use $ZZ^{(*)}$ channel in the discovery of the Higgs Boson in 2012 [8] as an example of the success of these methods.

3.3.1 Cut and count

The cut and count method is what is currently 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. If we take the Higgs discovery, in particular the $H \rightarrow ZZ^{(*)} \rightarrow 4l$ channel, then we can as example straight away make one kinematical cut that will decrease most SM backgrounds, that is requiring that there are four leptons in the final state! We can also check with the detector (and simulation) that there needs to specifically be muons and/or electrons in the final state. Furthermore the four leptons should respectively have a transverse momentum greater than 25, 15, 10, 7 GeV *why? these cuts are from ATLAS Open Data, can I find a source for the actual cuts used in the 2012 discovery?* To summarize the cuts made in the cut and count method, we have:

- Single-electron or single muon trigger satisfied
- Exactly four leptons with $p_T > 25, 15, 10, 7$ GeV, respectively
- Higgs-boson candidates are formed by selecting two same flavour opposite charge lepton pairs

Using these cuts, as well as the standard criteria cuts (see Chapter 5) what remains is to count the remaining events. 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. The remaining events after these cuts is shown in Figure 3.3.

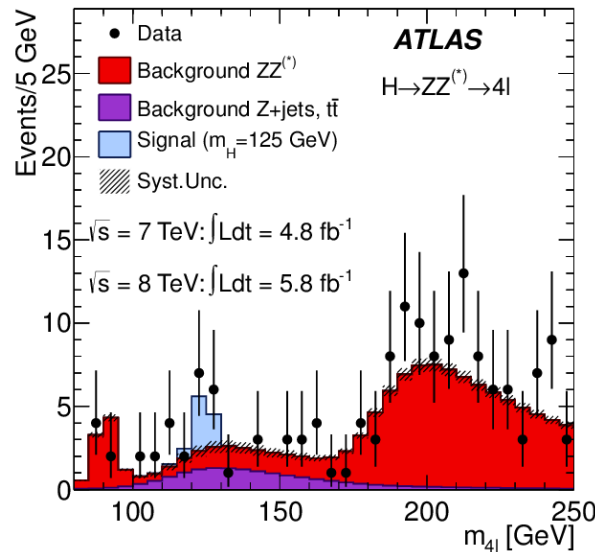


Figure 3.3: The Higgs discovery on the $ZZ^{(*)}$ channel, image taken from Ref. [8]

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. What 5σ means, and how we can get there is what this subsection is all about. But as not everything in experimental high energy particle physics is a new discovery, but is in fact for the most part *exclusions*. Then we need to explain what an exclusion is, and how to make them.

add source
to why 5?

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" [9], where a Bayesian analysis is performed to set limits on the studied processes. Going back to the discovery of the Higgs boson, for each candidate mass and decay channel, events are counted above a m_{ll} threshold. Using the signal+background hypothesis, the expected number of events in each channel 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

$$\mathcal{L}(N_{\text{obs}}|\sigma B) = \frac{(N_{\text{exp}})^{N_{\text{obs}}} e^{-N_{\text{exp}}}}{N_{\text{obs}}!} \quad (3.12)$$

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

$$\mathcal{L}_B(N_{\text{obs}}|\sigma B) = \int \mathcal{L}(N_{\text{obs}}|\sigma B) \prod g_i(\theta_i) d\theta_i \quad (3.13)$$

where a log-normal distribution is used for the $g_i(\theta_i)$. The nuisance parameters are taken to be: L_{int} , ε_{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 $H \rightarrow ll'l'$) are combined assuming the same branching fraction for each, thus Eq. 3.13 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 :

$$P_{\text{post}}(\sigma B|N_{\text{obs}}) = N \mathcal{L}_B(N_{\text{obs}}|\sigma B) P_{\text{prior}}(\sigma B) \quad (3.14)$$

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 assumed to be $\pm 30\%$ of the background, as this is roughly the standard value of other ATLAS papers.

To make exclusions we can use Eq. (3.14) to establish a *confidence limit* (CL). CLs are defined as the probability to observe the number of events observed in an experiment, N_{obs} , or *less* given signal+background. We usually define a signal+background hypothesis to be excluded when $\text{CL}_{s+b} < 5\%$. 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, rather than exclude them.

On the other side, to claim any discovery in particle physics we need to know the *significance* of any statistical fluctuation. 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

$$p = P(N \geq N_{\text{obs}} | \lambda = N_{\text{bkg}}) = \sum_{k=N_{\text{obs}}}^{\infty} \mathcal{L}(k | N_{\text{bkg}}) \quad (3.15)$$

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 define 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 probability 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

$$Z = \sqrt{2 \left[N_{\text{obs}} \ln \frac{N_{\text{obs}}}{N_{\text{bkg}}} - N_{\text{obs}} + N_{\text{bkg}} \right]} \quad (3.16)$$

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

$$Z = \sqrt{2 \left[(N_{\text{sig}} + N_{\text{bkg}}) \ln \left(1 + \frac{N_{\text{sig}}}{N_{\text{bkg}}} \right) - N_{\text{sig}} \right]} \quad (3.17)$$

However, Eq. (3.15) did not include any nuisance parameters, it used Eq. (3.12) instead of Eq. (3.13). We want to express the significance with uncertainties. From "Discovery sensitivity for a counting experiment with background uncertainty" from Glen Cowan [10], 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

$$Z = \sqrt{-2 \left(N_{\text{obs}} \ln \left[\frac{N_{\text{obs}} + \frac{N_{\text{bkg}}^2}{\sigma_{\text{bkg}}^2}}{(1 + \frac{N_{\text{bkg}}}{\sigma_{\text{bkg}}^2})N_{\text{obs}}} \right] + \frac{N_{\text{bkg}}^2}{\sigma_{\text{bkg}}^2} \ln \left[\frac{\frac{N_{\text{bkg}}}{\sigma_{\text{bkg}}^2} (N_{\text{obs}} + \frac{N_{\text{bkg}}^2}{\sigma_{\text{bkg}}^2})}{(1 + \frac{N_{\text{bkg}}}{\sigma_{\text{bkg}}^2}) \frac{N_{\text{bkg}}^2}{\sigma_{\text{bkg}}^2}} \right] \right)} \quad (3.18)$$

Which makes for a better estimate of the significance one has in reality.

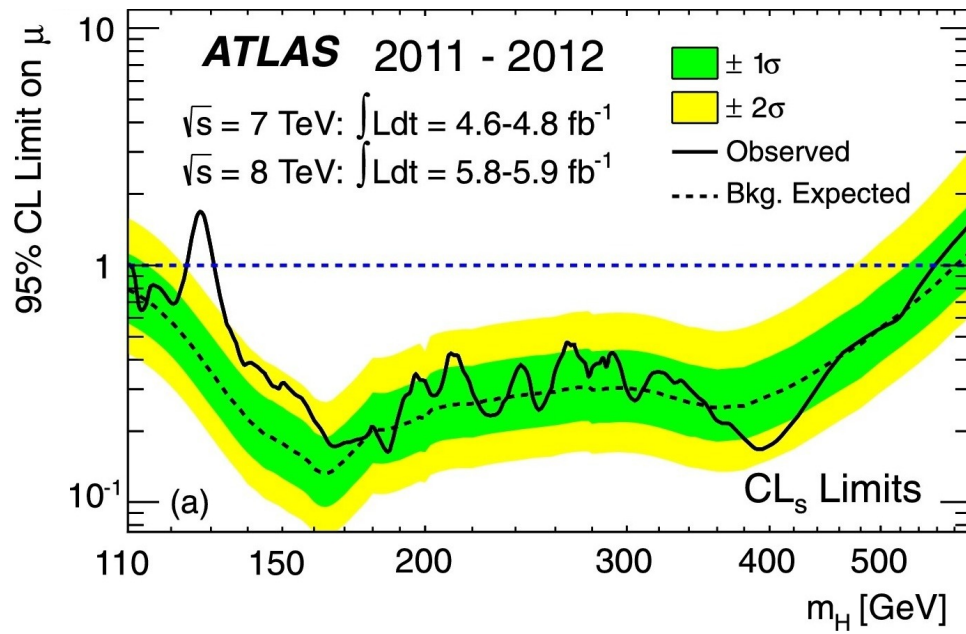


Figure 3.4: Confidence limit on the 2012 Higgs discovery, excerpt taken from Ref. [8]

3.4 Summary

Add more about kinematic variables, also mention classical data analysis and setting limits

We can now dive into how particles are identified now that the main four parts of the ATLAS detector; *inner-detector* (ID), *electromagnetic calorimeter* (ECAL), *hadronic calorimeter* (HCAL), and the *muon spectrometer* (MS), have briefly been explained. For an easier understanding on the different parts I refer the reader again to Figure 3.2 which shows visually how different particles interact with the detector. We can start by noting tht 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)⁹. 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 now interacts with the neutrons, giving us information about their energies (and location for neutrons). Lastly we see that the muon flies 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 3.2, 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 however, 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.

Question to Farid: Is it okay for me to do this and move on? Or should I write about this?

find sources

⁹So as we know that the proton is positively charged, then we know that the muon and electron in Figure 3.2 have negative charge

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 Machine Learning (ML) as its popular rise has proven to be effective at binary classification tasks [11, 12] 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 $\beta = \{\beta_1, \beta_2, \dots, \beta_n\}$ for a n-dimensional problem, which for our purposes can be seen as what are called *weights* and *biases* ($\beta = \{w, b\}$), the goal is to choose these parameters β such that we minimize a cost (also called loss) function $C(\beta)$ 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

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

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 \mathbf{t} . Then we give the network a score depending on how close the predicted output is to real values \mathbf{t} . Then we repeat the process after tweaking the parameters β 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 for granted that the reader is comfortable with linear algebra and jump straight into the mathematical foundations of ML.

4.1 Neural Networks

Before begining we can briefly explain the idea behind NNs. As stated by Hjorth-Jensen in [13]:

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

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

Where y , the output of the neuron, is the value of its *activation function* (See section 4.3.1), which has the weighted, w_i , sum of signals x_i, \dots, 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, called 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, \mathbf{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 \mathbf{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 β 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 $C(\beta)$ we can get closer to the minimum by calculating the gradient $\nabla_{\beta} C(\beta)$ wrt. the unknown parameters from the NN β . If we were to calculate the gradient at a specific point β_i in the parameter space, the negative gradient would correspond to the direction where a small change $d\beta$ 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 β such that it minimizes the cost function is given as

$$\beta_{i+1} = \beta_i - \eta \nabla_{\beta} C(\beta_i) \quad (4.3)$$

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 computationally 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 β_{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 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

$$y = f\left(\sum_{i=1}^n w_i x_i + b_i\right) = f(z) \quad (4.4)$$

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. (4.4) 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

$$z_i^1 = \sum_{j=1}^M w_{ij}^1 x_j + b_i^1 \quad (4.5)$$

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

$$y^{l+1} = f^{l+1}\left[\sum_{j=1}^{N_l} w_{ij}^{l+1} f^l\left(\sum_{k=1}^{N_{l-1}} w_{jk}^l \left(\cdots f^1\left(\sum_{n=1}^{N_0} w_{mn}^1 x_n + b_m^1\right) \cdots\right) + b_j^l\right) + b_i^{l+1}\right] \quad (4.6)$$

This shows that an MLP is nothing more than an analytic function, specifically a mapping of real-valued vectors $\hat{x} \in \mathbb{R}^n \rightarrow \hat{y} \in \mathbb{R}^m$. 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. (4.6) 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.

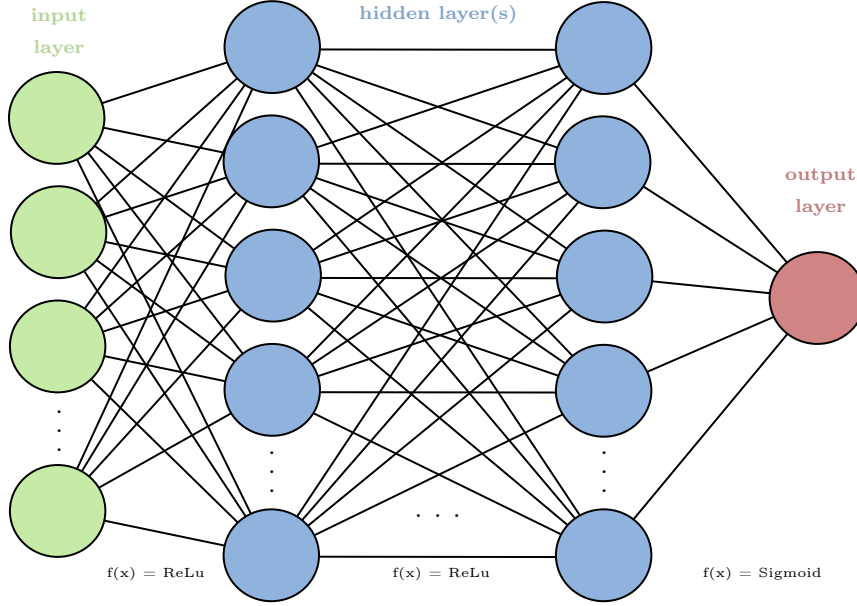


Figure 4.1: Basic illustration of a network with two hidden layers.

