Optimisation of an Adversarial Neural Network in the tW Dilepton Channel at ATLAS

Nicolas Erasmus Boeing

Masterarbeit in Physik
angefertigt im Physikalischen Institut

vorgelegt der
Mathematisch-Naturwissenschaftlichen Fakultät
der
Rheinischen Friedrich-Wilhelms-Universität
Bonn

Nov 2020

I hereby declare that this thesis was formulated by myself and that no sources or tools other than those cited were used.

Bonn, 17.11.2020 .....
Date

.....
Signature

1. Gutachter: Prof. Dr. Ian C. Brock
2. Gutachter: Prof. Dr. Florian Bernlochner

Acknowledgements

I am very grateful to Prof. Ian C. Brock, for giving me the opportunity of working on this topic and always being supportive. I'd like to thank Christian for always being helpful, even when things didn't go as planned. A special thanks also go out to each and every member of the research group for the interesting chats, valuable advice and, when we still could, the fun board game evenings.

I can't thank my parents enough for always supporting me on my studies.

Lastly, I am indebted to Frank for helping me keep my sanity in the rather uncertain times that most of this work was done in.

Contents

CHAPTER 1

Introduction

Humans have been striving to understand the world around them for thousands of years. The first example of physics, the study of the matter and energy that makes up everything around us, stem from ancient Greece around 2 400 years ago, when the philosopher Thales of Miletus proclaimed that every event had natural causes and could be explained [1]. Since then, physical sciences have been progressing steadily, going through many breakthroughs such as the invention of classical mechanics by Isaac Newton [2] or the formulation of the electromagnetic theory by James Clerk Maxwell [3].

The latest great breakthrough has been the development of quantum mechanics and subsequently the Standard Model of particle physics, describing the fundamental building blocks of matter and how they interact at the smallest scales and highest energies [4]. This branch of physics has remained a topic of active fundamental research ever since, with steadily improving tools and techniques.

In order to study new particles and their interactions at these scales they have to be accelerated to high energies, collided and the remnants of these interactions measured. This is done using large colliders that smash particles together at near light speeds, with the interaction products being measured by sophisticated detectors that can grow to the size of large buildings.

This research generates gigantic amounts of data that has to be filtered and analysed with ever more complex data analysis techniques. A recent, major advancement has been the development of machine learning methods such as neural networks with their unrivaled ability to learn even the most subtle patterns in complex data. This work will focus on adversarial neural networks, a recently invented method that promises to reduce impact of systematic uncertainties, unknown variations in data that are not statistical in nature.

First, some necessary experimental and theoretical basics are introduced, such as the Standard Model of particle physics, the Large Hadron Collider and ATLAS detector, and the tW dilepton decay channel that is the focus of this work. Next, the concept of machine learning is described and the methods of neural networks and adversarial neural networks introduced. Then, the particular network setup used in this thesis is outlined and the optimisation process detailed, followed by describing how its runtime could be heavily improved. Lastly, the results of using this adversarial network are presented and its viability for physics analyses discussed, as well as some potential future improvements outlined.

CHAPTER 2

Experimental and Theoretical Background

In this chapter some necessary experimental and theoretical basics are introduced. First, Section 2.1 briefly introduces the Standard Model of particle physics with its elementary particles and forces. This work is conducted using simulations based on the Large Hadron Collider (LHC) and ATLAS detector, so both machines are described in Section 2.2. Section 2.3 will go into more detail about the particular physics process studied in this analysis. Lastly, a quick introduction into how these detector events are simulated is given in Section 2.4.

2.1 The Standard Model of particle physics

Basis to all of modern particle physics is the so-called *Standard Model*, which was developed in the 1970s as a result of the rapid advancements in theoretical particle physics in the decades before [4]. It is a gauge theory describing all known fundamental particles and the interactions between them. While the Standard Model has limitations, such as not being able to explain phenomena like gravity and neutrino oscillations, it is in itself consistent and has been able to provide many experimental predictions since its conception. An overview of the Standard Model is shown in Figure 2.1.

2.1.1 Matter particles

There are a total of 12 matter particles and their respective antiparticles, particles with the same mass but reversed quantum numbers, in the Standard Model. They consist of 6 *quarks* and 6 *leptons* which can both be additionally grouped into three generations of pairs. Particles of higher generation retain the same quantum numbers but have higher masses, making them unstable. Due to this, all common matter in the universe is made up of particles in the first generation. These fundamental particles can interact using the electromagnetic, strong and weak forces, depending on their electric, colour and weak charges. While they can also interact with gravity, it is many orders of magnitude weaker than even the weak force at subatomic ranges, so it can be neglected.

Quarks are the only matter particles that carry a colour charge, enabling them to interact using the strong force. They are grouped into up-type quarks with a charge of 2/3, consisting of up, charm and top quark, and down-type quarks that have a charge of $-1/3$ and consist of down, strange and bottom

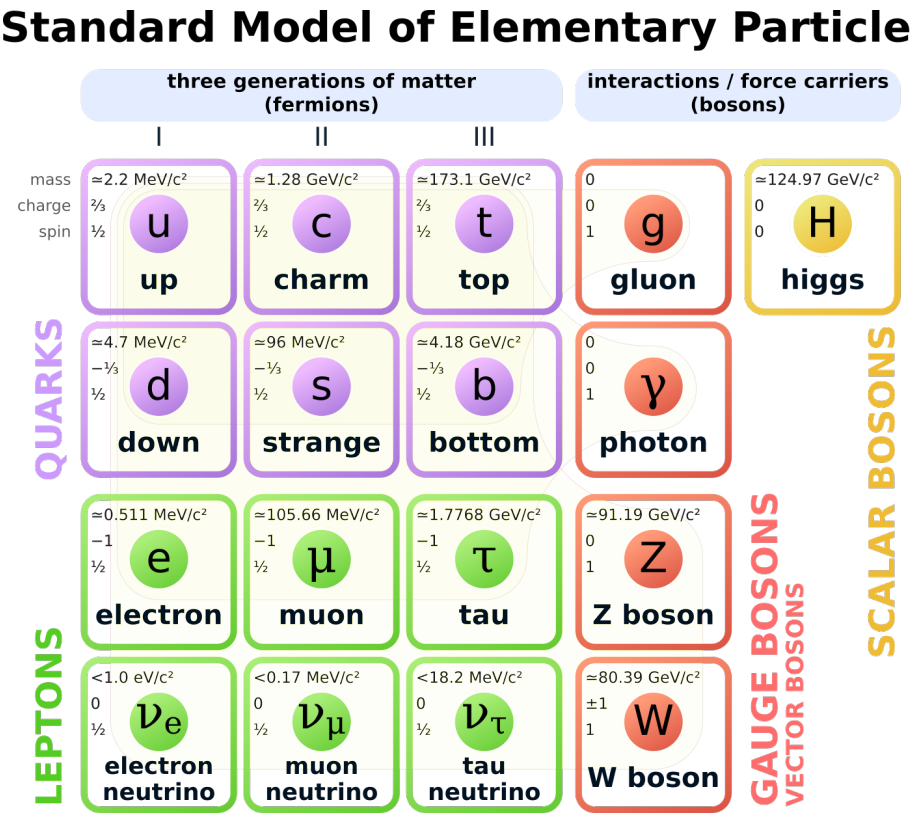


Figure 2.1: Overview of the Standard Model of particle physics [5]

quarks [6]. Quarks can (with the exception of the top quark) form bound states, called hadrons. The only stable hadrons are the proton, made up of two up and one down quark, and neutron, comprising two down and one up quark. Together they make up the overwhelming amount of mass in ordinary matter.

Leptons consist of three particles, each with a corresponding neutrino. The electron, muon and tau lepton have an electric charge of -1, allowing them to interact electromagnetically. The electron, muon and tau neutrinos don't have electric or colour charge, so they can only interact weakly. This means that they can pass through large amounts of matter without interacting, making them very difficult to detect.

2.1.2 Forces and gauge bosons

There are a total of four fundamental forces, three of which are described by the Standard Model. They are mediated by exchange of characteristic *bosons*, particles with integer spin.

The strong force, described by the $SU(3)$ symmetry group, is responsible for the formation of bound states between quarks and is much stronger than other forces at close ranges. It is mediated by 8 massless flavoured gluons, which only couple to particles that have a colour charge. Because they themselves have a colour charge, gluons can self-interact, giving rise to *colour confinement*. It prevents quarks and gluons from existing freely, instead clumping into colour-neutral hadrons.

The most common force in everyday life is the electromagnetic force. At a range of 1 fm it is

approximately 137 times weaker than the strong force. However because it can act at long ranges, it is the cause of nearly all phenomena outside of nuclei. Most importantly, it allows electrons to bind with nuclei, forming atoms and molecules. The mediator of the electromagnetic force is the neutral photon, coupling to all particles with an electric charge.

The weak force plays an important role in nuclear β-decay and enables quarks to change flavour. It is 10^{-8} times weaker than the strong force and is mediated by the neutral Z^0 boson and charged W^+ and W^- bosons. Both weak bosons are massive with a mass of 91 MeV/c^2 and 80 MeV/c^2 respectively [7] and have a spin of 1. At high energies electromagnetic and weak forces can be described as a single electroweak interaction.

The Standard Model currently does not explain gravity and is incompatible Einstein's theory of General Relativity, the most successful theory of gravity. However, at a range of 1 fm, gravity is 10^{-37} weaker than the strong force and as such can be disregarded.

2.1.3 The Higgs boson

In the Standard Model, the Higgs field gives elementary particles their mass due to spontaneous symmetry breaking. The Higgs boson is an excitation of the Higgs field with spin 0 and a mass of 125 GeV [7]. It was first observed in 2012 at the LHC, almost 50 years after being theorised by Peter Higgs [8].

2.2 The Large Hadron Collider and the ATLAS detector

As mentioned previously, only particles of the first generation, namely up and down quarks (bound in protons and neutrons) as well as the electron, can be readily observed in everyday life. To create any other particles and study them, colliders are needed to accelerate particles to high speeds and let them interact. Additionally, as some processes are extremely rare, a very high rate of collisions is needed to precisely measure them. In order to compare the data gathered to the Standard Model predictions, events are simulated using Monte Carlo methods.

This thesis works with simulations of data taken by the ATLAS detector at the Large Hadron Collider (LHC).

2.2.1 The Large Hadron Collider

The Large Hadron Collider is a circular collider with a circumference of 27 km, which makes it the largest and most powerful collider in the world [9]. It is located in a tunnel 100 metres underground, at the headquarters of the European Organization for Nuclear Research (CERN) on the Franco-Swiss border near Geneva, Switzerland. At the LHC, protons can be collided with a center-of-mass energy of 13 TeV and the interaction products measured at one of several detectors.