4.1.4 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

$$C(\hat{W}) = \frac{1}{2} \sum_{i=1}^n (y_i - t_i)^2 \quad (4.7)$$

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. (4.5) for a layer l

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

where the right side is written in matrix notation. From the definition of z_j^l with an activation function, Eq. (4.4), we have

$$\frac{\partial z_j^l}{\partial w_{ij}^l} = y_i^{l-1} \quad (4.8)$$

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. (4.14), gives us

$$\frac{\partial y_j^l}{\partial z_j^l} = y_j^l(1 - y_j^l) = f(z_j^l)(1 - f(z_j^l)) \quad (4.9)$$

Furthermore, we need to take the derivative of Eq. (4.7) with respect to the weights, doing so for a respective layer $l = L$ we have

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

where the last partial derivative is easily computed using the chain rule with Eq. (4.8) and Eq. (4.9)

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

Such that

$$\frac{\partial C(\hat{W}^L)}{\partial w_{jk}^L} = (y_j^L - t_j) y_j^L(1 - y_j^L)y_k^{L-1} := \delta_j^L y_k^{L-1} \quad (4.10)$$

where we have defined the error

$$\delta_j^L := (y_j^L - t_j) y_j^L(1 - y_j^L) = f'(z_j^L) \frac{\partial C}{\partial y_j^L} \quad (4.11)$$

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 product¹. This error δ^L is an important expression, since as we can see in the index form of this expression in Eq. (4.11), 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. (4.11) is easily computed. Thus we can also see how the weight changes the cost function using Eq. (4.10) quite easily. Another thing we can compute with Eq. (4.11) is

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

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

$$\delta_j^L = \frac{\partial C}{\partial b_j^L} \frac{\partial b_j^L}{\partial z_j^L} = \frac{\partial C}{\partial b_j^L} \quad (4.12)$$

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. (4.10 - 4.12) we see that if a neuron output y_j^L is small, then the gradient term, Eq. (4.10), 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

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

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. (4.11) 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. (4.5) (replacing 1 with $l + 1$) we get

$$\delta_j^l = \sum_k \delta_k^{l+1} w_{kj}^{l+1} f'(z_j^l) \quad (4.13)$$

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

4.1.5 Summary of idea

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. (4.6)
- Secondly we commit all the way in Eq. (4.6) and compute all \mathbf{z}_l , activation function and \mathbf{y}^l .
- After that we compute the output error δ^L by using Eq. (4.11) for all values j .
- Then we back propagate the error for each $l = L - 1, L - 2, \dots, 2$ with Eq. (4.13).
- 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

While the idea behind neural networks is to mimic the human brain, Boosted Decision Trees (BDTs) are different in the sense that they are not trying to mimic a brain, but rather split data sets. The idea of BDTs has been proposed as early as 2001 by Friedman [\[14\]](#), where bla bla.

4.3 Tools used for both algorithms

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

4.3.1 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 4.1.3, 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

$$f(x) = \frac{1}{1 + e^{-x}} \quad (4.14)$$

which is the most basic activation function, this will be used from the input layer to our first 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)

$$f(x) = x^+ = \max(0, x) = \begin{cases} x & \text{if } x > 0, \\ 0 & \text{otherwise.} \end{cases} \quad (4.15)$$

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

4.3.2 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 studying 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

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

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 β with the help of a logistic function. Using this we get the probability as

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

and

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

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

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

from which we obtain the log-likelihood

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

By taking the parameter β to first order in x_i and reordering the logarithm we get

$$C(\beta) = - \sum_{i=1}^n (y_i(\beta_0 + \beta_1 x_i) - \log(1 + \exp(\beta_0 + \beta_1 x_i))) \quad (4.16)$$

This equation we will use throughout 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.

more motivation

L2-regularization

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

$$C(\beta) \rightarrow C(\beta) + \lambda \sum_{ij} w_{ij}^2 \quad (4.17)$$

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.

4.3.3 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. (4.7) to

$$C(\hat{W}) = \frac{1}{2} \sum_{i=1}^n \chi_i (y_i - t_i)^2 \quad (4.18)$$

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

4.3.4 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 emphasis 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.

Area under the ROC-curve

To evaluate how well our networks do. We will use the accuracy, 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 explicate the ROC curves. The ROC curves helps us illustrate how successful 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 network 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 give an event a score from 0 to 1², 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

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

add source
for article?

write more
motivation?

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 unbalanced. As an 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 inclined 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 $\text{TPR} = 0$ and a $\text{FPR} = 0$, 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?

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 distribution of the network predictions.

It is standard practice to make fill the histograms 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 addition 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.

4.4 The declaration of model independence

Should I make this a whole chapter instead? As it is a central part of my thesis. And maybe move it into part 2, "Methods"

Part II

Implementation / Methods

Chapter 5

Data Analysis

Before preparing the ML networks we need to make the dataset that will be used to classify signal and background events, in this section we will discuss which kinematical cuts are used to define the signal region search.

more fluff

5.1 Standard Model Background Estimation

The SM backgrounds we will look at are W, TTbar, Drell Yan (Z + jets), Diboson, Single top. Since we are doing a dilepton DM search, which we expect to behave similarly to a neutrino, we need to make a cut on the missing transverse energy (MET) and require the final state to have a lepton pair. 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, meaning we will *only* look at events where the MET is greater than this. This both because of the amount of events that are in a dilepton final state, and also because most models studied in this thesis have roughly more MET than 50 GeV.

The way we define our lepton pairs is not by just looking at Same Flavour Opposite Sign (SFOS) leptons ($e^\pm e^\mp, \mu^\pm \mu^\mp$), but also all other possible combinations as these might be important for theories such as SUSY. We will also look at the not so well defined by MC; Different Flavour Same Sign (DFSS), DFOS and SFSS lepton pairs.

Other than the standard criteria cuts, these are the only cuts that will be used in this search.

	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	$E_T^{miss} > 50 \text{ GeV}$

Table 5.1: Table showcasing the cuts used for this search.

5.2 Kinematic Variables

For this thesis there are many possible kinematic variables that can be used as features for our ML algorithms.

As we require the final state to only have two leptons it is therefore natural to look at the kinematics for both of these. The first thing we will look at is the transverse momentum, p_T , of each lepton. We will also look at the azimuthal angle, ϕ , to know where in the detector the leptons are located. In addition we will look at the pseudorapidity, η , to know how close to the beam the leptons are. Given that we have two leptons in the final state it is natural to look at the invariant mass, m_{ll} , of these.

There are other kinematic variables of interest that arent directly related to the lepton pair. The most important for this kind this search is the missing transverse energy, E_T^{miss} , as this how we expect DM to be recorded. Another variation of this that takes into account the uncertainty of the detector is of interest, this is called for MET-significance, E_T^{miss}/σ . We will also study the transverse mass, m_T , and tranverse energy, E_T , recorded in the events. In addition we will look at a variable called *hadronic activity*, H_T , which is the scalar sum of the all the jets (including leptonic jets) in an event. We will also look at the ratio between the missing transverse energy and hadronic activity.

We willll also look at the number of b- and light jets. We will also look at a SUSY variable called the stransverse mass, m_{T2} , which as defined by Barr et.al. [4]. We will also look at the difference in azimuthal angle between: the lepton pair, $\Delta\Phi(l_1, l_2)$, the dilepton jet and MET jet, $\Delta\Phi(l_l, E_T^{miss})$, 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, $\Delta\Phi(l_c, E_T^{miss})$

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

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

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} calculation.

The kinematic variables used are summarized in Table 5.2 and the distribution showing the agreement between MC and data is shown in Appendix A.

check if you write this as "variables for ML" or not

write criteria for both

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}/σ	met_sig
Transverse mass in in event, m_T	mt
Stransverse mass in in event, m_{T2}	mt2
Transverse energy in in event, E_T	et
ϕ between lepton pair, $\Delta\Phi(l_1, l_2)^*$	dPhiLeps
ϕ between lepton 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	rt
Number of b-jets	nbjets
Number of light jets	nljets

Table 5.2: Table showcasing the kinematic variables that will be used as features.

* These have poor MC and data agreement.

There is also the possibility to include jet-related kinematic variables. But these will become a problem as they will create *jagged arrays*, these are arrays that might not always have a value in them. For instance, if we were to look at the η of the three jets with highest p_T in all of Run II then it becomes clear why this is a problem. The reason being that there might not always be three jets in an event! The problem with these jagged arrays is one that is better suited when discussing how to prepare our ML networks and will be discussed further there.

As of the jet kinematics, we will look at the p_T, ϕ and η of the three jets with highest p_T , we will also look at the invariant mass of the two jets with highest p_T , to not confuse this with the invariant of the dilepton pair, I will call this variable for m_{jj} . The jet kinematic variables used are summarized in Table 5.3 and the distribution showing the agreement between MC and data is shown in Appendix A.

Should I have an appendix showing the distribution of over 30 variables?

Kinematic variable	Feature name
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

Table 5.3: Table showcasing the kinematic variables that need padding.

5.3 Dark Matter samples

Chapter 6

Machine Learning

6.1 Data Preparation/LOG

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 5.3, 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 re-move this sentence, but its just so we know

something more to add?

6.1.1 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

6.1.2 "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 6.2.3, 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 sub-section 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 ≤ 600 GeV, the *middle mass region* to be $> 600 \cap \leq 1100$ GeV and the *high mass region* to be ≥ 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.

6.2 Neural Network Training

For this thesis we purely utilize **TensorFlow v. 2.7.1 GPU** for NN. Before implementing everything into a real NN we need to first prepare the data in a special way. The first and most important thing for this whole project is that the `batch_size` should be as big as possible whenever we try **anything** when 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.

The highest possible batch size that could be used for this thesis was 2^{24} which means that there are roughly 17 million samples pr. 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 takes greater computational power.

With this out of the way there are still a few things that needs to be done to get the best NN for our purposes. These have their own section as they are more difficult to tackle than the batch size.

6.2.1 Padding of data

There are two problems that need to be addressed when utilizing NNs as compared to BDTs. The first one which is the hardest one to solve, as no one has found a reliable solution yet, is the padding of jagged arrays. Padding is the process of filling missing values on a dataset that goes into a network. This is a problem that doesn't usually appear in other fields than high energy particle physics. To repeat the problem, in the dataset being used in this thesis, we include seven kinematic variables that might not even be there, these are the p_T , η and ϕ of the three highest p_T jets and the invariant mass m_{jj} of the two highest p_T jets. The obvious reason for this is that there might not be a jet in every event, thus we have missing variables. As previously mentioned, the way that the dataset was made, set the missing values of p_T and m_{jj} to zero and the values of η and ϕ to -999 if the jet of the variable was not present.

The jet p_T and m_{jj} being 0 is a valid form of padding the dataset, as this doesn't break any fundamental 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 η might be physically possible (in the future) but as of today the ATLAS detector has a $|\eta| < 4.9$ as the pseudorapidity states how close to the beamline the particles recorded are. However having a p_T of a jet equal to zero and still recording the η and ϕ breaks the laws of physics, so this is a problem that needs to be fixed.

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 "missing" values set). However this is not popular among physicists since it breaks conservation laws when we say there are jets when there really isn't. 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 pattern for the network which might lead to a bias.

Another approach is 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 physics, as this might remove precious signal events. One could also just remove the features with missing values

source needed

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.

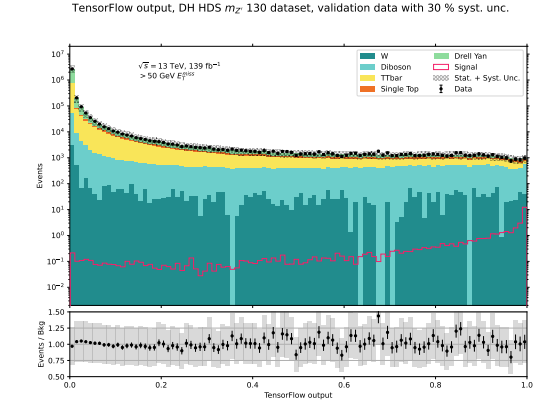
For this project we have tried to create new kinematic variables that work around the need of padding. These features are just *counting features*, meaning that they only count the number of jets that fulfill the criteria. In this project we decided to look at the number of b-jets with $p_T > 20$ GeV, the number of light jets with $p_T > 40$ GeV, the number of 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 reason why the light jets and forward calorimeter jets have a higher p_T criteria than the standard 20 GeV, is for the data and MC to agree. A summary of these variables is shown in Table 6.1.

should I
write this
at all

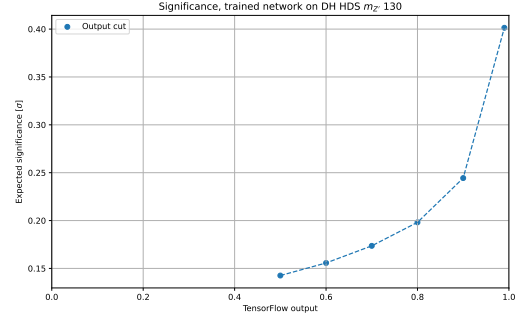
Kinematic variable	Feature name
Number of b-jets with $p_T > 20$ GeV	n_bjetPt20
Number of light-jets with $p_T > 40$ GeV	n_ljetPt40
Number of jets recorded in Central calorimeter	n_jetsetaCentral
Number of jets recorded in Forward calorimeter with $p_T > 50$ GeV	n_jetsetaForward50

Table 6.1: Table showcasing plausible kinematic variables that will not need padding.

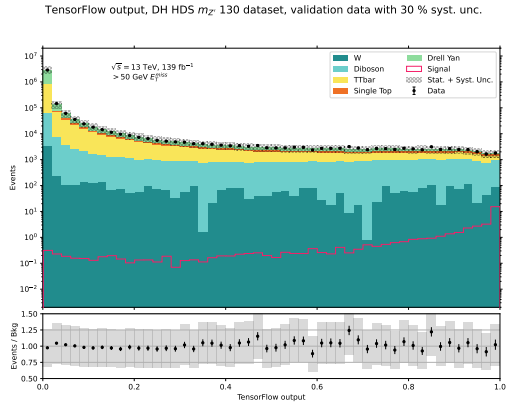
When training our NN with these new variables we would hope that it learns more physics by hopefully recognising patterns between all high level features. I have however tested this and the difference in a optimized network (more info on this later) is minimal, and depending on our binning choice might make these new features less suited for our task. The results using 100 bins and using 50 bins can be seen in Figure 6.1. For the remainder of our NN endeavours we will opt to not use these new features, and just remove the features that have missing variables all together.



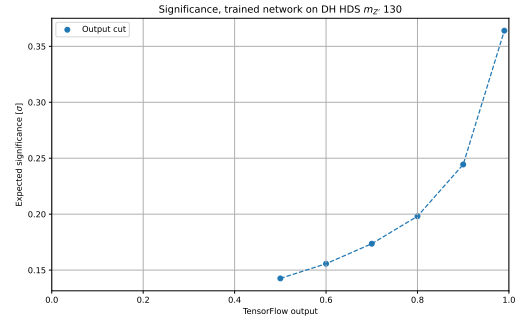
(a) When including new variables and using 100 bins



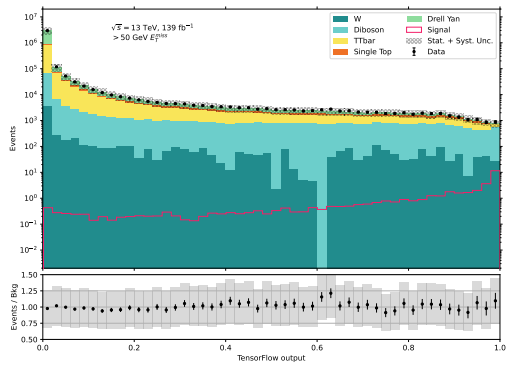
(b) Expected significance of a)



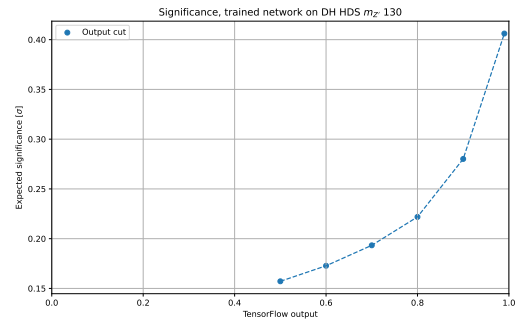
(c) When including new variables and using 50 bins



(d) Expected significance of c)



(e) When excluding new variables and using 50 bins



(f) Expected significance of e)

Figure 6.1: NN prediction when using new features to avoid padding. This is testing a dataset with a Z' DM model.

6.2.2 Normalization of data

Moving onto the second problem, which is not as problematic as the previous, we have the Normalization of data. Since neural networks send a lot of data into multiple neurons and multiple layers using activation functions and other mathematical tools, then it is important to make sure that the signal doesn't die when moving around the network. A fast way for the signal to die off is to not normalize the data and send it through the network, the reason it might die is because we send in numbers which vary significantly to each other, 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 "obviously 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 importance 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 erasing the problem above. Mathematically speaking this is done by

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}} \quad (6.1)$$

Where X is the array containing all features, while X_{min} and X_{max} is the lowest and highest value in 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*

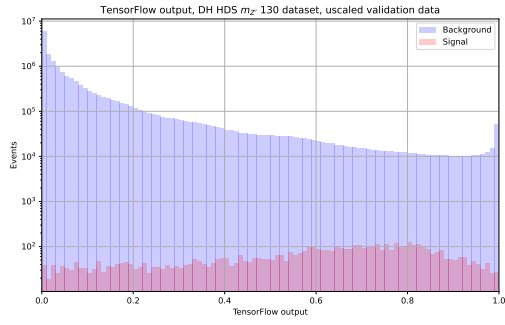
$$X_{norm} = \frac{X - \bar{X}}{\sqrt{\sigma_X^2}} \quad (6.2)$$

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 do this normalization for you, one is called `Batch_normalization` which normalizes the data that enters the network pr. batch, this is usually used in Convolutional NNs as it improves computational speed. And another one is simply `Normalize` which does the same as Eq. (6.2) for the whole training set going in, this is however computationally heavy to use. There is also `Layer_normalization`, where you normalize the activations of the previous layer in a batch *independently*, rather than *across* a batch like `Batch_Normalization`. But of these methods are also based on Z-score normalization.

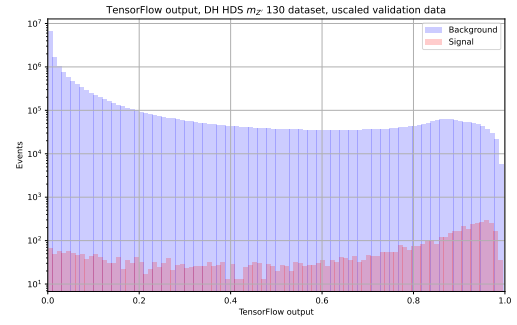
There is a big difference when normalizing data ourselves and using TensorFlow, and that is that TensorFlow will remember how the data was normalized when training and will normalize the testing data in the same way (using the same variables), making testing easier. These different normalization methods have been tested and can be seen in the Figure 6.2. We can see that the best results come from Z-score and Batch normalization. We can see how the expected significance differs on both cases in Figure 6.3.

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 batches 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 normalization for the signal search.

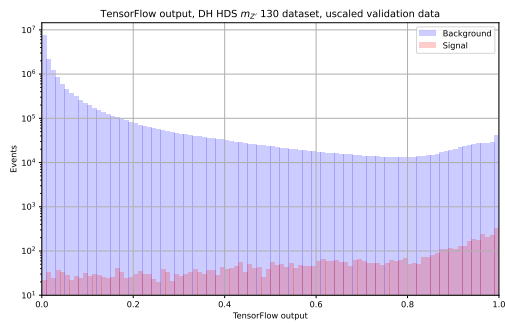
should I even mention this if I will not use NNs further?



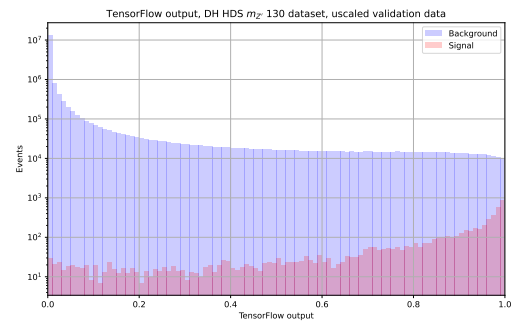
(a) No normalization



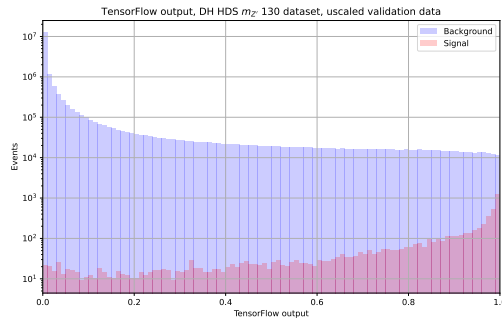
(b) Layer normalization



(c) Min max scaling



(d) Z-score



(e) Batch normalization

Figure 6.2: NN prediction when using different normalization methods. This is testing a dataset with a Z' DM model.

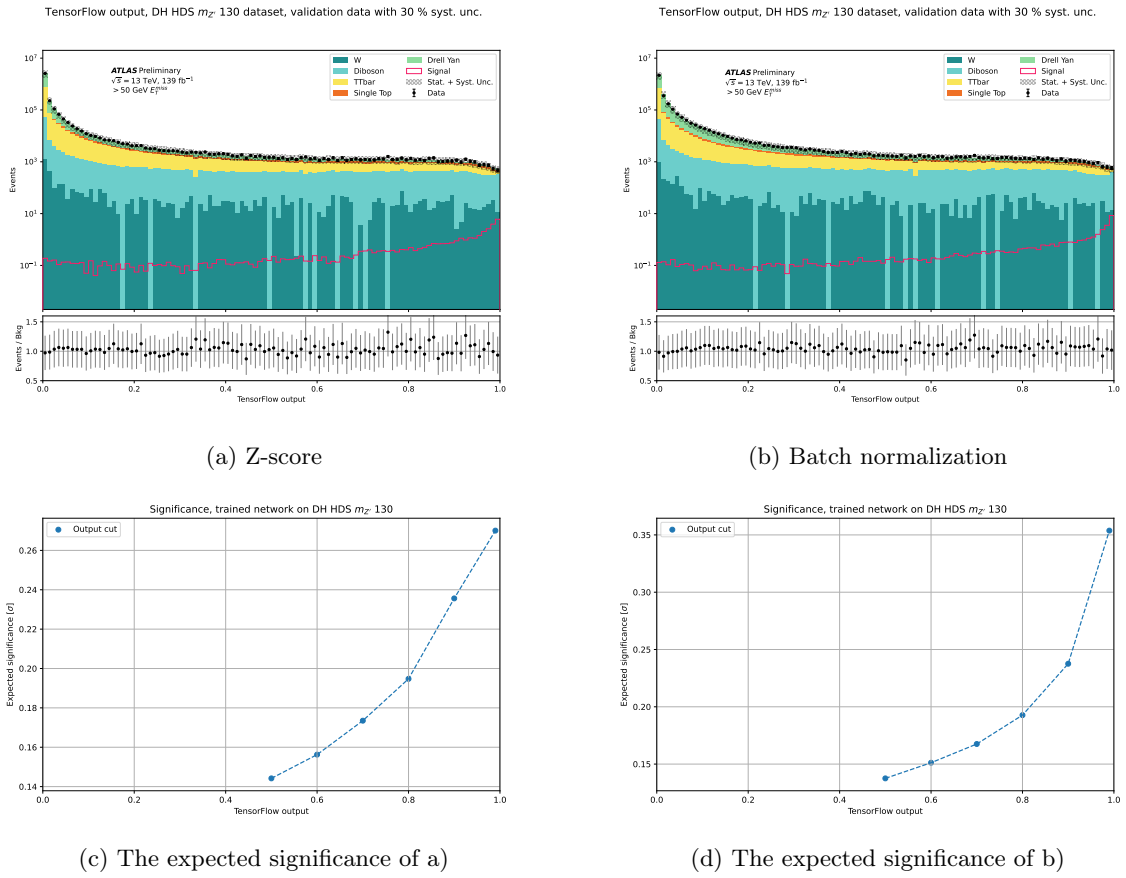


Figure 6.3: Comparison of the best normalization methods. Figure a) and b) show the validation data of both cases, c) and d) show the expected significance of the validation plots when making a cut on the output.