An overview of the LHC accelerator complex can be seen in 2.2. Protons are inserted in bunches of about 10^{11} protons. After receiving some initial energy in LINAC 2 and BOOSTER, they are further boosted by the Proton Synchrotron (PS) and Super Proton Synchrotron (SPS) before being inserted into the LHC ring at an energy of 450 GeV. There they begin travelling in opposite directions through the two parallel beam pipes and continue being accelerated to 6.5 TeV. Superconductive dipole magnets are used to keep the protons in their circular orbit while quadrupole magnets focus the beam. At the

four interaction points, where the proton bunches are collided, the main LHC experiments are located. ATLAS and CMS are general-purpose detectors, designed to cover a large range of possible decay products [10, 11]. ALICE is a heavy-ion detector specialised on studying the quark-gluon plasma [12], while the LHCb experiment is dedicated to precision measurements of CP violation and searching for rare B meson decays [13].

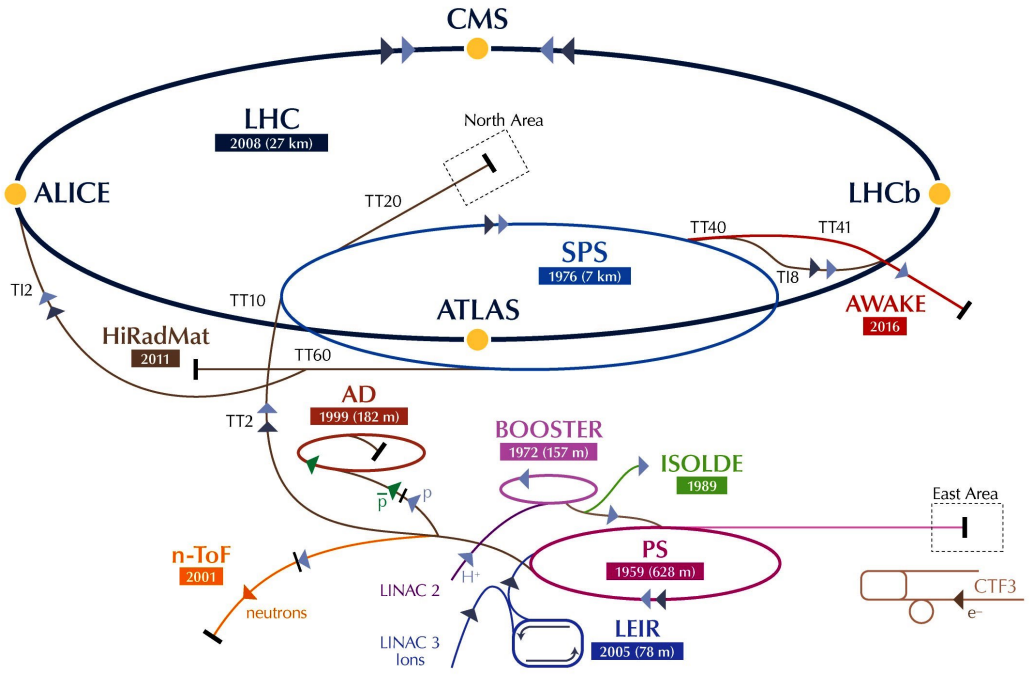


Figure 2.2: Overview of the LHC complex [14]

2.2.2 The ATLAS detector

The ATLAS detector (**A** **T**oroidal **L**HC **A**pparatu**S**) is one of the general purpose detectors at the LHC, designed to detect a large amount of physics processes, including searches for physics beyond the Standard Model. It is shaped in the form of a large cylinder around the beam pipe and at 46 m long and 26 m of diameter is the largest detector at the LHC [15]. It weighs about 7 000 t and has an estimated material cost of 500 million Euro.

ATLAS consists of three major parts: the inner detector, the calorimeters and the muon spectrometer. A massive magnet system surrounds the inner detector and the muon spectrometer. An overview of ATLAS' systems is shown in Figure 2.3.

Coordinate system

The coordinate system used in ATLAS is right-handed with the origin being the nominal interaction point. The z-axis points in the direction of the beam pipe, the x-axis points to the centre of the LHC ring and the positive y-axis is defined as pointing upwards [10]. Additionally, the azimuthal angle ϕ is the angle around the beam axis (z) while the polar angle θ is measured from the beam axis.

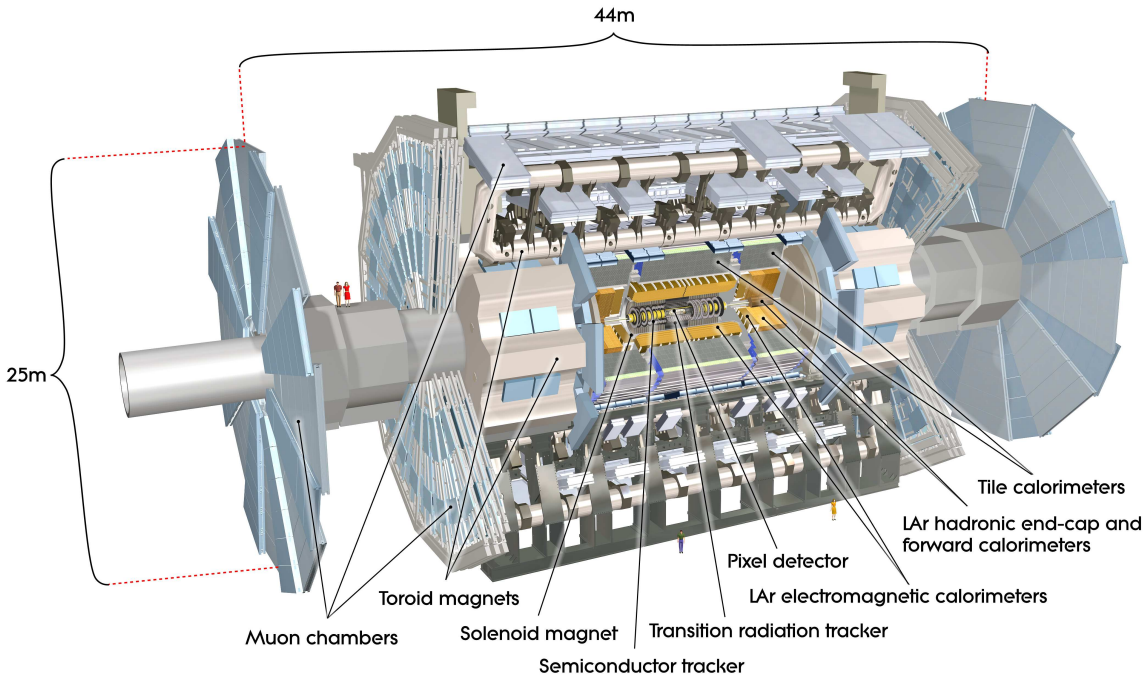


Figure 2.3: Cut-away view of the ATLAS detector, showing the pixel detector, electromagnetic and hadronic calorimeters, muon system and the magnet system [10].

Based on these coordinates some other useful variables can be defined. The *pseudorapidity* η replaces θ and is defined as

$$\eta = -\ln \tan\left(\frac{\theta}{2}\right). \tag{2.1}$$

The distance ΔR between objects is defined as

$$\Delta R = \sqrt{\Delta\eta^2 + \Delta\theta^2}. \tag{2.2}$$

The transverse momentum is defined in the x-y plane:

$$p_{\mathrm{T}} = \sqrt{p_x^2 + p_y^2}. \tag{2.3}$$

With the definition of the transverse plane around the beam axis, the total transverse momentum of all final states cancels out. If any deviation from this can be measured, one can conclude that particles have passed through the detector without interacting.

Lastly, transverse mass m_{T} is defined as

$$m_{\mathrm{T}}^2 = E_{\mathrm{T}}^2 - p_{\mathrm{T}}^2 = \sum p_{\mathrm{T}}^2 - \sum \vec{p}_{\mathrm{T}}^2 \tag{2.4}$$

with the transverse energy $E_{\mathrm{T}} = \sqrt{p_{\mathrm{T}}^2 + m^2}$. It is invariant under Lorentz boosts along the beam axis.

The magnet system

The ATLAS detector is equipped with a system of large superconducting magnets, consisting of a solenoid, a barrel toroid and two end-cap toroids. The system spans 26 m in length and has a diameter of 22 m, storing an energy of 1.6 GJ [10]. The central solenoid generates a magnetic field of 2 T for the inner detector, bending any charged particles in the transverse plane, while the barrel and end-cap toroids provide a 4 T magnetic field for the muon spectrometer [16].

The Inner Detector

The Inner Detector (ID) is the part of the ATLAS detector closest to the beam pipe and has a radius of about 1 m and a length of 7 m. It is designed to provide high resolution tracking in order to reconstruct primary and secondary vertices, as well as charged particle tracks within the pseudorapidity range $|\eta| < 2.5$. The ID consists of three sub-components: pixel detectors, silicon microstrip trackers (SCT) and the transition radiation tracker (TRT). The 2 T magnetic field provided by the central solenoid bends the tracks of charged particles, allowing for the measurement of momentum and charge [17]. An overview of the ID is shown in Figure 2.4.

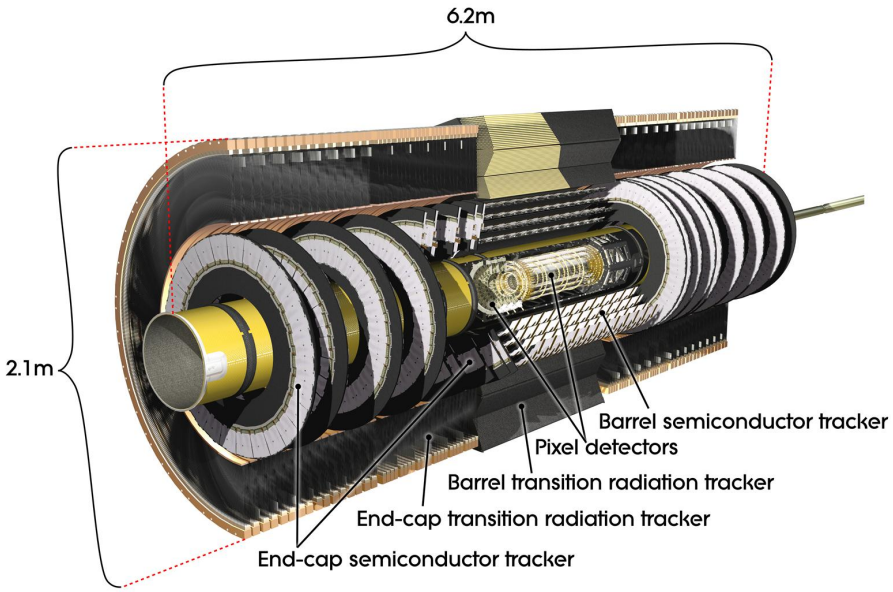


Figure 2.4: Cut-away of the ATLAS Inner Detector, showing the arrangement of pixel detectors, SCT and TRT [10].

The pixel detector is the component of the ID closest to the beam pipe and consists of four layers of high precision semiconductor pixels. The innermost layer, called Insertable B-Layer (IBL), is located only 31 mm from the beam pipe and can achieve a resolution of $8 \times 40\,\mu\text{m}$ [18].

The SCT is a silicon microstrip detector located around the pixel detectors. It works similarly to the pixel detectors, however it uses readout strips instead of individual pixels. The module consists of four cylindrical barrel layers covering the range $|\eta| \leq 1.4$ and 9 disks per end-cap for the remaining range $1.4 < |\eta| < 2.5$. It can achieve a resolution of $17 \times 580\,\mu\text{m}$.