6.2.3 Weights

A big problem that needs to be addressed in this thesis is what we should use as *sample weights* (see Section 4.3.3). 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.

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 Section ... 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 6.4.

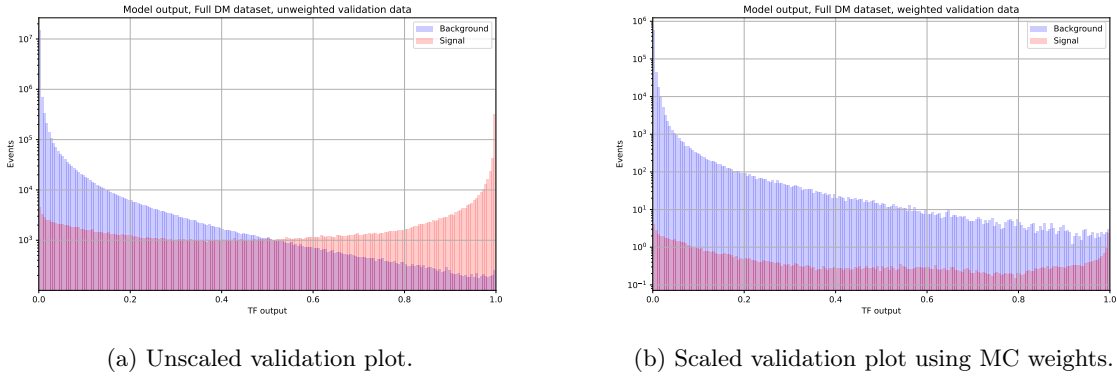


Figure 6.4: Validation plot of unweighted network on the first version of the FULL Z' DM dataset.

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 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 6.2.

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

Table 6.2: 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

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 6.5 and the ROC scores in Figure 6.6.

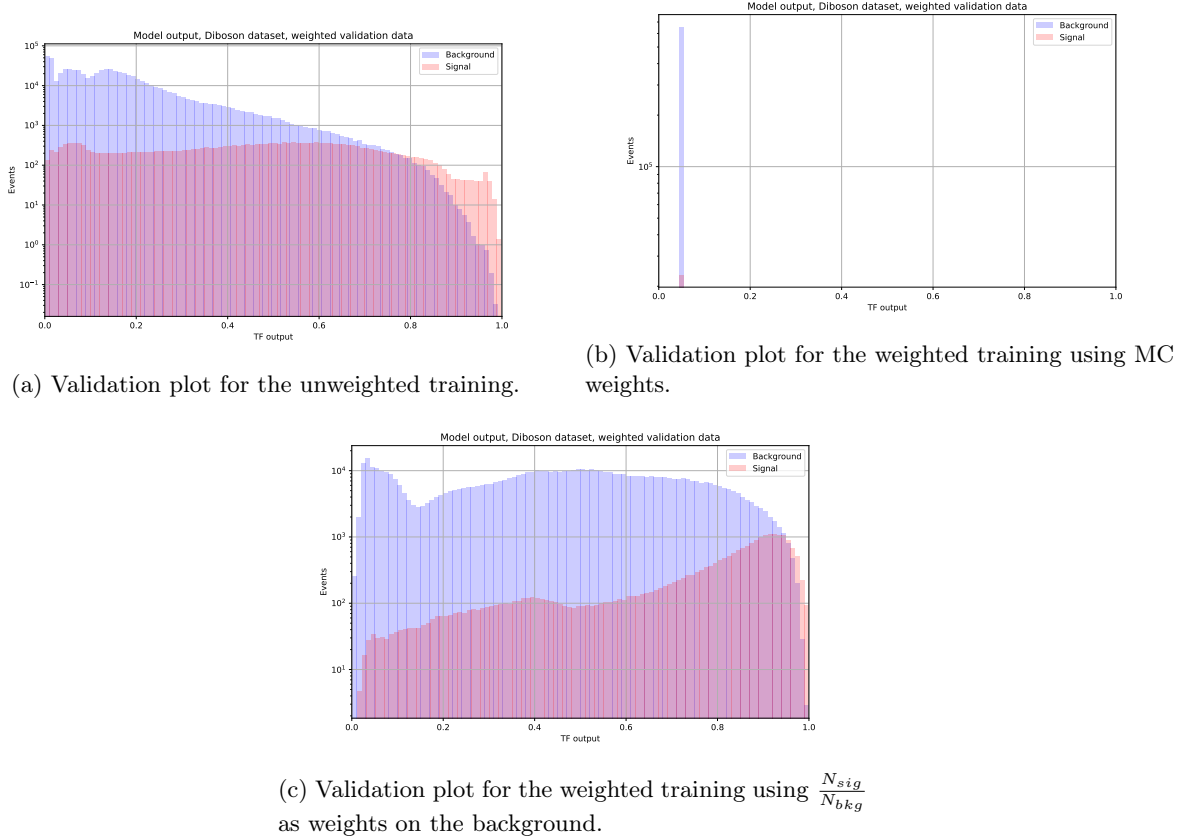
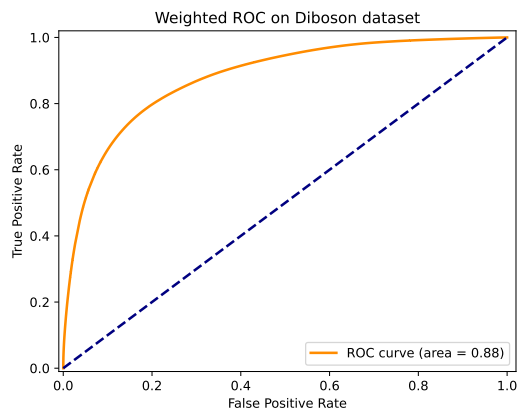


Figure 6.5: Result of the different network training weighting.

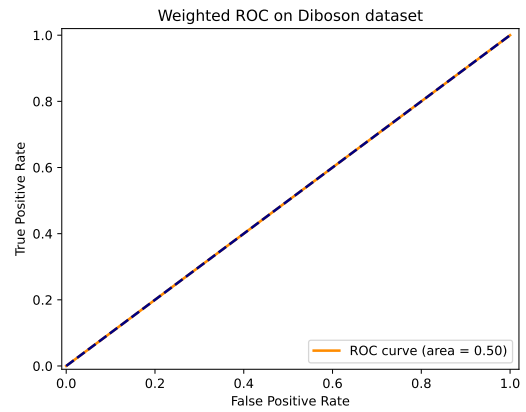
From the results we see that the new weighting method has a poorer result than the unweighted one, and the weighting method that re-weight MC events to expected events just crashes the whole network. The reason for why the latter does not work might be since the network gets punished by guessing "too 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 default 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

¹Not to be confused with AUC!



(a) ROC score for the unweighted training.



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

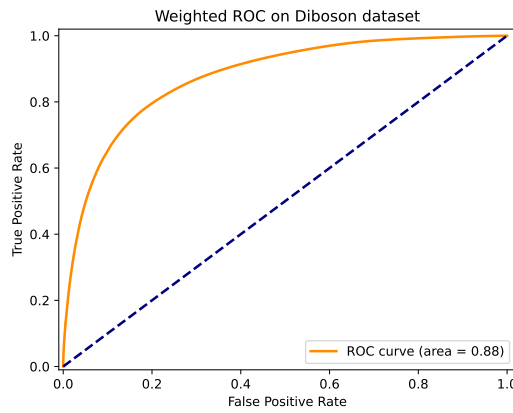
(c) ROC score for the weighted training using $\frac{N_{sig}}{N_{bkg}}$ as weights on the background.

Figure 6.6: Result of the different network training weighting.

6.2.4 Balanced weights

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 whether an event is signal or background using validation plots, this is seen in 6.7.

This whole section is my reasoning for trying to re-weight the MC events to expected events. But is just plain wrong.

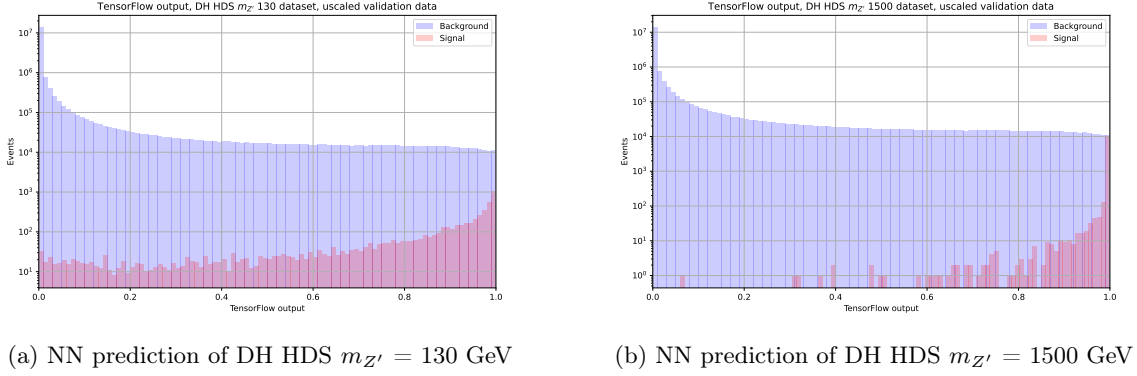


Figure 6.7: NN comparison of trained models in FULL Z' dataset

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 $m_{Z'} = 1500$ GeV than the other, even if this model has a much lower cross section, as seen in the correctly scaled plots below.

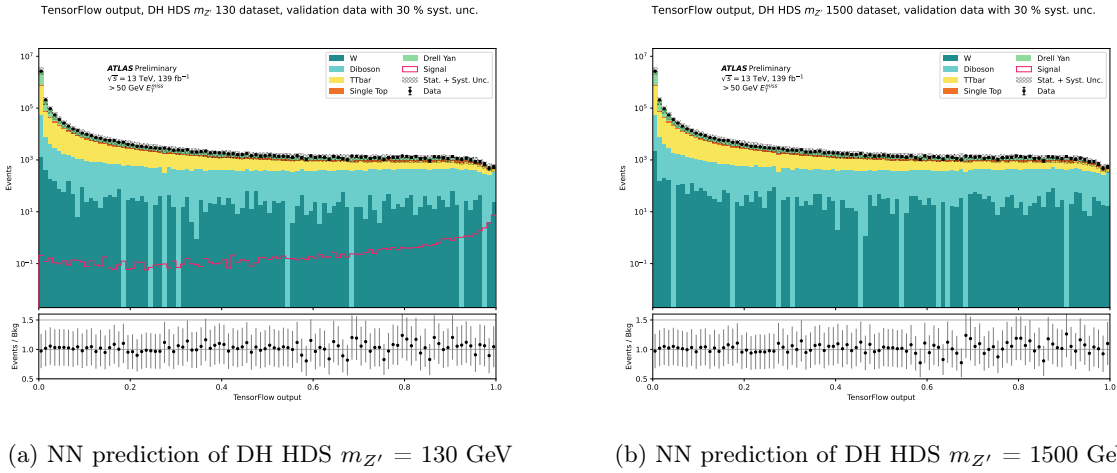


Figure 6.8: Correctly scaled plots.

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`.

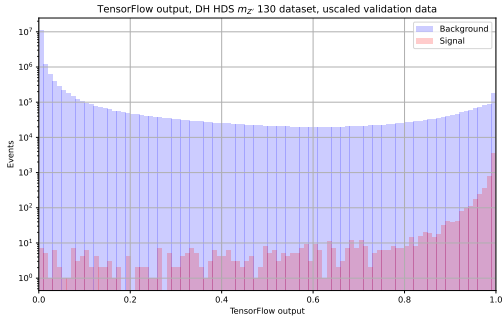
`class_weight` works as a dictionary that weights events that are keys on the dictionary. For our purposes we can make a dictionary where we weight signal and background events differently, this is the same type of scaling that was done in the previous section. `sample_weight` takes in individual weights for every single event that goes into the network, 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 together. Thus we could try to make use of this "balanced weighting" method to see if the network learns the different models better.

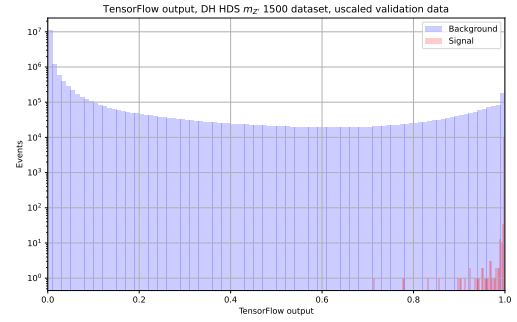
We can see in Table 6.3 how uneven the data (FULL Z') is. To balance the data we have four options. We can either *weigh up the signal* events by the ratio of background over signal. Or *weigh down the background* with the ratio of signal over background. However this ratio might differ a lot when using the MC raw event ratio or the SOW events ratio. I have tested all four possibilities and these can be seen below.

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

Table 6.3: 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.

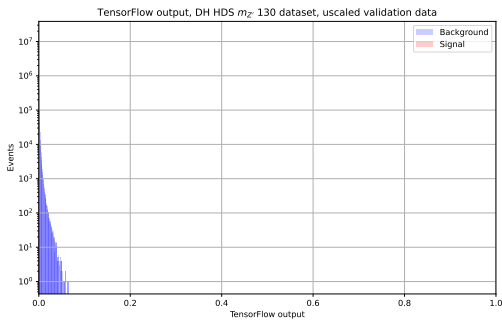


(a) NN prediction of DH HDS $m_{Z'} = 130$ GeV

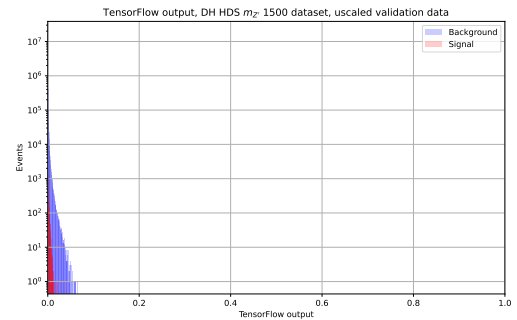


(b) NN prediction of DH HDS $m_{Z'} = 1500$ GeV

Figure 6.9: NN prediction when weighting the 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 6.10: NN prediction when weighting the signal with the raw event ratio.

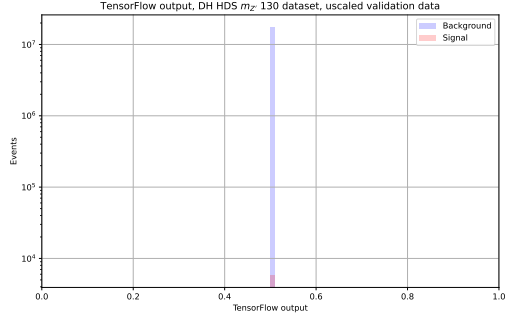
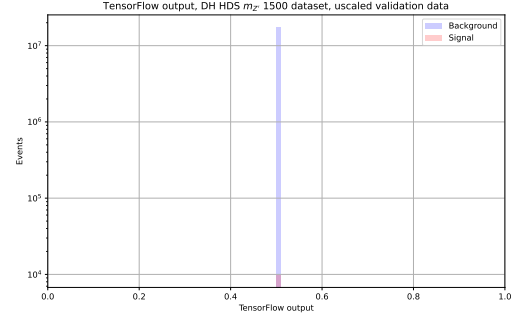
(a) NN prediction of DH HDS $m_{Z'} = 130$ GeV(b) NN prediction of DH HDS $m_{Z'} = 1500$ GeV

Figure 6.11: NN prediction when weighting the background with the SOW ratio.

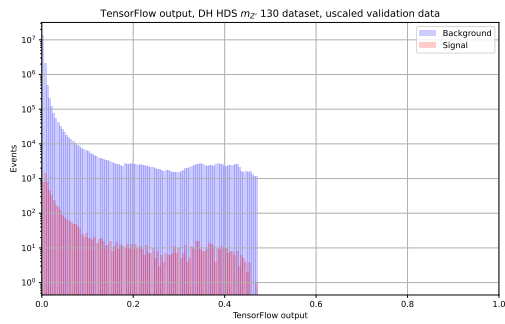
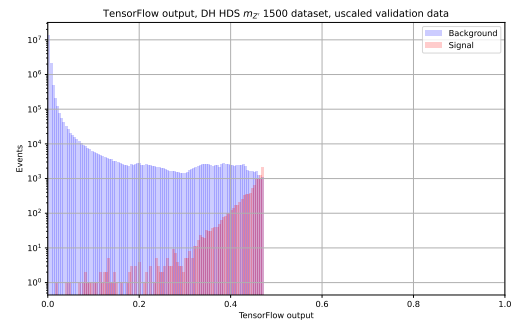
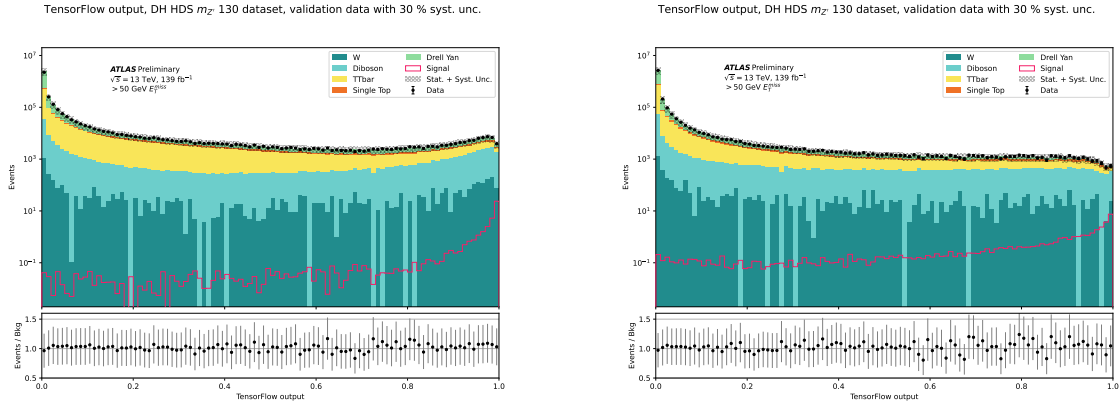
(a) NN prediction of DH HDS $m_{Z'} = 130$ GeV(b) NN prediction of DH HDS $m_{Z'} = 1500$ GeV

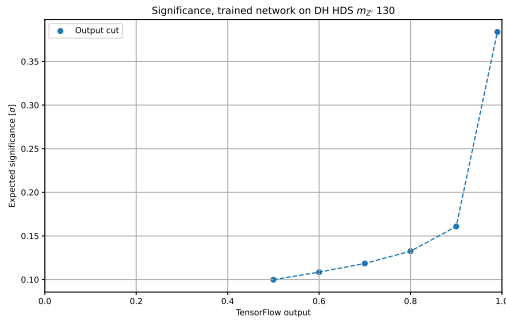
Figure 6.12: NN prediction when weighting the background with the raw event ratio.

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 scale up the MC weights with the ratio between the SOW of the background 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.

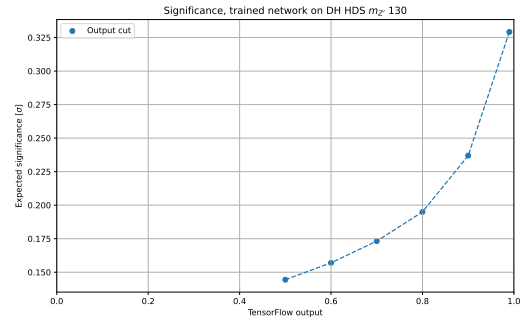


(a) MC weights + scaling signal up with SOW ratio

(b) Only using raw MC events weigh down background



(c) The expected significance of a)



(d) The expected significance of b)

Figure 6.13: 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 because I utilized the same hyperparameters when training both networks, 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 6.14.

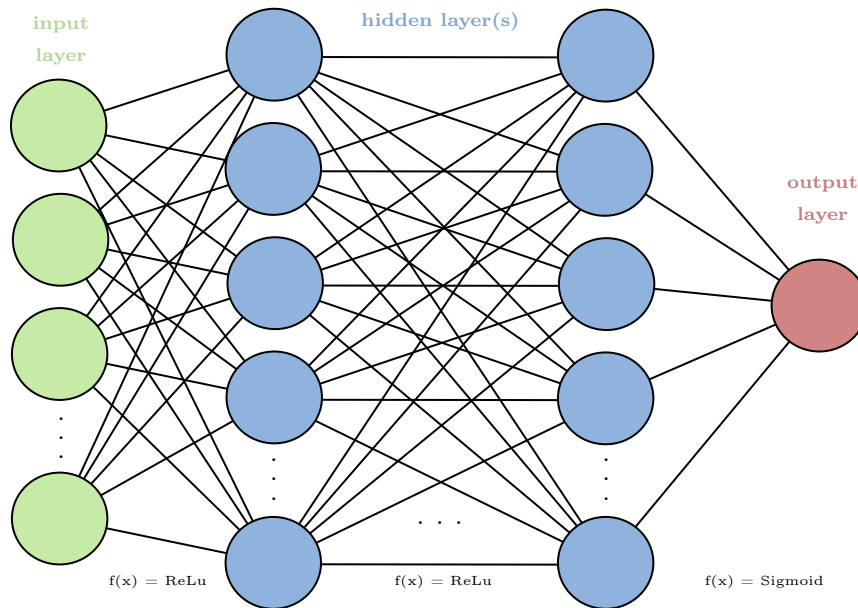


Figure 6.14: Architecture of the NN used on this thesis. The neurons on the hidden layer(s) is a hyperparameter, as well as the number of hidden layers.

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

Algorithm 6.1: Neural network definition using TensorFlow

```

1  import tensorflow as tf
2  from tensorflow.keras import layers
3
4  def Neural_Network(inputsize, n_layers, n_neuron, eta, lamda):
5
6      model=tf.keras.Sequential()
7
8      for i in range(n_layers):
9          if (i==0):
10             model.add(layers.Dense(n_neuron, activation='relu', kernel_regularizer=
11                 tf.keras.regularizers.l2(lamda), input_dim=inputsize))
12          else:
13             model.add(layers.Dense(n_neuron, activation='relu', kernel_regularizer=
14                 tf.keras.regularizers.l2(lamda)))
15
16      model.add(layers.Dense(1,activation='sigmoid'))
17
18      sgd=tf.optimizers.Adam(learning_rate=eta)
19
20      model.compile(loss=tf.losses.BinaryCrossentropy(),
21                  optimizer=sgd,
22                  metrics = [tf.keras.metrics.BinaryAccuracy()])
23      return model
24

```

6.2.6 Grid Search

To get the best performance on our NN, we need to find which hyperparameters 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**, excluding 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. (7.1) just in case there is too few events after the network prediction. The expected significance will also be calculated when making a cut on 0.85 on the network 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.***

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_lambda_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 6.15, while the AUC is in Figure 6.16.