The TRT is a gaseous ionisation detector, consisting of many layers of straw tubes, each acting as an individual wire chamber. Particles pass through the Xenon-filled tubes and cause a hit by ionising the gas. Radiators between the straw tubes increase the transition radiation from particles passing through the boundary between different materials [19]. Every straw has a diameter of 4 mm and contains a gold-plated W-Re wire. A total of 50 000 are located in the barrel and an additional 320 000 make up the TRT end-cap.

The calorimeters

The ATLAS calorimeter system is designed to measure the energy of particles and covers a range of $|\eta| < 4.9$. It consists of the electromagnetic calorimeter (ECal) and the hadronic calorimeter (HCal), as well as the forward calorimeter (FCal) for the high-η region. High energy particles enter the calorimeter and produce secondary particles by interacting with the material, resulting in cascading particle showers. In an electromagnetic calorimeter electrons lose their energy via bremsstrahlung and photons lose their energy via pair production. In a hadronic calorimeter hadrons lose their energy via nuclear interactions. Through these processes the energy of particles is deposited in the calorimeters and can be measured.

The ECal is a liquid argon (LAr) electromagnetic calorimeter, measuring the energy of electrons and photons. It consists of the barrel section ($|\eta| < 1.475$) and two-end cap components ($1.375 < |\eta| < 3.2$). Its primary job is to measure the energy of electrons and photons.

The HCal consists of three components. The tile calorimeter is a sampling calorimeter located in the barrel and covering the region $|\eta| < 1.7$. It is made up of steel as the absorber material and scintillating tiles as the active material [10]. The hadronic end-cap calorimeter is an LAr calorimeter consisting of two wheels per end-cap and covers a pseudorapidity range of $1.5 < |\eta| < 3.2$. Copper is used as an absorber material.

Lastly, the FCal is an LAr calorimeter integrated into the end-cap cryostats consisting of three layers, covering the pseudorapidity range of $3.1 < |\eta| < 4.9$. The first layer uses a copper absorber and is optimised of electromagnetic calorimetry while the other two layers are made of tungsten and measure energy of hadrons.

The muon spectrometer

The muon spectrometer (MS) forms the outer layer of the ATLAS detector. It is designed to track muons passing through the detector and measure their momenta in the region $|\eta| < 2.7$. The muon system consists of trigger chambers for fast muon identification and high-precision chambers for accurate momentum measurements. Bending of particle tracks is provided by the large toroid magnets integrated into the muon system.

The trigger chambers are designed to identify muons in the range $|\eta| < 2.4$. It contains Resistive Plate Chambers (RPC), a type of spark counter, in the barrel region and Thin Gap Chambers (TGC), multi-wire proportional chambers with a small wire-to-cathode distance in the end-cap. These detectors are specialised on quickly identifying bunch-crossings and discriminating the p_T of muons, allowing for fast identification of potentially interesting events.

Monitored drift tube chambers (MDT), a form of gaseous tube detectors, make up the majority of the high-precision muon chambers, both in the barrel and end-caps. They allow for precision tracking of muons throughout the entire muon system. Additionally, in the first layer of the end-cap at

$|\eta| > 2$, where high counting rates are expected, Cathode-strip chambers, another type of multi-wire proportional chamber, are used.

The trigger system

During operation, proton bunches cross at a rate of 40 MHz (every 25 ns) with each bunch containing 10^{11} protons. This results in a total collision rate of 1 billion proton-proton collisions per second. The amount of data generated at this rate is 70 TB/s, which is impossible to store and analyse [16]. To mitigate this, several triggers are used to identify potentially interesting events and thereby step down the data rate considerably.

The first trigger in ATLAS is the Level 1 trigger (L1). It is a hardware trigger, based fully on custom electronics. Using rough data from the calorimeters and muon system it can identify potential Regions-of-Interest (RoI), regions of the detector where the L1 trigger sees possible trigger objects, and reduces the event rate to 100 kHz [20].

If an event passes the L1 trigger, it is passed onto the high-level trigger (HLT) which is software-based. It uses full precision calorimeter and muon data, as well as data from the ID with fast reconstruction algorithms to identify objects that would be interesting in physics analyses, such as missing transverse momentum, jets or tau leptons. This way, the HLT reduces the event rate to 1 kHz. These events are then written to permanent storage, where they can be studied in detail in offline physics analyses.

Particle identification & object reconstruction

Particles leave a unique signature as they pass through the ATLAS detector. Information from the trackers, calorimeters and muon spectrometers is used in reconstruction algorithms to identify these objects. Typical signatures of commonly seen particles are shown in Figure 2.5.

To reconstruct tracks in the inner detector, first raw hits are combined into clusters using a clusterisation algorithm [22]. From these clusters tracks are built and lastly fits to the tracks are performed. Additionally vertices, points where multiple tracks originate from, are reconstructed using a vertex finding algorithm [23]. They can be classified into primary vertices, a collision point, and secondary vertices, a point where a particle decayed into multiple charged particles.

Calorimeter hits are combined into clusters of cell signals to extract a significant signal from any background. These clusters can then be used to reconstruct physics objects. A typically used algorithm for this task is the topological cell clustering algorithm [24].

In the MS hits identified in each muon chamber are used to form segments, which are then fitted together to form muon tracks [25].

From these reconstructed sections physics objects can be identified:

Electrons Electrons are reconstructed by combining tracks in the inner detector with energy deposits in the electromagnetic calorimeter. In the precision region of the detector ($|\eta| < 2.47$) electrons are selected using a likelihood-based (LH) identification, taking into account tracks, calorimeter clusters and combined tracking and calorimeter information [26]. Several electron signal efficiencies can be provided by using different working points of the LH discriminant, referred to as *Loose*, *Medium* and *Tight*. Additionally, prompt electrons produced in signal processes are differentiated from background processes by constructing an isolation quantity.

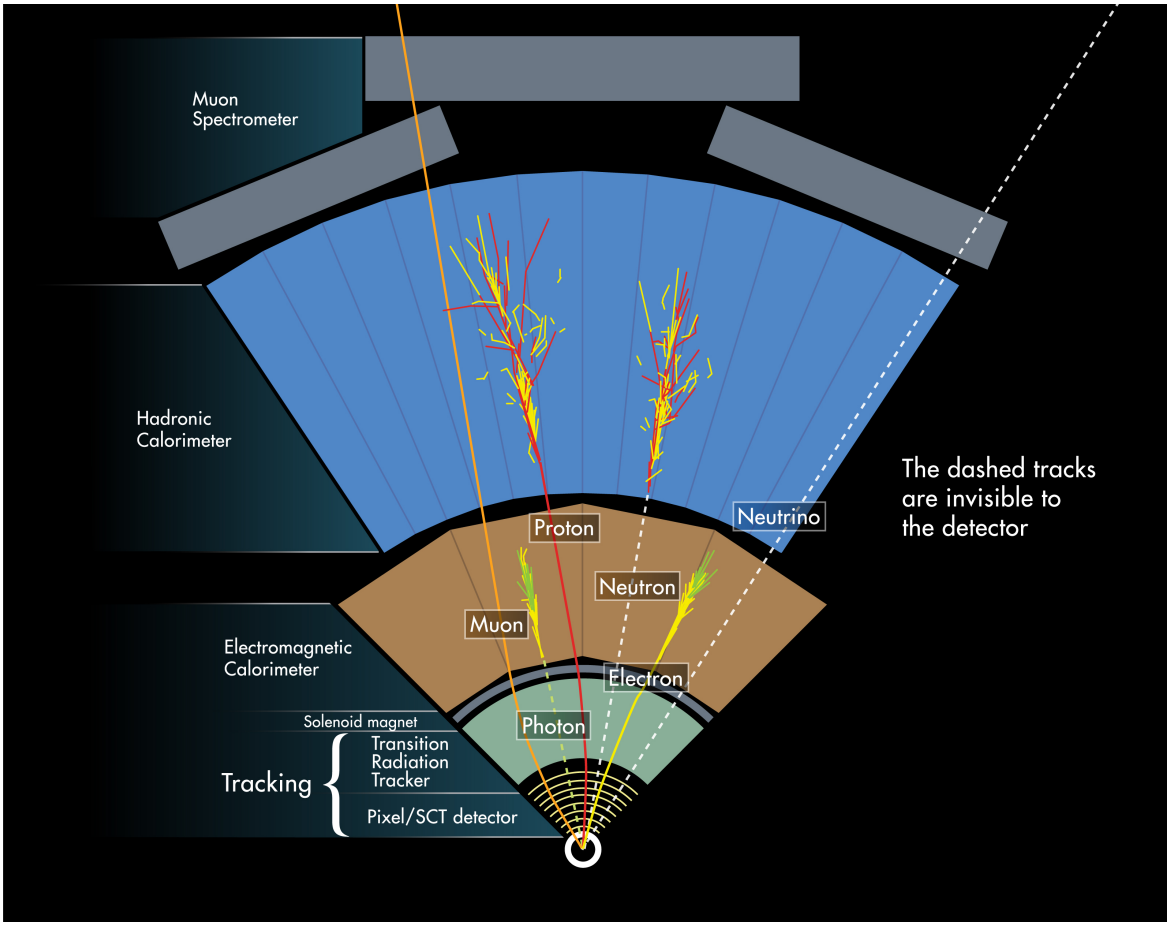


Figure 2.5: Overview of common particle signatures in the ATLAS detector [21]

Muons To reconstruct muons, tracks from the ID and MS are matched, sometimes with additional information from the calorimeters [25]. Similarly to electron reconstruction several working points can be chosen. Isolation criteria also apply.

Jets Jets are streams of particles originating from the hadronisation of a parton. First, clusters in the calorimeters are identified using topological cell clustering [24]. Next, jets are reconstructed from these clusters using the anti-k_T algorithm [27]. To differentiate jets coming from additional pp collisions a jet vertex fraction (JVF) variable is defined and jets above a threshold value can be cut [28].

Tagged jets Jets originating from b-quarks, called b-jets, have characteristic properties that enable them to be identified over jets originating from other quarks. The most important of these properties is the relatively long lifetime of b-quarks, around 1.5 ps, displacing the hadronisation point from the primary vertex by several millimetres, which can be seen as a secondary vertex. At ATLAS, b-tagging is performed with a multivariate algorithm (MV2) that uses the output of several simpler tagging algorithms, typically based on the impact parameter or secondary vertex identification [29].

Missing transverse momentum Because the transverse plane is defined perpendicular to the beam axis, the transverse momentum should be preserved in each event. Any missing transverse momentum E_T^{miss} indicates that neutrinos have escaped the detector undetected. E_T^{miss} is calculated by adding the scalar sum of the p_T of all fully reconstructed particles and jets (*hard* term) and p_T of any charged-particle tracks associated with the hard scatter event but not with any of the reconstructed objects (*soft* term) [30].

2.3 Top quark physics

The only quark that can be measured directly is the top quark, making it a particularly interesting object to study for particle physicists. This work will focus on the production of a single top quark alongside a W boson.

2.3.1 The top quark

The top quark was first predicted by Kobayashi and Maskawa in 1973 and, after a decade-long search, finally directly discovered independently by the CDF and DØ experiments in 1995 [31, 32].