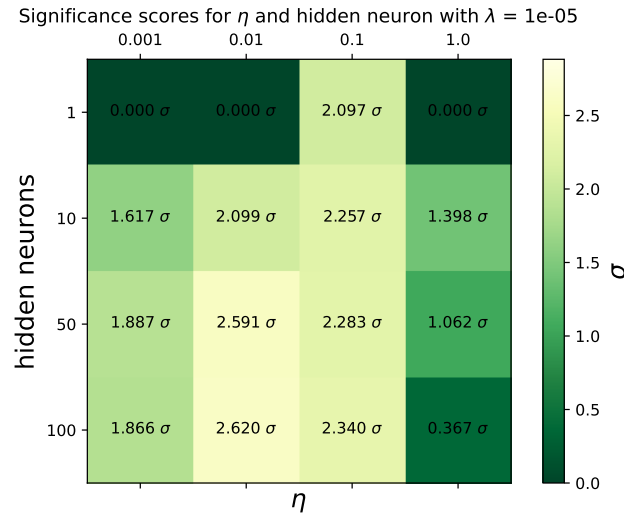
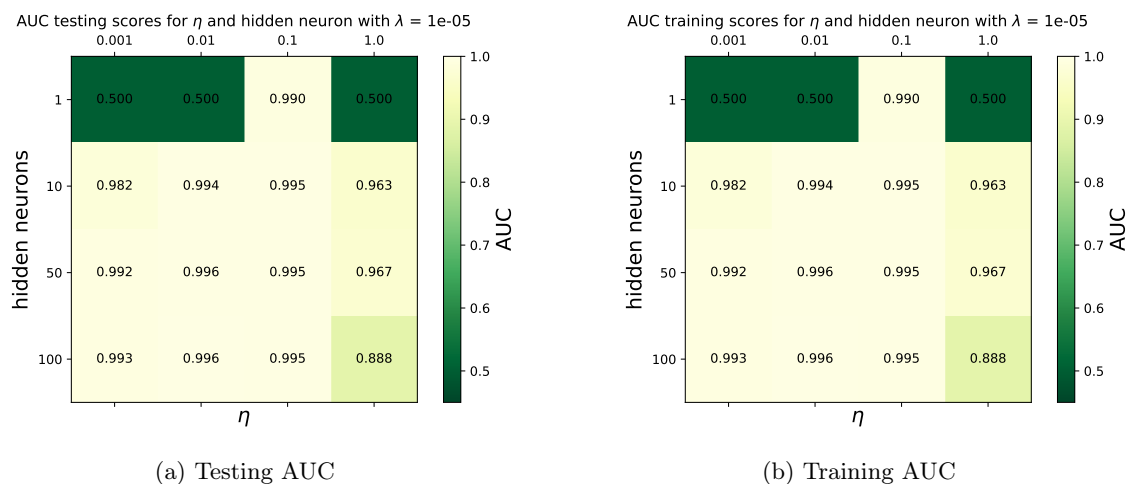
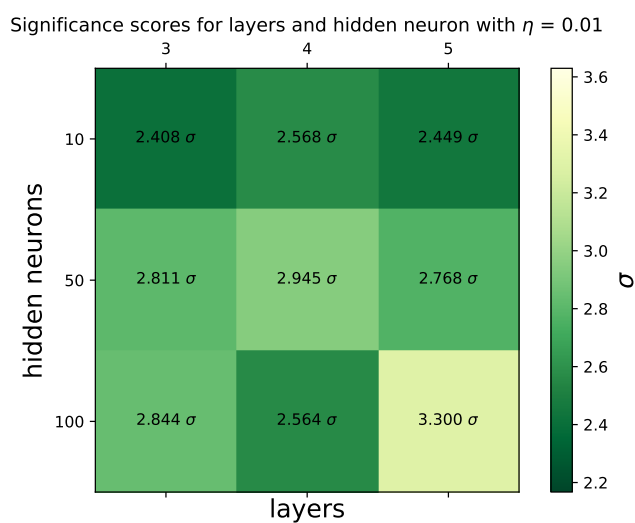
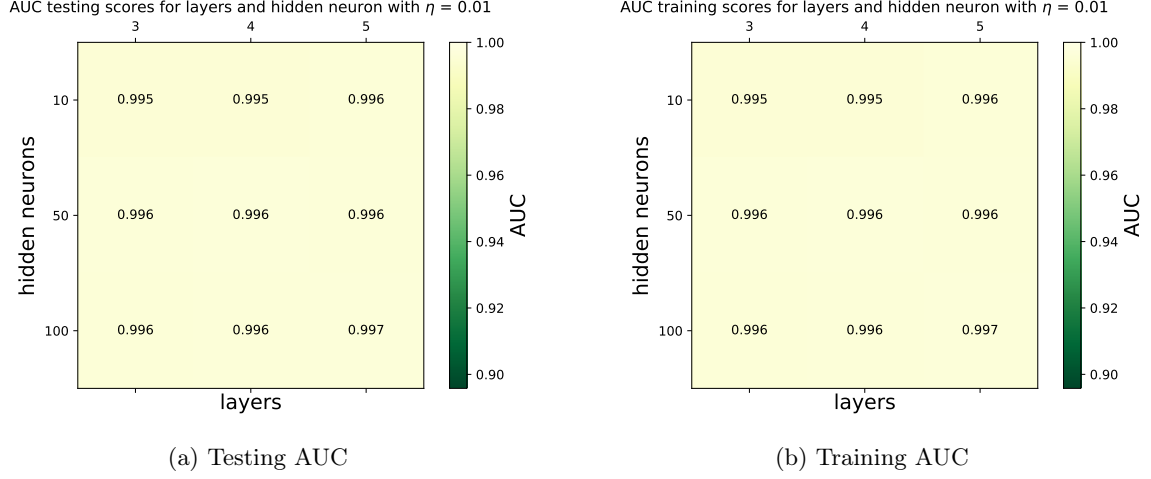


Figure 6.15: Grid search significance with $\lambda = 10^{-5}$ and **n_layers** = 2

Figure 6.16: 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 6.17, while the AUC is in Figure 6.18.

Figure 6.17: Grid search significance with $\lambda = 10^{-5}$ and $\eta = 0.01$

Figure 6.18: 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 6.17, 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. (6.2), and the weighting method explained in section 6.2.3. The results are shown in the figure below.

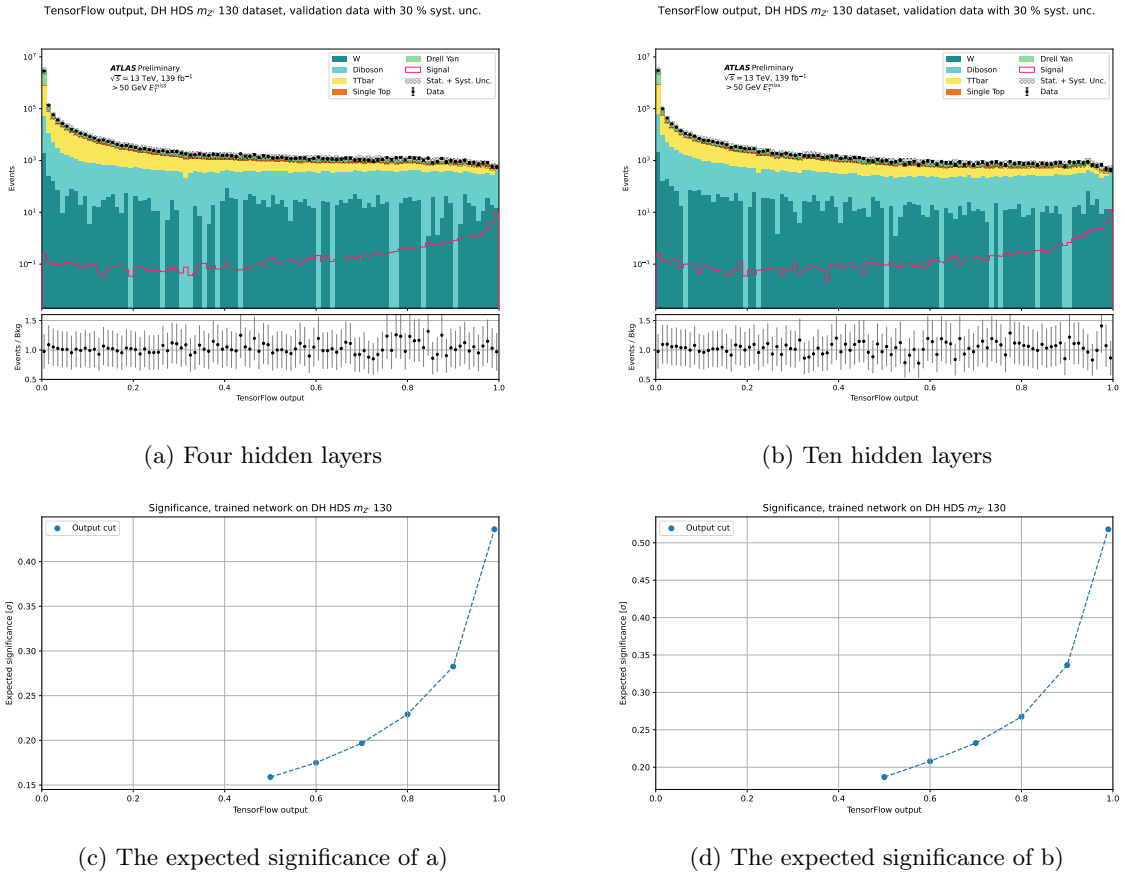


Figure 6.19: Comparison of the network performance when having four and ten hidden layers. Figure a) and b) show the validation data of both cases, c) and d) show the expected significance of the validation plots when making a cut on the output.

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. [11], which could potentially avoid the imbalance problem, but this is proposed as a plausible new research project due to time constrain on this thesis.

6.3 Boosted Decision 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.

remember to mention it

add source/remove this comment?

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.

6.3.1 Weights

cite sherpa?

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

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 6.2.4 completely disappears, 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 account higher order diagrams and needs to add negative weights to "counter" the overcounting of diagrams [15], 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.

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) [16] 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. Mathematically speaking, if we have an array of weights W , we can update this like

$$W \rightarrow |W| \frac{\sum_i W_i}{\sum_i |W_i|} \quad (6.3)$$

such that the weights are at least in the same scale. I have tried all of these options and the results can be seen in Figure 6.20.

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 background. In theory it makes no sense whatsoever 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 strictly speaking we are invoking a semi-supervised 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 independent! 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?

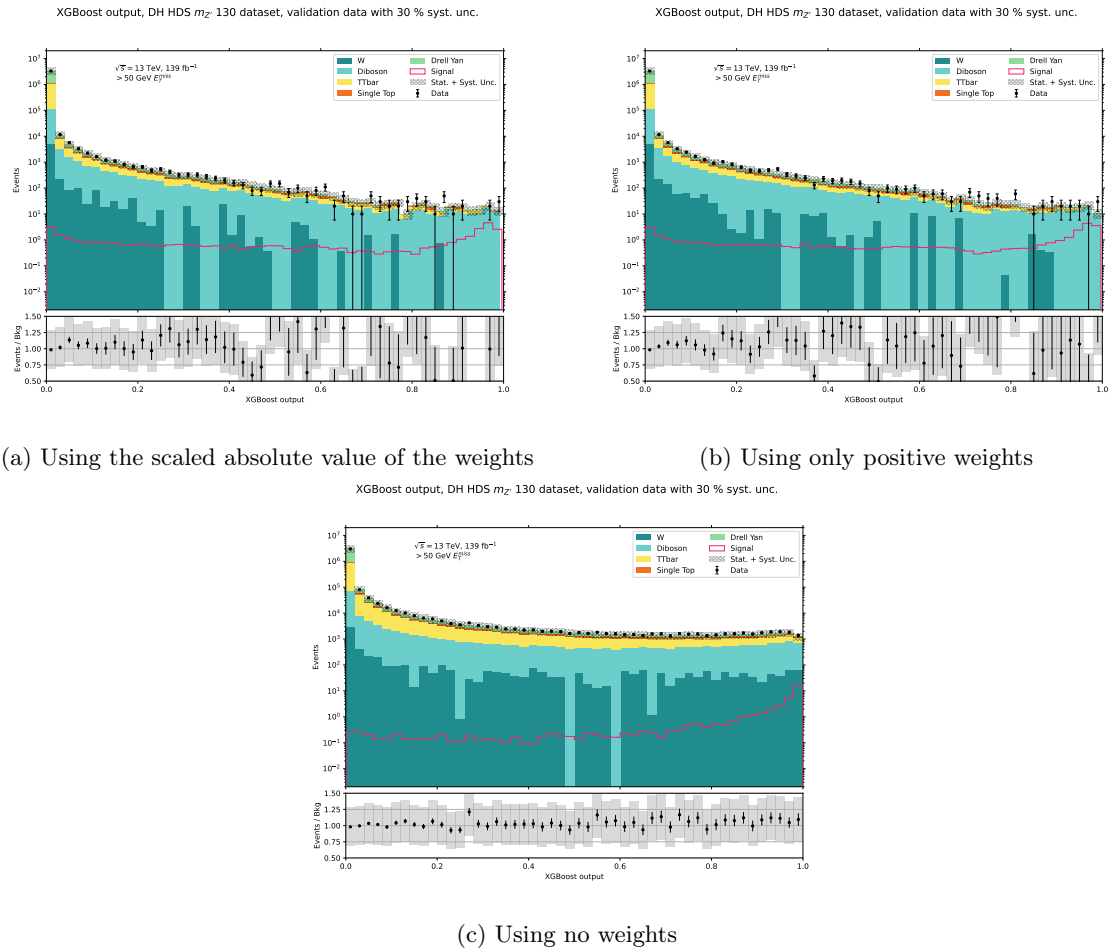


Figure 6.20: Difference when using different weighting methods. All networks were trained using the balancing method explained in Section 6.2.3

6.3.2 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 overtraining

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 tree 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. The results can be shown in Figure 6.21. 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 6.22.