At a mass of $m_t = (172.76 \pm 0.30)\,\text{GeV}/c^2$ [7] it is the heaviest of all known elementary particles, many times heavier than any other fermion. It is an up-type quark with an electric charge of $+2/3$ and a spin of $1/2$. The top quarks lifetime is extremely short at $\tau_t \approx 5 \times 10^{-25}$ s, about twenty times shorter than the hadronisation time, leading to the top quark being the only quark not to form bound hadron states.

The top quark decays to a W boson and a b-quark with a branching fraction of almost 100%. The W boson will then decay leptonically ($\Gamma\left(W \to \ell\nu_\ell\right) \approx 33\%$) or hadronically ($\Gamma\left(W \to q\bar{q}\right) \approx 67\%$), as shown in Figure 2.6.

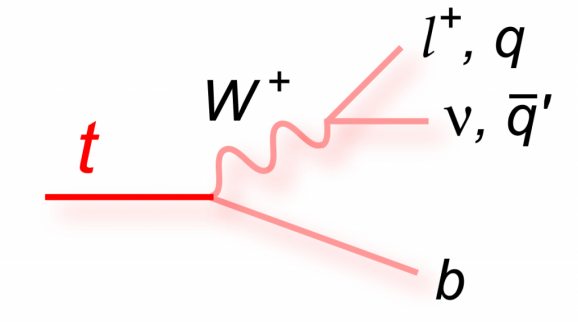


Figure 2.6: Feynman diagram of top decay [33]

2.3.2 Top quark production at the LHC

At hadron colliders top quark production can be classified into two mechanisms: top quark pair production refers to production of a top and an anti-top quark via the strong interaction while single-top quark production is the production of single top quarks via the electroweak interaction.

Top quark pair production

The dominating process of top quark production at the LHC is top quark pair production. At the design LHC center-of-mass energy of $\sqrt{s} = 14\,\text{TeV}$ gluon fusion ($gg \rightarrow t\bar{t}$) dominates leading order $t\bar{t}$ production, making up about 90% of all top quark pairs created in the collider [7]. Quark-antiquark annihilation ($q\bar{q} \rightarrow t\bar{t}$) contributes with about 10%. Both processes are shown in Figure 2.7. The cross section of top quark pair production at the LHC at $\sqrt{s} = 13\,\text{TeV}$ for a top quark mass of $m_t = 172.5\,\text{GeV}/c^2$ is [7]

$$\sigma_{t\bar{t}} = 831.8^{+19.8}_{-29.2} \pm 35.1\ \text{pb}. \tag{2.5}$$

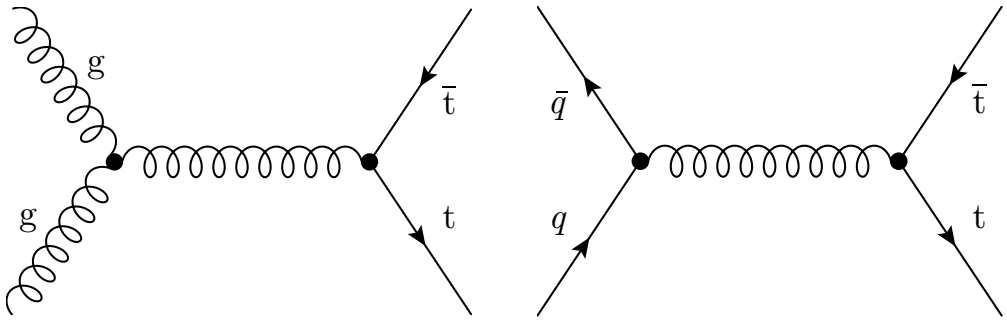


Figure 2.7: Feynman diagrams for leading order $t\bar{t}$ production: (a) gluon fusion, (b) $q\bar{q}$ annihilation

Single top quark production

Single top quarks are produced using three mechanisms: t-channel, s-channel and associated production of a top quark and W boson (tW-channel). They have smaller cross sections compared to $t\bar{t}$ production. Examples of leading order diagrams for each channel are shown in Figure 2.8.

The t-channel, where a sea b-quark exchanges a W boson with a light quark and turns into a top quark, dominates at the LHC with a cross section of $\sigma_{t-\text{channel}} = 217.0^{+9.0}_{-7.7}$ pb at $\sqrt{s} = 13\,\text{TeV}$ [34]. In the s-channel a light quark-antiquark pair annihilates to a virtual W boson, creating a top-bottom quark pair. Due to the difficulty of finding an initial antiquark, this process has a low cross section at the LHC with $\sigma_{s-\text{channel}} = (10.3 \pm 0.4)$ pb at $\sqrt{s} = 13\,\text{TeV}$. Lastly, the process where a bottom quark interacts with a gluon, producing a top quark and an on-shell W boson, is called tW-channel. At $\sqrt{s} = 13\,\text{TeV}$ this channel has a cross section of $\sigma_{tW-\text{channel}} = (71.7 \pm 1.8 \pm 3.4)$ pb.

2.3.3 The tW dilepton channel

The final state of the tW-channel consists of a W boson and a top quark. As mentioned in Section 2.3.1 the top quark will decay to a bottom quark and a W boson with branching fraction $\mathcal{B} \approx 100\%$.

The b-quark can be measured as a b-tagged jet, while the two W bosons can each decay either hadronically or leptonically, leading to three different final states:

- both W decay hadronically ("all-hadronic" channel);

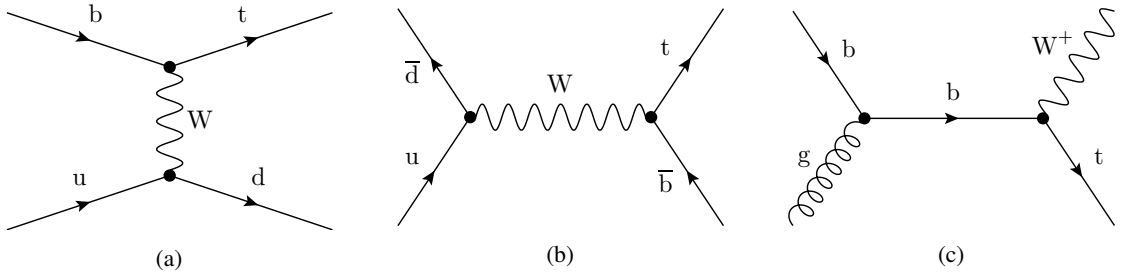


Figure 2.8: Example diagrams for leading order single top production: (a) t-channel, (b) s-channel, (c) tW-channel.

- one W decays hadronically and the other leptonically ("lepton+jets" channel);
- both W bosons decay leptonically ("dilepton" channel).

Despite having the lowest branching fraction, the tW dilepton channel is the easiest to separate from any backgrounds due to its clean final state. It will be the only tW decay mode considered in this work.

Event selection

A leading order diagram with the full final state of the tW dilepton channel is shown in Figure 2.9.

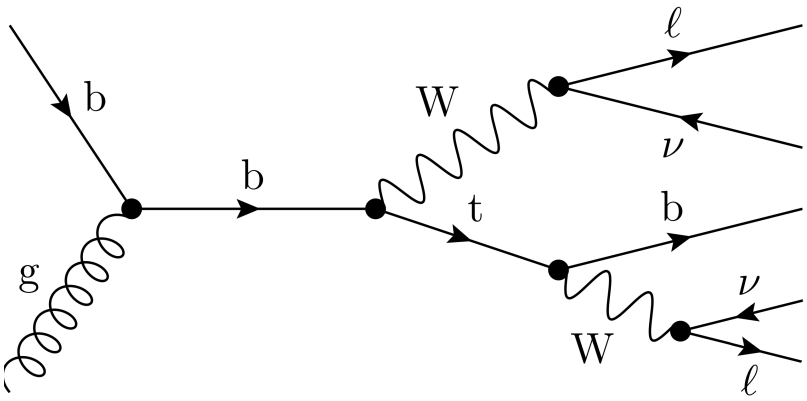


Figure 2.9: Leading order example of the tW dilepton channel with full final state

The expected signature contains one b-tagged jet, two opposite charge leptons and missing transverse energy from the neutrinos in the leading order. At next-to-leading order (NLO) gluon splitting can produce an additional quark in the final state. In order to reduce various backgrounds, the following cuts are applied:

- Single electron or muon trigger passed
- For electrons: tight identification, isolation, $E_{\mathrm{T}} > 26\,\mathrm{GeV}$
- For muons: tight isolation, $p_{\mathrm{T}} > 26\,\mathrm{GeV}$ OR no isolation, $p_{\mathrm{T}} > 50\,\mathrm{GeV}$
- Opposite sign electron-muon pair

- Leading lepton $p_{\mathrm{T}} > 27\,\mathrm{GeV}$
- Veto if third lepton $p_{\mathrm{T}} > 20\,\mathrm{GeV}$
- At least one jet with $p_{\mathrm{T}} > 25\,\mathrm{GeV}$, $|\eta| < 2.5$, b-tagged at 77% WP

$t\bar{t}$ background

By far the most important background in the tW dilepton channel is top quark pair production with both W bosons decaying leptonically. Its final state has only one additional b-quark compared to the tW dilepton final state. Additionally, its cross section is an order of magnitude larger than the cross section of tW:

$$\sigma_{tW} \approx 72\,\mathrm{pb}; \tag{2.6}$$

$$\sigma_{t\bar{t}} \approx 832\,\mathrm{pb}. \tag{2.7}$$

In order to best deal with this background three regions are defined: The highest fraction of tW signal is expected in the `1j1b` region, defined by events having exactly one b-tagged jet and no additional jet. Because of radiative corrections, `2j1b` is also considered a signal region. It contains two jets of which one is b-tagged. Lastly, in order constrain the $t\bar{t}$ background normalisation, a `2j2b` control region is defined. It contains events with exactly two jets, both of which are b-tagged. In this work, only the `1j1b` and `2j2b` regions will be considered.

2.4 Monte Carlo simulations

Monte Carlo simulations are an important ingredient in physics analyses at ATLAS. Data simulated using the current understanding of the Standard Model can be compared to real data from the ATLAS detector to find new physics and is also typically used to develop new analyses without having to work on real data. An aspect crucial to this work is the availability of *truth labels* in MC simulated data. These truth labels contain the true origin of an event, thereby enabling the use of machine learning algorithms, which rely on tagged data to train. There are two steps to MC simulations at ATLAS: First, events are generated from the initial collision to the final states. Next, the response of the ATLAS detector to these events is modelled.

Event generation During event generation, the full physics process leading to a certain final state is simulated. An overview of this process is shown in Figure 2.10. At first the initial hard scatter is simulated, followed by the resulting cascade of colour charged particles being radiated, called parton shower. Next, the hadronisation of resulting partons due to confinement as well as their subsequent decays are modelled. Additionally, any secondary interactions between the proton remnants also have to be simulated [35].

Detector simulation In the next step, the response of the ATLAS detector is simulated. The includes modelling the path of each particle through the detector, its interaction with the various materials and the response of the electronics [37]. Afterwards, the event is reconstructed as described in Section 2.2.2.

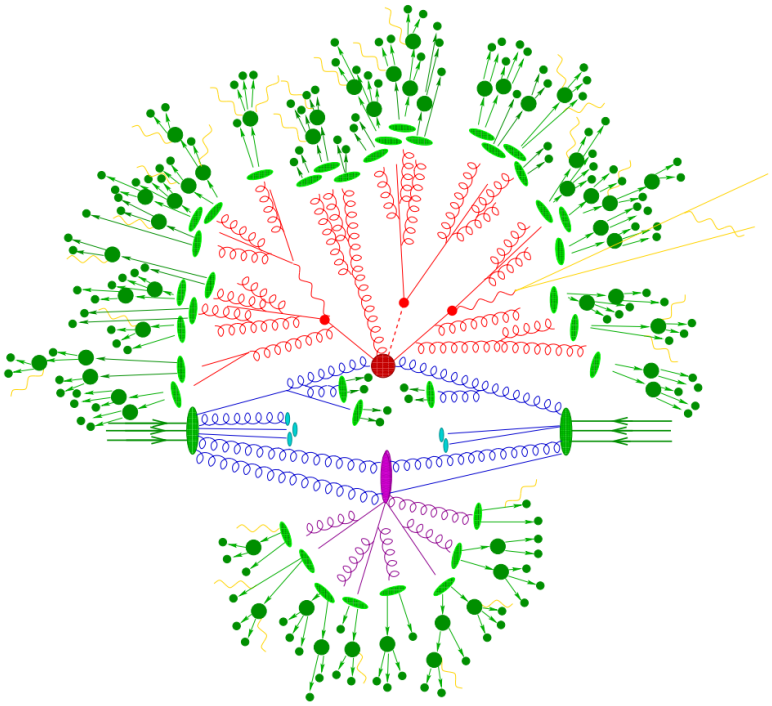


Figure 2.10: Overview of a proton-proton collision simulated during event generation. The initial hard scatter is represented by the large dark red blob, surrounded by parton showers in blue and red. Hadronisation events are shown in light green, subsequent decays in dark green. The secondary scattering event is shown in purple, soft photon radiation in yellow [36].

Nominal tW dilepton events are generated using Powheg-Box 1 [38], parton showering is simulated using Pythia 8 [39]. $t\bar{t}$ simulation is performed using Powheg-Box 2, with parton showering again being modeled by Pythia 8. The detector simulation is performed by the Geant 4 simulation toolkit [40].

2.4.1 Systematic uncertainties

MC simulations cannot perfectly reflect real physics. Instead, processes are represented by imperfect models. This introduces systematic uncertainties that have to be accounted for in physics analyses. This work will take a closer look at the behaviour of two systematic uncertainties in the tW dilepton channel, *DR/DS* and *Parton showering*.

Diagram Removal/Diagram Subtraction (DR/DS)

At next-to-leading order (NLO) tW and $t\bar{t}$ begin to interfere. Examples of two interfering diagrams (*doubly resonant* diagrams) are shown in Figure 2.11. Due to $t\bar{t}$ having a much higher cross-section this affects tW in a significant way that has to be accounted for [41]. Two treatments of this interference term in the calculation of the tW cross section exist:

- **Diagram Removal (DR)**: Any doubly resonant diagrams in the NLO tW amplitude are removed

from the calculation.

- **Diagram Subtraction (DS)**: A subtraction term is implemented, cancelling the $t\bar{t}$ contribution at the cross section level

Results should be calculated using both of these methods, and if similar, the difference between the two schemes can be accounted for as a systematic uncertainty. Here, *DR* is used as the nominal sample while *DS* is used as the systematic sample.

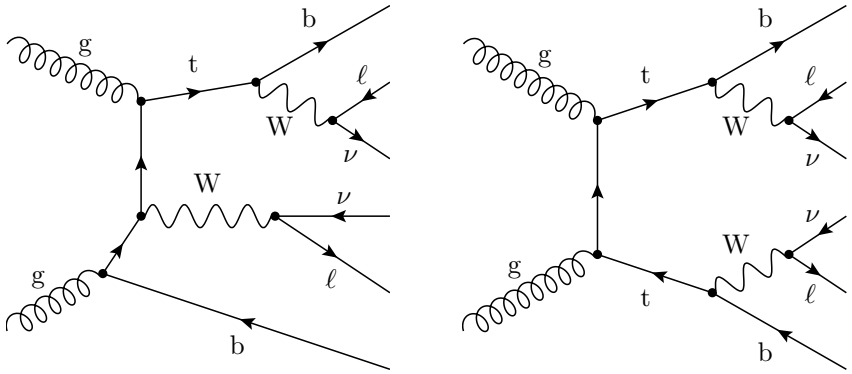


Figure 2.11: Example diagrams for (a) NLO tW production, (b) LO $t\bar{t}$ production with identical final states.

Parton showering (PS)

A crucial step in the generation of events is the modelling of parton showering and hadronisation. The nominal sample is generated using Pythia 8, which incorporates the Lund string model. Herwig 7 [42] uses another method, the cluster model, to calculate this step. Both samples are compared to estimate the systematic uncertainty.

CHAPTER 3

Machine Learning with Neural Networks

This chapter will introduce the concept of machine learning using advanced neural network techniques. Section 3.1 gives a general introduction into the technique of machine learning, Section 3.2 will explain one of the most advanced methods of machine learning, neural networks. Lastly, Section 3.3 will detail the specific type of neural networks used to conduct this analysis.

3.1 Introduction to machine learning

ENIAC, designed and built during the early 1940s, is widely considered to have been the first general-purpose digital computer. The room-filling device was made of thousands of vacuum tubes, capacitors and resistors and could perform 5 000 computations per second [43]. Since then the capability of computers has been increasing exponentially, largely due to the invention and subsequent miniaturisation of transistors, allowing for new mathematical techniques to be developed.

One such technique is the concept of machine learning, a process where computers can learn certain tasks based on training data. Previously, while computers could process data faster than humans, every step still had to be defined by hand, requiring a detailed understanding of the dataset. Machine learning can greatly simplify this step by making the computer learn any connections by itself, based on some set of input data.

Machine learning has been a quickly growing discipline over the last decades. As computers become powerful enough to solve more complex problems, new applications for machine learning are being found at a high rate. One technique that has seen particular attention in this field has been the artificial neural network, a type of machine learning that is loosely based on the functioning principle of the human brain. It works by building a network of interconnected nodes and learns by adjusting the threshold at which each node fires. Neural networks have been used in many applications, from successfully playing the game Go, a game previously thought to be too complex for computers, over object recognition in computer vision to powering recommendation engines on many large websites.

In physics, machine learning has seen increased use in all steps of the analysis process. Examples include triggering, object reconstruction or simulation [44]. In this work, it will be used to aid in separation of signal to background samples, using many variables. Only subtle differences exist between the samples, making this a difficult task, even for modern machine learning solutions.

However, many challenges still exist in this field. Models have to be individually designed and any free parameters optimised separately for every task, a step that still mostly involves human work

and requires some understanding about the input data. Additionally, as these models become more complex and the amount of data increases, performance is still a major factor that cannot be ignored. With inadequate hardware, complex models can take many hours or days of computation time to train to a reasonable level. New hardware is being developed to cope with this. Of particular interest is exploiting the high parallel computing capabilities of graphics processors (GPUs) that were originally designed to render 3D scenes.

3.2 Neural networks

The fundamental building block of the human brain is a type of cell called neuron. It receives signals from several other cells, processes them, and passes the result on to other neurons through nerve fibres called axons. The brain is made up of almost 100 billion neurons in a network so complex, little is understood about it to this day.

Neural networks aim to capture these biological principles in an artificial network. Instead of human cells being connected through axons, many *nodes*, usually arranged in layers, are connected together using *weights*. Each node processes any incoming data using an *activation function* and passes it along to other nodes. To train, data is fed into the network through an input layer and after having travelled through the network the output is compared to the desired result in a *loss function*. Using a technique called *backpropagation*, weights are updated to nudge the output closer to the expectation. Each update of weights is called an *epoch*. As this process is repeated many times, the network learns to identify patterns in the data and can then be used to process previously unknown data. An example of a full neural network is shown in Figure 3.1.

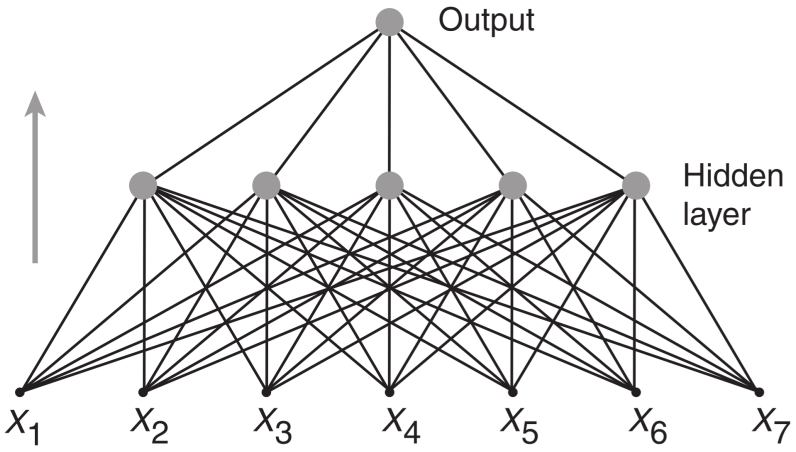


Figure 3.1: Example of a neural network. The input layer with variables x_1 to x_7 can be seen at the bottom. Data is passed to the nodes in the hidden layer, and then to the single output node.

3.2.1 The perceptron model

The fundamental principle of a neural network is the *perceptron model*. It was first proposed by Warren McCulloch and Walter Pitts in 1943 and subsequently improved. An overview of its components is shown in Figure 3.2. The *node* is the equivalent of the neuron cell, receiving signals $x_1 \dots x_N$

from many other nodes. Weights $w_1 \dots w_N$ are applied to the signals before reaching the node. Additionally, an optional constant bias b can be added. In the node, the sum

$$\left(\sum_{i=1}^{N} x_i w_i\right) + b \tag{3.1}$$

is computed. The result is then passed to an *activation function* $f(\Sigma)$. Similar to neurons in the brain, it decides whether the node "fires", depending on the input. It can also be used to normalise the output. The result of the activation function is then sent to any connected nodes in the network.

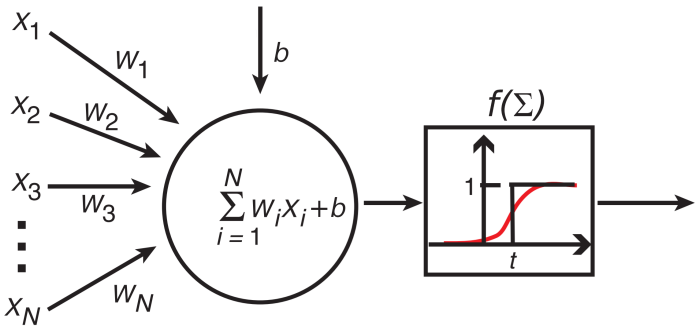


Figure 3.2: Overview of the perceptron model, showing the central node, incoming connections x_N with weights w_N, an optional bias b and the output through an activation function f. The step function is shown in black, the sigmoid function in red.