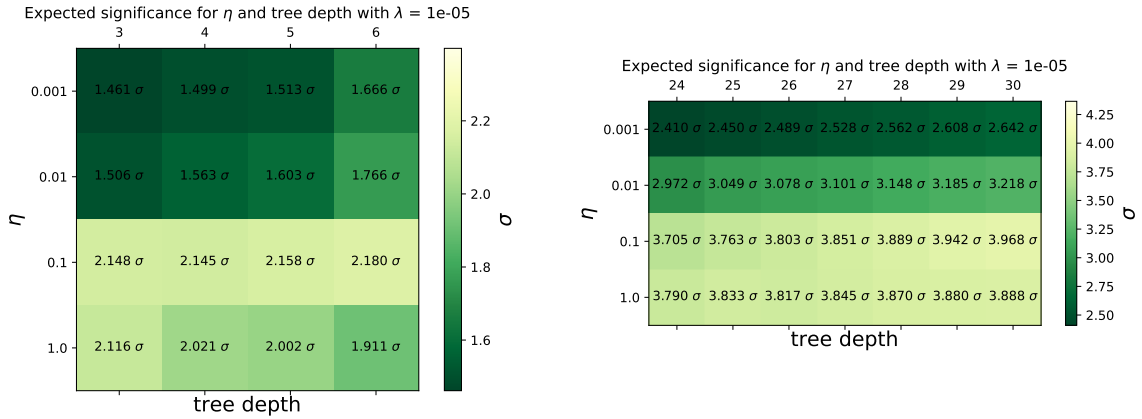
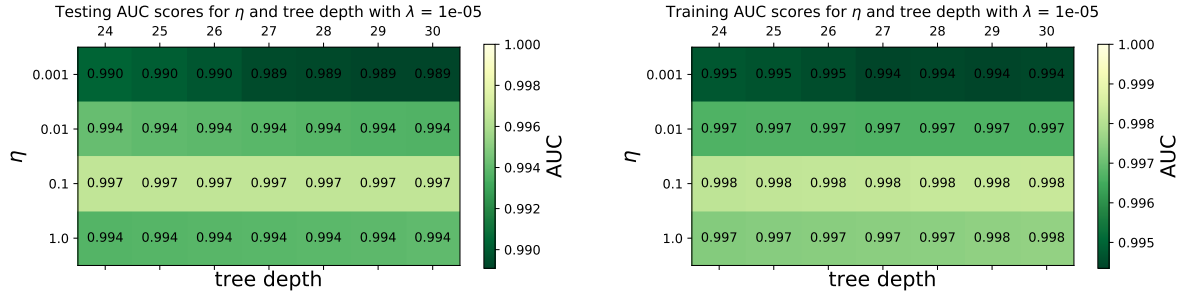


Figure 6.21: Grid search expected significance going to a depth of up to 30

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 exactly one event per 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 6.23 as most important.



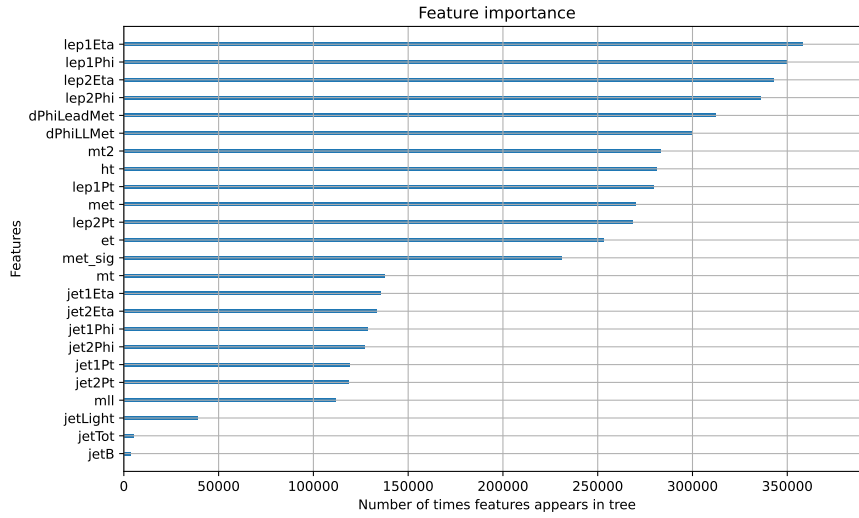
(a) Testing AUC

(b) Training AUC

Figure 6.22: Grid search AUC going to a depth of up to 30

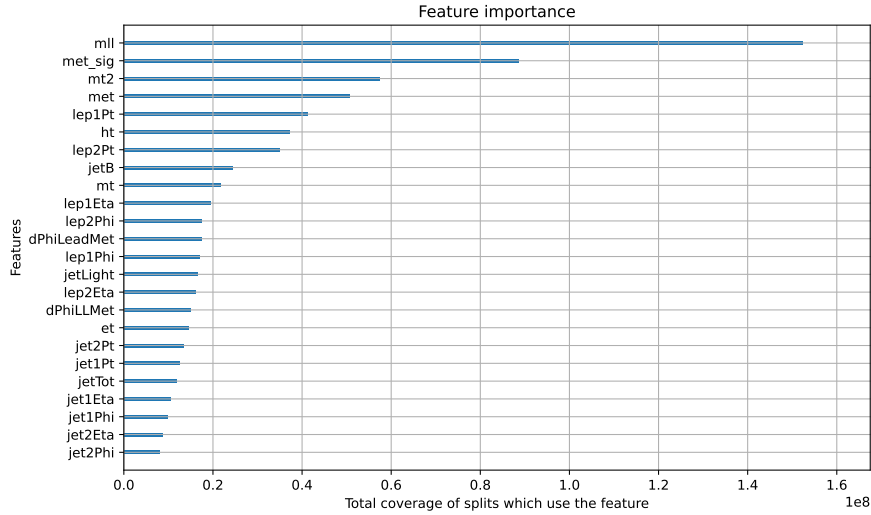
As we can see these features vary a lot depending on which metric we use to evaluate the importance. When using the "coverage" metric, which as stated is defined as the number of samples affected by the split, we get the features we physically expect to be important when trying to single out a DM model. 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.



(a) Using "weight" metric

Coverage is defined as the number of samples affected by the split



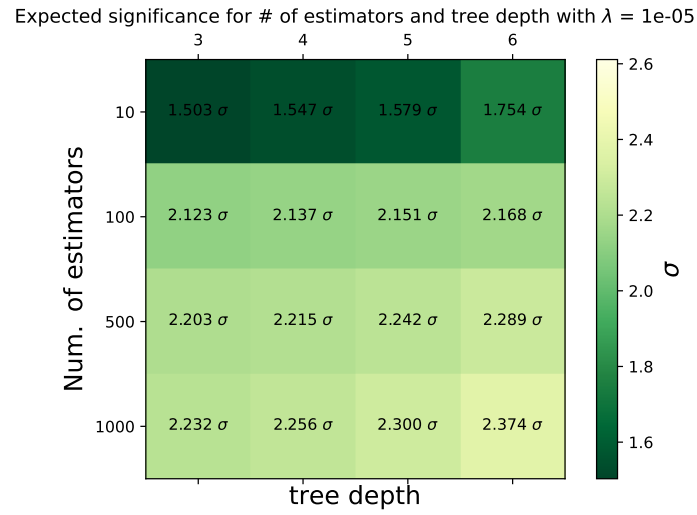
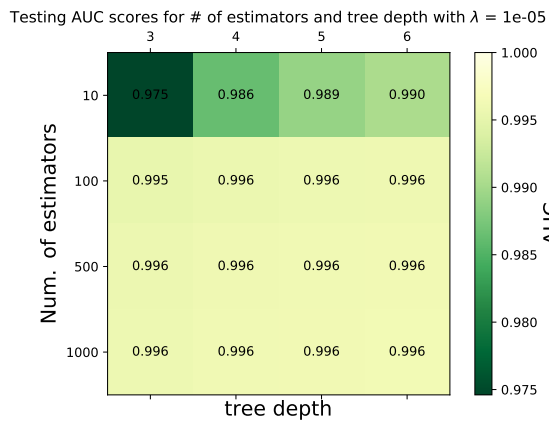
(b) Using "coverage" metric

Figure 6.23: Feature importance of depth 30 network trained on FULL Z' DM data set when testing it on DH HDS $m_{Z'} = 130$ GeV model.

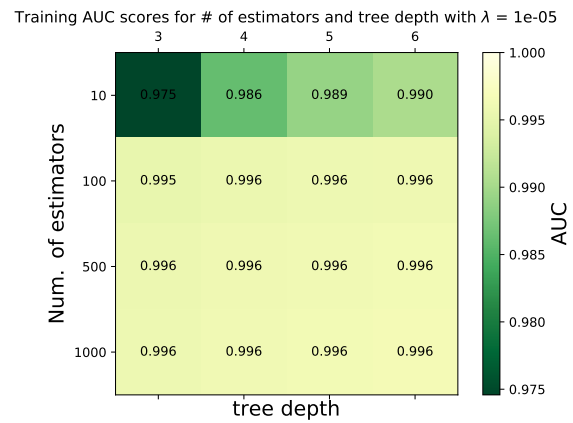
As the previous results are to be taken with a heavy grain of salt, I conducted another grid search. On the second grid search I set the values of $\eta = 0.1$ as the trend showed this giving the best results with less overtraining, and $\lambda = 10^{-5}$. This grid search had `n_estimators` $\in [10, 100, 500, 1000]$ and depth $\in [3, 4, 5, 6]$. The expected significance is shown in Figure 6.24. The testing and training AUC can be seen in Figure 6.25.

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 6.26. 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.

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

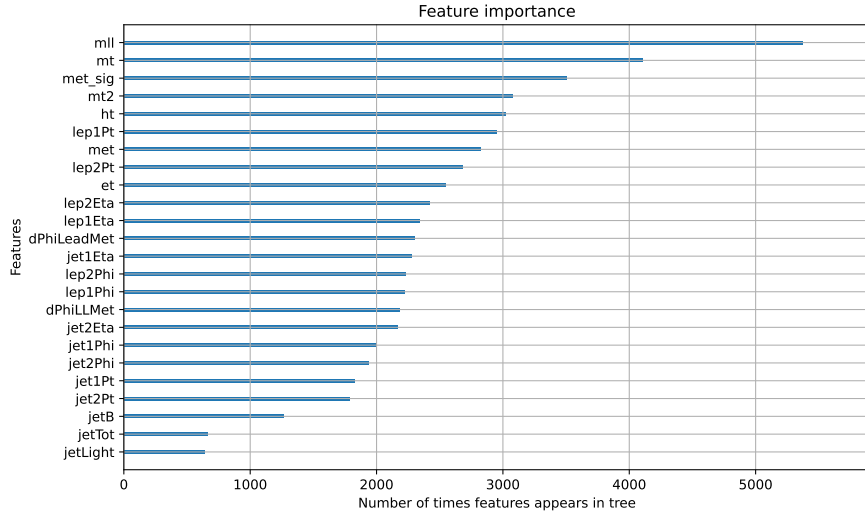
Figure 6.24: Grid search expected significance when setting $\lambda = 10^{-5}$ and $\eta = 0.1$ 

(a) Testing AUC



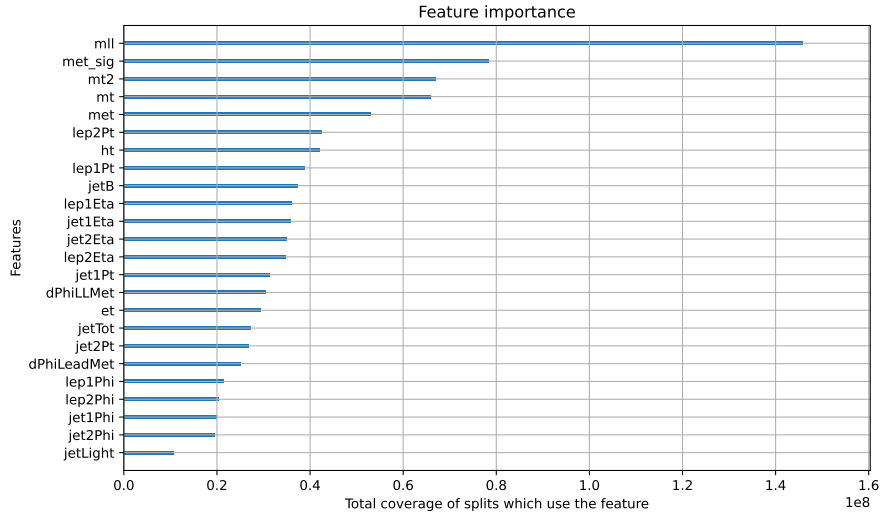
(b) Training AUC

Figure 6.25: Grid search AUC when setting $\lambda = 10^{-5}$ and $\eta = 0.1$



(a) Using "weight" metric

Coverage is defined as the number of samples affected by the split



(b) Using "coverage" metric

Figure 6.26: 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.

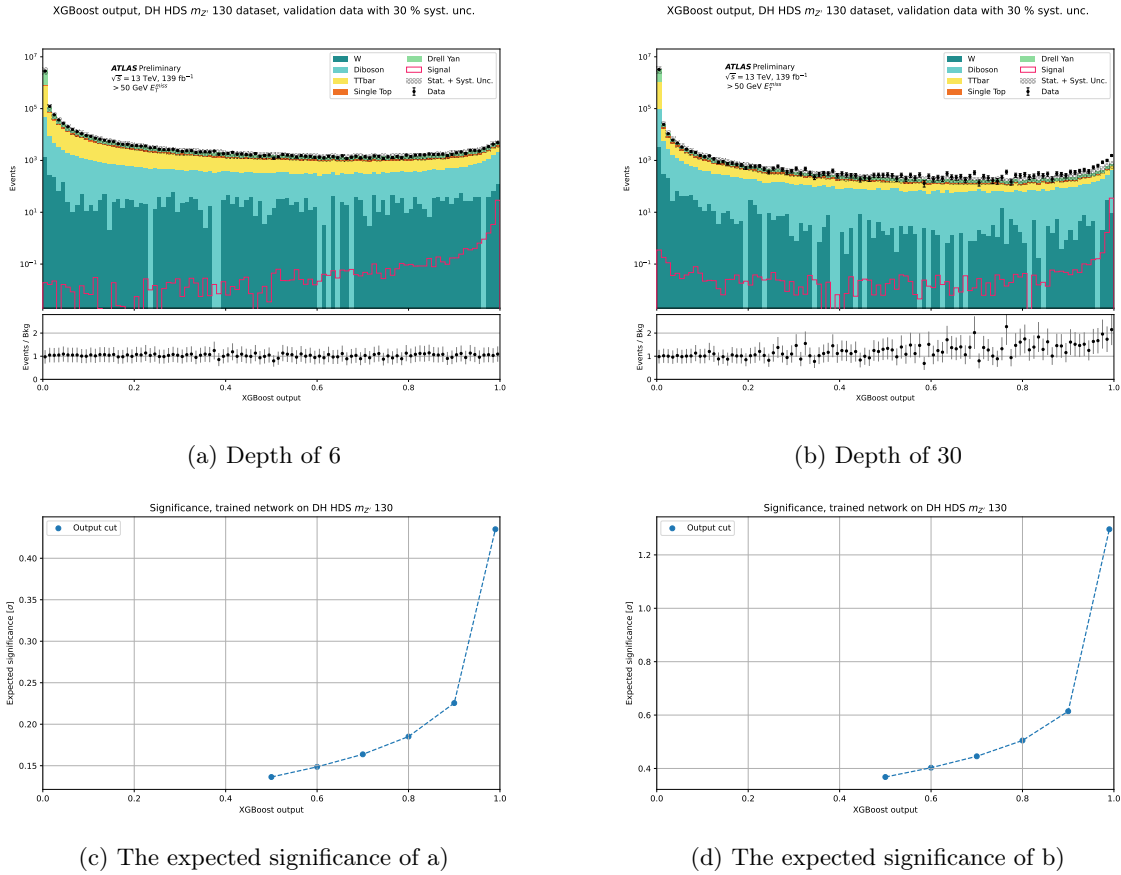


Figure 6.27: Comparison of the network performance when having a depth of 6 and 30. Figure a) and b) show the validation data of both cases, c) and d) show the expected significance of the validation plots when making a cut on the output.

The difference is extreme, when looking at the monster of depth 30 we can get an expected significance of 1.2σ (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 points 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.

Part III

Results

Chapter 7

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 calculate the expected significance. The three models I chose to test are all High Dark Sector models with $m_{Z'} = 130\text{GeV}$. 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 7.1.

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

Table 7.1: Table showcasing the cuts used when doing the cut and count method.

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

$$Z = \sqrt{2 \left[(s+b) \ln \left(1 + \frac{s}{b} \right) - s \right]} \quad (7.1)$$

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 7.2 for the electron channel and Table 7.3 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 $m_{ll} < 150 \text{ GeV}$. This makes sense since the models in question all have a $m_{Z'} = 130\text{GeV}$. 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.

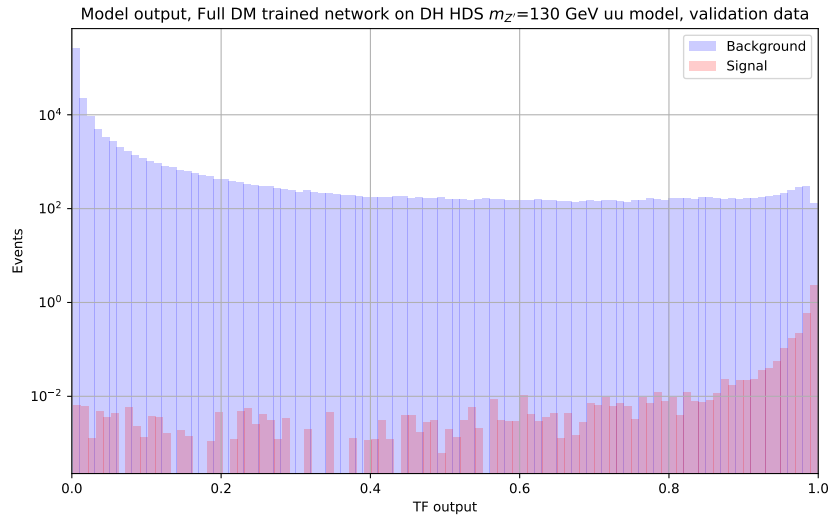
	LV	DH	EFT	Background
Events before cuts	15	20	0	1,256,624
Events after cuts	4	6	0	117
Expected significance [σ]	0.4	0.6	0	

Table 7.2: Table showcasing the result of the cut and count method for the electron channel.

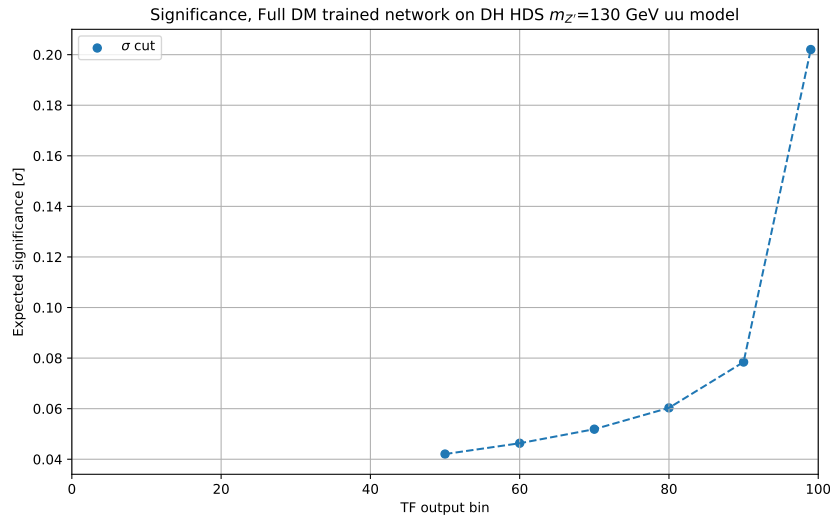
If we were to compare these results with what our NN and BDT that trained on the full dataset we see that we can calculate the expected significance in different locations for the validation plots. Testing on the networks that trained using the data scientist method on the full DM dataset we get the results shown in Figure 7.1 for XGBoost and Figure 7.2 for the Neural Network.

	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	

Table 7.3: Table showcasing the result of the cut and count method for the muon channel.

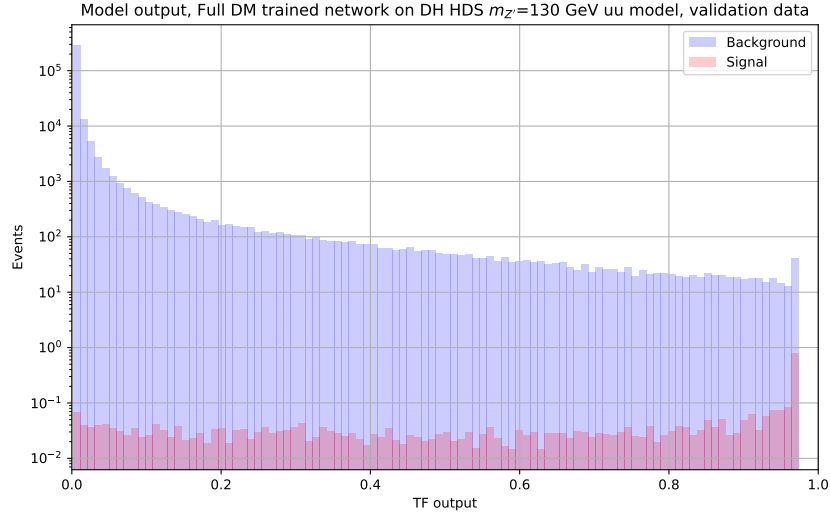


(a) Validation plot.

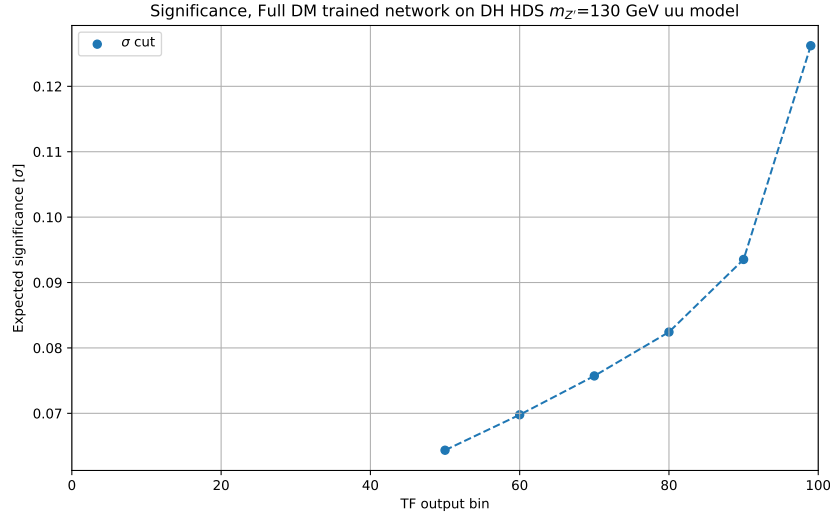


(b) Expected significance when looking at bins and forth.

Figure 7.1: 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 7.2: 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.

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

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

Table 7.5: 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

in spacetime. balance re-weighted bkg with raw MC signal. chatgpt was right about sample weights (its summed over though, keep that in mind)

Bibliography

1. Thomson M. Modern Particle Physics. Cambridge University Press, 2013
2. Jackson JD. "Kinematics". Particle Data Group 2008. Available from: <https://pdg.lbl.gov/2008/reviews/kinemarpp.pdf>
3. 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>
4. 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](https://doi.org/10.1088/0954-3899/29/10/304). Available from: <https://doi.org/10.1088/0954-3899/29/10/304>
5. 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](https://doi.org/10.1088/1748-0221/3/08/S08003). Available from: <https://cds.cern.ch/record/1129811>
6. Pequenaio J and Schaffner P. How ATLAS detects particles: diagram of particle paths in the detector. 2013. Available from: <https://cds.cern.ch/record/1505342>
7. Guevara R. The HiggsML and a simple Quantum-enhanced Machine Learning algorithm. 2022. Available from: <https://github.com/rubenguevara/QuantumMachineLearning>
8. 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](https://doi.org/10.1016/j.physletb.2012.08.020). Available from: <https://doi.org/10.1016/j.physletb.2012.08.020>
9. 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](https://doi.org/10.1007/jhep09(2014)037). Available from: [https://doi.org/10.1007/jhep09\(2014\)037](https://doi.org/10.1007/jhep09(2014)037)
10. 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>
11. 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](https://doi.org/10.1140/epjc/s10052-016-4099-4). Available from: <https://doi.org/10.1140/epjc/s10052-016-4099-4>
12. 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](https://doi.org/10.1038/ncomms5308). Available from: <https://doi.org/10.1038/ncomms5308>
13. 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>
14. Friedman JH. Greedy function approximation: A gradient boosting machine. The Annals of Statistics 2001; 29:1189–232. DOI: [10.1214/aos/1013203451](https://doi.org/10.1214/aos/1013203451). Available from: <https://doi.org/10.1214/aos/1013203451>
15. Danziger K, Höche S, and Siegert F. Reducing negative weights in Monte Carlo event generation with Sherpa. 2021. DOI: [10.48550/ARXIV.2110.15211](https://arxiv.org/abs/2110.15211). Available from: <https://arxiv.org/abs/2110.15211>

16. 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>