Activation functions

The choice of activation function can greatly influence the performance of a neural network, a selection of the most common ones is described in the following.

Step function In the brain, neurons fire with a predefined signal if the sum of inputs is above a threshold, and do not fire at all if not. This is represented by the step function

$$f(x) = \begin{cases} 0 & x < t \\ 1 & x \geq t \end{cases}, \tag{3.2}$$

where t is the threshold. An advantage of this function is that the output will always be either 0 and 1, so it cannot "explode".

Sigmoid function Another activation function is the sigmoid function. Unlike the step function it is continuous and is usually preferred as it can capture more information. The sigmoid function is defined as

$$f(x) = \frac{1}{1 + e^{-x}}. \tag{3.3}$$

The sigmoid is typically used for the output node in classification tasks.

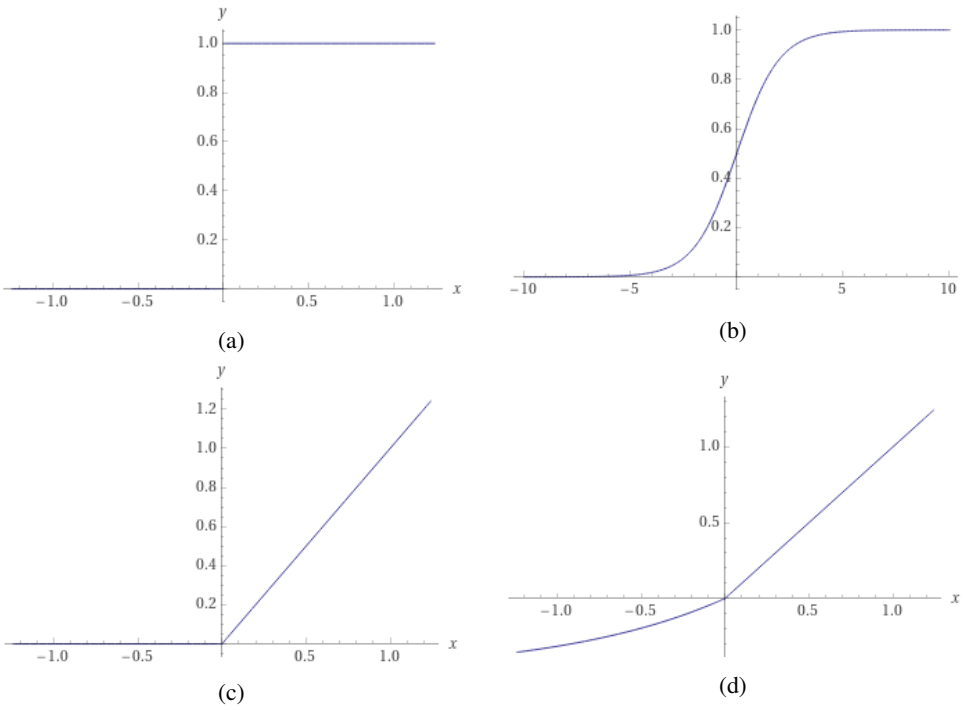


Figure 3.3: Common activation functions: (a) Step function with $t = 0$, (b) Sigmoid, (c) ReLU, (d) ELU

Rectified linear unit function (ReLU) A relatively new development in neural networks is the ReLU activation function. It has the advantage that it doesn't saturate in the positive x direction, often improving learning, and being very efficient to compute. It is defined as

$$f(x) = \begin{cases} 0 & x \leq 0 \\ x & x > 0. \end{cases} \tag{3.4}$$

ReLU has become the preferred activation function for deep neural networks.

Exponential linear unit function (ELU) ELU is an evolution of the ReLU function, defined as

$$f(x) = \begin{cases} \alpha(e^x - 1) & x \leq 0 \\ x & x > 0. \end{cases} \tag{3.5}$$

Instead of falling to 0 for all $x \leq 0$ it transitions into an exponential function, eliminating the discontinuous point at $x = 0$ and shifting the mean of activations closer to zero, which has been shown to improve learning [45]. It is, however, more computationally intensive.

3.2.2 Training the network

After the network structure has been built, it has to be trained. This is typically done through an algorithm called *backpropagation*, where the deviation from the desired result is calculated and its

gradient is propagated back to the weights.

For signal/background classification in physics analyses, the output of the network needs to be in binary classification mode. In this case, the output will range between 0 and 1, with 1 being an event that is definitely signal and 0 being an event that is definitely background. Values in between are meant to represent the probability that a certain event is a signal event.

Loss function

At the beginning of training the output will be essentially random. Any data used as training data has to be labelled with a *truth label*, containing the correct classification of each event. To determine the deviation from the output to this truth label, *loss function* is used. It computes the error that the network makes as each training event passes through it. As it trains, the value of the loss function should decrease, signalling that the network is learning more features in the data and improving its classification performance. For binary classification tasks the most common loss function is *binary cross-entropy*. It is defined as

$$L = -\left(y \ln\left(p\right) + \left(1 - y\right) \ln\left(1 - p\right)\right), \tag{3.6}$$

where y is the truth label and p is the estimated probability for the network's guess $\hat{y}$.

Optimisers

The training of the neural network can be seen as finding the global minimum in a multi-dimensional function called *loss surface*, representing the impact of every parameter in the neural network on its performance, such as the weights for every connection. This is done by using an *optimiser* algorithm.

Gradient Descent (GD) The most basic optimiser algorithm is called *Gradient Descent*. It works by calculating the gradient at training step t

$$g_t = \frac{1}{N} \nabla \sum_{i=1}^{N} L\left(f\left(\hat{y}_i, \Theta_t\right), y_i\right) \tag{3.7}$$

where i represents one event in the dataset of size N and L is the loss function based on the network configuration f and truth labels y_i. Θ denotes the parameters that are optimised during network training. Based on this gradient all parameters are updated as follows:

$$\Theta_{t+1} = \Theta_t - \ell g_t. \tag{3.8}$$

ℓ is a parameter called learning rate. It determines the size of steps taken with each update. If ℓ is too low, training may take a very long time, while a too high learning rate may cause the network to be unable to find the global mininmum or even not to converge at all. The optimal value for ℓ changes depending on the network topology, so it has to be optimised as one of the *hyperparameters*, free parameters in the neural network that are adjusted by hand for best performance.

Stochastic Gradient Descent (SGD) with momentum GD has some significant disadvantages. A single step is calculated on the whole training dataset, severely slowing down training if the dataset

is large. Additionally, it is prone to get trapped in local minima and saddle points. The *Stochastic Gradient Descent* algorithm improves training time significantly by updating after a small sample of the dataset, called *batch*. This is achieved by modifying the gradient to

$$g_t = \frac{1}{m} \nabla \sum_i L\left(f\left(\hat{y}_i, \Theta_t\right), y_i\right) \tag{3.9}$$

with m being the amount of events processed in each step, also called *batch size*. The optimal batch size depends on the capabilities of the hardware, though it should not be large enough to contain a significant fraction of the entire dataset.

A further improvement is the addition of *momentum*. It is designed to modify the step size based on previous gradients, thereby speeding up training and reducing the probability of getting stuck in local minima. Instead of directly updating the weights, the velocity v is first updated:

$$v_t = \alpha v_{t-1} - \ell g_t(\Theta_t) \tag{3.10}$$

α is called *momentum* and is another hyperparameter to be optimised. Parameters are then updated with the velocity according to

$$\Theta_{t+1} = \Theta_t + v_t \tag{3.11}$$

Nowadays a modification of momentum, called *Nesterov momentum* [46] is often used. It effectively changes from which point in parameter space the gradient is calculated, giving it the ability to "look ahead". This can help if the momentum term does not point in the direction of minimised loss, in essence correcting it in the same step. The momentum equation is changed to

$$v_t = \alpha v_{t-1} - \ell g_t(\Theta_t + \alpha v_{t-1}). \tag{3.12}$$

Adaptive moment estimation (Adam) A very popular and more advanced optimiser is the *Adam* algorithm [47]. Classic gradient descent based optimisers struggle when the topology of the loss surface changes rapidly, since parameters cannot be easily tuned to every step of the training process. Adam tackles this issue by computing adaptive learning rates for each parameter based on an exponential moving average of gradients. The first and second moments are estimated with

$$v_t = \left(\beta_1 v_{t-1} + \left(1 - \beta_1\right) g_t\right) \frac{1}{1 - \beta_1^{t+1}}; \tag{3.13}$$

$$r_t = \left(\beta_2 r_{t-1} + \left(1 - \beta_2\right) g_t^2\right) \frac{1}{1 - \beta_2^{t+1}}. \tag{3.14}$$

The parameters Θ_t are then updated using

$$\Theta_{t+1} = \Theta_t - \frac{\ell v_t}{\sqrt{r_t} + \epsilon}. \tag{3.15}$$

β_1 and β_2 are hyperparameters controlling the decay rate of the exponential moving averages, ϵ is small and exists to prevent the denominator from blowing up with very small r_t. Adam is considered to be the most widely used optimiser.

Weight initialisation

In order to prevent activations to either go to zero or explode, preventing efficient training, all weights in the network have to be initialised. Many initialisers exist, one of the most common is the *Xavier normal initialiser* [48]. It initialises weights by drawing from a truncated normal distribution with mean 0 and standard deviation

$$\sigma = \sqrt{\frac{2}{m}}, \tag{3.16}$$

where m denotes the number of incoming connections, sometimes also referred to as *fan in*.

3.2.3 Overfitting and regularisation

Overfitting

The goal of training the neural network is that it picks up general trends in the data and is capable of then categorising previously unknown data with high precision. However, if the network parameters are not tuned correctly or it trains for too long, it might pick up fluctuations and random noise in the training sample. This process is called *overfitting* or *overtraining*. A simplified illustration of overfitting can be seen in Figure 3.4.

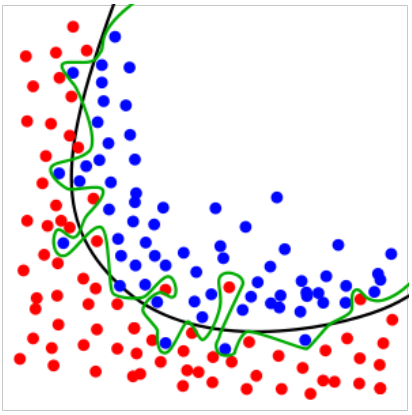


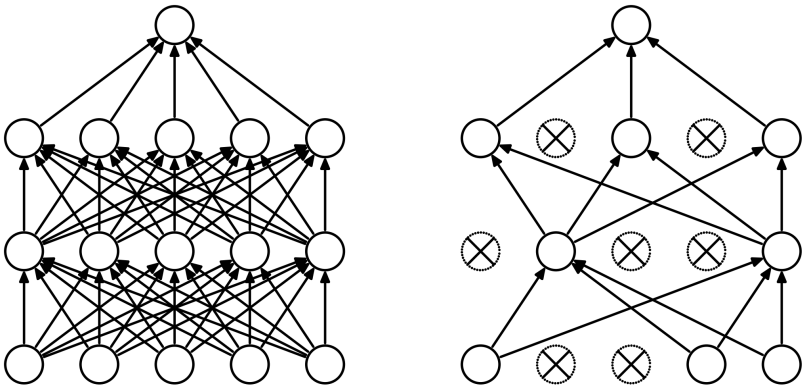
Figure 3.4: Simplified illustration of overfitting. Red and blue dots represent data that should be separated by the neural network. The black line represents an optimal fit to the data, while the green line represents an overfitted network picking up random fluctuations [49].

In order to ensure the quality of the training and identify potential overfitting, the dataset is split into two parts: the *training sample* and the *validation* or *test sample*. The training sample is used to perform the actual training on the network. The validation sample is not used for any training, instead the network uses this sample only to predict. The performance using the training and validation datasets should be similar. If the network performs significantly better with the training data it is likely that overfitting has occurred. By monitoring performance with both datasets continuously during training, overtraining can be identified and countermeasures can be taken.

Regularisation

Techniques to reduce the chance of overfitting are collectively called *regularisation*. Many methods exist, and finding new ones has been a topic of continuing research in the machine learning field.

Dropout A widely used regularisation technique is *dropout* [50]. It works by dynamically turning off nodes in a layer during each training step, thereby thinning the network and reducing strong correlations between nodes, a typical sign of overfitting. This effectively means that at each training step a slightly different, unique neural network is trained. Additionally, because it reduces the size of the network, each training epoch is accelerated. A sketch of the impact of dropout is shown in Figure 3.5.



(a) Standard Neural Net (b) After applying dropout.

Figure 3.5: Effect of dropout on a neural network: (a) A fully connected network, (b) A network with dropout applied. Some nodes have no incoming or outgoing connections, effectively being turned off

Dropout can be added to every layer separately. 20% dropout for the input layer and 50% dropout for each hidden layer has been suggested as a good starting point [50].

Batch normalisation Another modern regularisation method is *batch normalisation* [51]. In deep neural networks the distribution of the input in each layer shifts during changes, as the parameters of previous layers are adjusted. This is called *internal covariance shift*. It can make it difficult to train these networks and makes them sensitive to learning rates and parameter initialisation.

In order to address these issues, batch normalisation normalises input for every layer, for each training batch. This reduces the sensitivity to initialisation and allows higher learning rates without causing overfitting or even a full failure of the training process.

Each output in the batch $\mathcal{B} = \{x_{1\ldots m}\}$ is normalised to the mean $\mu_{\mathcal{B}}$ and standard deviation $\sigma^2_{\mathcal{B}}$ of the current batch.

$$\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} x_i, \tag{3.17}$$

$$\sigma^2_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} \left(x_i - \mu_{\mathcal{B}}\right)^2. \tag{3.18}$$

The normalised output $\hat{x}_i$ then is calculated with

$$\hat{x}_i = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \varepsilon}}. \tag{3.19}$$

Early stopping A simple and effective method to prevent overtraining is *early stopping*. It works by monitoring one or more metrics and stopping the training if a certain criterion is met [52]. The most typical setup monitors the loss of the validation sample and stops the training if it stops improving for several epochs, as this is a typical sign of overfitting. A visualisation of early stopping is shown in Figure 3.6. The time that the early stopping algorithm waits before stopping training is called *patience*. It should be low enough to stop training quickly enough without triggering due to random fluctuations in the loss.

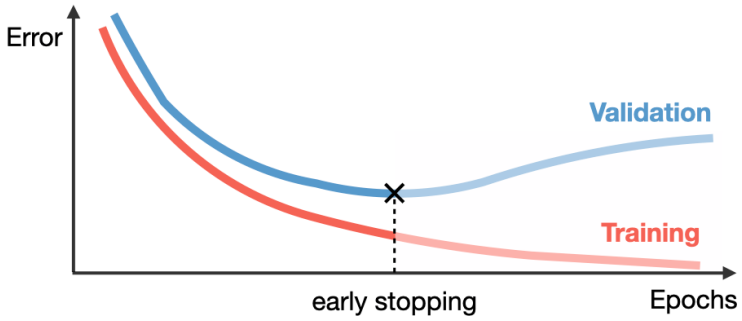


Figure 3.6: Sketch of early stopping, using the loss of the validation sample [53].

3.2.4 Performance metrics

In order to evaluate the quality of a neural network several metrics exist. Generally, it is good to take multiple metrics into account rather than relying on a single one, as they may show different aspects of the training process. The metrics used during this work will be described in the following.

Loss curve A simple way of evaluating a neural network is to inspect its *loss curve*. It shows how the result of the loss function behaves as the network goes through multiple training steps. Ideally, the loss curve should decrease quickly at the beginning and then transition into slowly approaching a minimum, preferably the global minimum. A loss curve that is very noisy, not smooth, reverses direction or otherwise diverges from the ideal shape usually means that the network isn't training correctly or not at all.

The loss curve can also be used to identify overtraining by plotting the losses of the training and test sample together. Generally, both curves should track each other. A test sample loss that diverges from the training sample loss indicates overtraining.

Receiver operating characteristic curve (ROC) The *ROC curve* is a metric specific to binary classification tasks. It plots the false positive rate (FPR) on the x-axis against the true positive rate

(TPR) on the y-axis. These values are defined as:

$$\text{TPR} = \frac{n(\text{true positive})}{n(\text{true positive}) + n(\text{false negative})} \tag{3.20}$$

and

$$\text{FPR} = \frac{n(\text{false positive})}{n(\text{false positive}) + n(\text{true negative})} \tag{3.21}$$

This way, the ROC curve visualises the relation of these two values, depending on at what discrimination threshold is chosen. Random guessing results in a diagonal line with a slope of 1, as the network is equally likely to make a right or wrong decision. A well trained network should result in a smooth curve above the diagonal. A ROC curve that is lopsided, contains sudden jumps or is below the diagonal often indicates issues with the training. It can also help pick a good threshold above which to accept events as signal when the trained model is used for inference. The ROC curve can also be used to identify overtraining in the final, trained model. If it is plotted for both test and training samples, both curves should agree.

An important measure based on the ROC curve is the *area under the curve* (AUC), that can be calculated by simple integrating the ROC curve. It measures the probability that the classifier will rank a randomly chosen positive sample higher than a randomly chosen negative sample [54].

Separation histogram A metric that is very useful in physics-based tasks is the separation histogram. The network response to each event is binned and added to a histogram, depending on the truth label. These histograms are then plotted together, typically after being normalised. This provides an easy to understand visualisation of how good the neural net is at separating signal from background events.

3.3 Adversarial neural networks

Deep neural nets have been very successful at discriminating tasks, taking some high-dimensional input such as an image, an audio waveform or complex data and outputting some class label. However, deep *generative* nets have not seen much success. These are a neural network type that has the ability to generate new data by capturing the joint probability $p(X, Y)$ with X being a set of data and Y being a set of corresponding labels. This is due to the fact that they have to capture much more complex relationships than "simple" discriminative models. While their output may contain some key features of whatever object should be generated, it is very easy to identify as fake. A way to improve this is the use of an *adversarial neural network* (ANN[1]).

3.3.1 Generative Adversarial Networks

ANNs were first introduced as a solution by Ian Goodfellow el. al. in 2014, with the introduction of a *Generative Adversarial Network* (GAN) [55]. Instead of just training a generative neural network, a second network is added that works against the generator. This so-called *adversary* is another discriminator model that tries to learn the task of distinguishing between real data and "fake", generated data. A sketch of this network structure is shown in 3.7. The goal of this model is to create a

[1] Not to be confused with Artificial Neural Networks (ANN). Any mention of "ANN" in this work will refer to adversarial neural networks

competition between the models, where the adversarial sniffing out fakes drives the generative net to produce more real looking data.

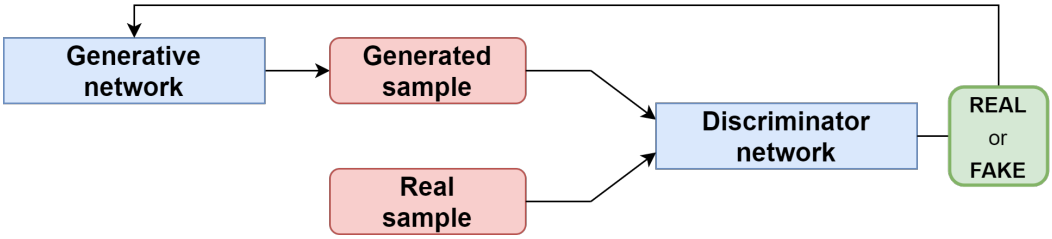


Figure 3.7: Overview of a Generative Adversarial Network

In a GAN, the generator receives input noise variables $p_z(z)$ and maps it to data as $G\left(z;\theta_g\right)$, where θ_g are the parameters of the generative neural network. Another neural network $D\left(\boldsymbol{x};\theta_d\right)$ is trained to maximise the probability of correctly labelling data and samples generated by G. G is trained simultaneously to minimise $\log\left(1-D\left(G\left(z\right)\right)\right)$. This process is represented by the following equation:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}\left[\log D\left(\boldsymbol{x}\right)\right] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}\left[\log\left(1-D\left(G\left(\boldsymbol{z}\right)\right)\right)\right]. \tag{3.22}$$

$V(D,G)$ is the combined value function for both networks.

3.3.2 Adversarial neural network as a pivot

To classify Monte Carlo simulated events into signal and background, classifier neural networks are increasingly the tool of choice. A weakness of using a simple classifier is that it can be sensitive to systematic uncertainties present in the sample, causing a large covariance shift. In order to reduce the impact of these systematic uncertainties, a network model should approach a *pivot*, a quantity whose distribution is invariant with respect to systematic uncertainties.

A solution based on ANNs was introduced by Kagan et.al. in 2016 [56]. In this approach the discriminator network takes the place of the generator in the GAN model. Instead of using a generated and a truth sample as input to the second network, nominal and systematic samples are used. A systematic sample is a variation of the nominal sample with slight shifts in distributions, representing the impact of systematic uncertainties. The adversarial is then trained to differentiate between nominal and systematic samples. If the classifier is trained against this adversary it could reduce the impact of systematic uncertainties on the model and make the whole model similar to a pivot, achieving an overall more robust classification. In this variant of the ANN concept the network equation first introduced in Equation 3.22 becomes

$$\min_f \max_r V(r,f) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{nom}}(\boldsymbol{x})}\left[\log r\left(\boldsymbol{x}\right)\right] + \mathbb{E}_{\boldsymbol{z}\sim p_{\text{sys}}}\left[\log\left(1-r\left(f\left(\boldsymbol{z}\right)\right)\right)\right]. \tag{3.23}$$

f represent the discriminator and r the adversary. The discriminator is trained for best classification, represented by $\log r\left(\boldsymbol{x}\right)$. The second network is trained to differentiate between nominal and systematic distributions represented by $\log\left(1-r\left(f\left(\boldsymbol{z}\right)\right)\right)$.

The loss function takes the place of the value function $V(r, f)$. To train the combined network a combined loss function is required:

$$\mathcal{L} = L_f - \lambda L_r. \tag{3.24}$$

λ is an additional hyperparameter that controls the impact of the adversary on the overall training.

Each training iteration happens in two parts. First, the discriminator is trained using the combined loss function $\mathcal{L}$, then the adversary is trained using its loss function L_r. An overview of the whole setup can be seen in Figure 3.8.

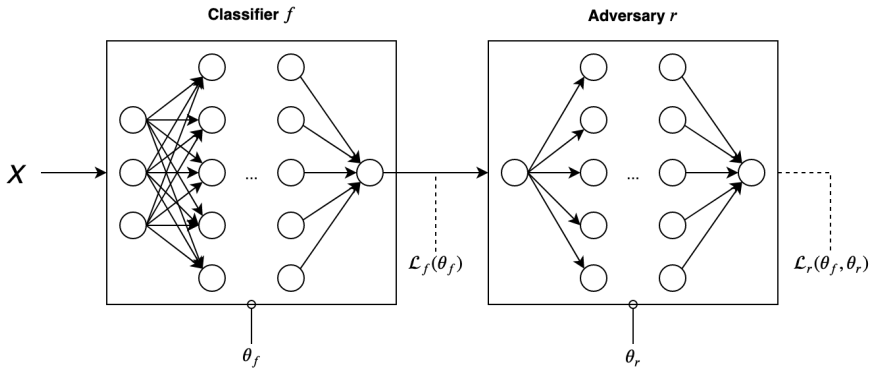


Figure 3.8: Sketch of the ANN setup: MC samples X are fed into the classifier f with parameters θ_f. Its output is passed into the adversary r with parameters θ_r

.

CHAPTER 4

Adversarial Neural Network in the tW Dilepton Channel

In this chapter the setup an adversarial neural network in the tW dilepton channel is described. Section 4.1 introduces the motivation to trial an ANN in this channel, Section 4.2 details the technical implementation using the TensorFlow and Keras libraries. Afterwards, the choice of Monte Carlo samples and variables is described in Sections 4.3 and 4.4. Performance metrics are introduced in Section 4.5. Lastly, the hyperparameters of this network and the strategy of optimising them is detailed in 4.6. Chapter 5 will then show how the process of optimisation could be heavily sped up.

4.1 Motivation

As explained in Section 2.3.3, one of the biggest difficulties in the tW dilepton channel is the separation of tW signal events and $t\bar{t}$ background events. This is due to their very similar signature in the detector, differing by only a single bottom quark. In recent analyses a simple machine learning algorithm for classification problems, "Boosted Decision Trees" (BDT) [57], has been used with good success. An example of separation and ROC curve of a BDT training in the `1j1b` region are shown in Figure 4.1. Recently, a simple feed-forward deep neural network based on Keras has shown similar performance. A comparison plot showing ROC curves for both the BDT and neural network techniques is shown in Figure 4.2.

However, both of these techniques suffer from high systematic uncertainties influencing the training performance. As shown in 3.3.2, an adversarial neural network setup might help reduce the impact of these uncertainties on tW-$t\bar{t}$ separation.

A proof of concept has first been described in [60], however it was plagued by significant issues leaving the viability in question. This thesis will attempt to address two unsolved problems. First, the crippling performance constraints of the network, often taking more than 24 hours for a full training, need to be improved. After that, a more thorough attempt at reducing the instabilities of the network is performed.

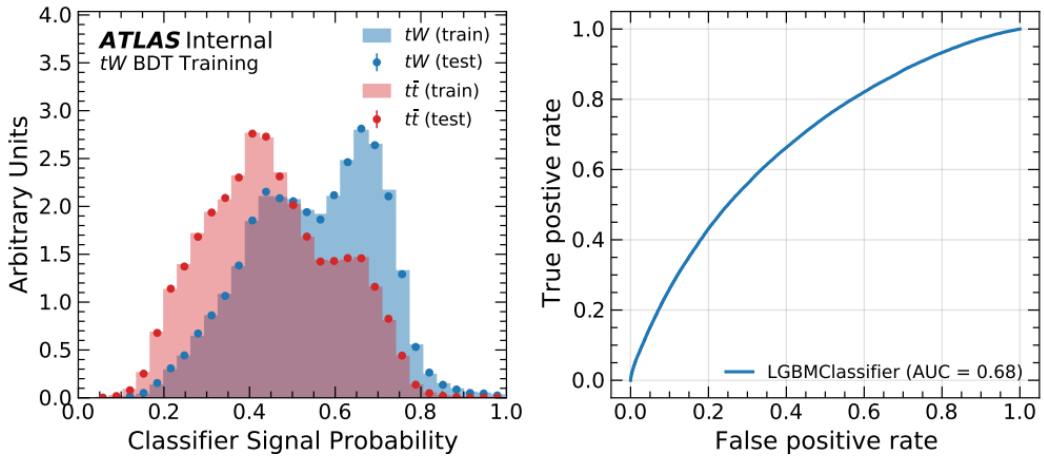


Figure 4.1: Training and testing separation of tW and $t\bar{t}$ in the `1j1b` region using a BDT classifier and its associated ROC curve [58]

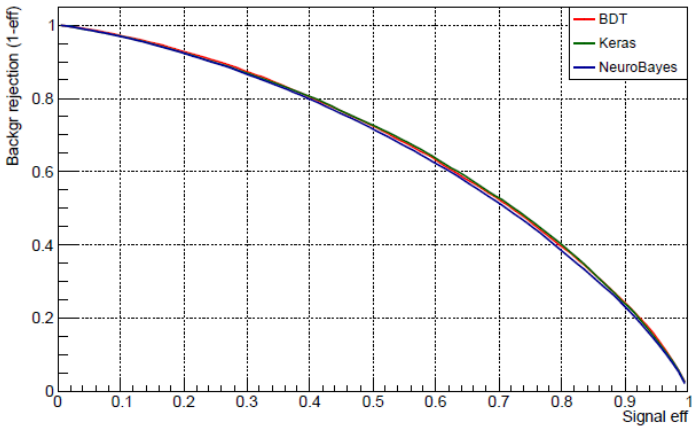


Figure 4.2: Comparison of ROC curves for various machine learning methods in the `1j1b` region. The "Keras" line represents a feed-forward neural network with multiple hidden layers [59].

4.2 Tensorflow & Keras

The tW dilepton adversarial neural network is built using Keras [61]. Keras is a popular machine learning API, interfacing with the open-source machine learning library TensorFlow 2 [62]. This allows for fast and intuitive building of efficient neural network structures while ensuring maximum performance. For this ANN, the functional API of Keras is used, a more flexible way to write complex models compared to the typically used sequential model [63].

4.2.1 Network setup

First the discriminator and adversary as described in 3.3.2 are built separately using the Keras functional API. Both networks are then combined into the full model. The following network topology is used:

Discriminator

- Input: variables as defined in 4.4.
- 5 hidden layers with 128 nodes each, ReLU activation
- Regularisation: Dropout (rate to be optimised), Batch normalisation
- Output: 1 node, sigmoid activation
- Optimiser: SGD, learning rate and momentum to be optimised

Adversary

- Input: Discriminator output as described in 3.3.2
- 4 hidden layers with 128 nodes each, ReLU activation
- Regularisation: Dropout (rate to be optimised), Batch normalisation
- Output: 1 node, sigmoid activation
- Optimiser: SGD, learning rate and momentum to be optimised

Losses for both networks use the built-in loss function `binary_crossentropy`. They are first defined separately

$$\mathcal{L}_f = \texttt{binary\_crossentropy}(\theta_f) \tag{4.1}$$

$$\mathcal{L}_r = \texttt{binary\_crossentropy}(\theta_r) \tag{4.2}$$

and then combined, with λ as an additional hyperparameter:

$$\mathcal{L}_r(\theta_f, \theta_r) = \mathcal{L}_f(\theta_f) - \lambda \mathcal{L}_r\left(\theta_r\right). \tag{4.3}$$

The definition of targets is shown in Table 4.1. The classifier sees tW as signal and $t\bar{t}$ as background, while the adversary considers any nominal samples as signal, while systematic samples are background.

In order to improve runtime and reduce the risk of overtraining or the network getting out of control an early stopping algorithm is used. If turned on, it stops the training if either the discriminator or combined validation losses stop improving.

	f	r
tW_{nom}	1	1
$t\bar{t}_{\text{nom}}$	0	1
tW_{sys}	1	0
$t\bar{t}_{\text{sys}}$	0	0

Table 4.1: Targets used in the adversarial neural network training

4.2.2 Network training

An overview of the training procedure is shown in Figure 4.3. In order to reach a good starting point, the classifier is first pretrained using just its loss $\mathcal{L}_f$, typically for 10 epochs. The adversary may also be pretrained, though the best results have been found with little adversary pretraining. After pretraining is completed, combined training is started. Training loops between the discriminator, using the combined loss $\mathcal{L}_f - \lambda\mathcal{L}_r$ and the adversarial with loss $\mathcal{L}_r$. Typically, for a single iteration, each network is trained for one epoch, though more advanced configurations are possible.

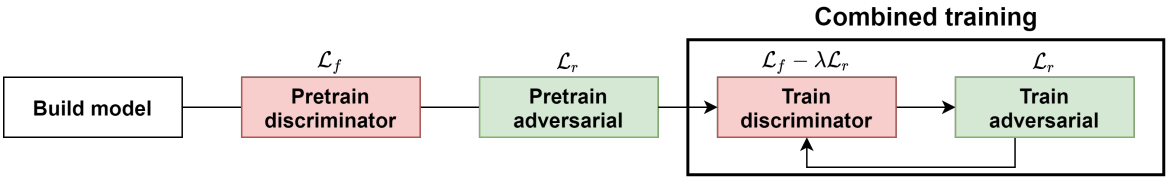


Figure 4.3: Training procedure for the ANN

Practically, this can be done by manually training each model for one epoch per iteration, switching which models weights are updated in between and implementing early stopping manually. Alternatively, a TensorFlow callback can be used to turn on and off weight updates between epochs. While the latter method is less flexible it is preferred as starting and stopping training manually causes significant overhead.

Note that "epoch" always refers to one training step of either network, while "iteration" refers to one full step of training *both* networks, including switching and early stopping. With the network setup used in this work, one iteration will always contain two epochs and consequently take twice the time to complete.

4.3 Monte Carlo samples

Monte Carlo samples are split into three campaigns. `MC16a` matches 2015 and 2016 data, `MC16d` matches 2017 data and `MC16e` matches 2018 data. For this work tW and $t\bar{t}$ samples from the Rel21 v29 production version are used. The full list of samples can be seen in Appendix A.

4.4 Variable selection

While creating a variable ranking specifically for this adversarial neural network might yield good results in the future, it currently requires computation time that is out of the scope of this work. Instead, the choice of variables is adopted from the previous work on this topic [60]. There, using simple kinematic variables was shown to be ineffective and quickly cause overtraining. Instead, a more complex set of kinematic variables was chosen, based on the most significant variables during BDT training [58].

These variables have been shown to work well with neural networks too, so they are an obvious choice. The list of variables in the `1j1b` and `2j2b` regions is shown in Table 4.2, with the following definitions:

- $p_{\mathrm{T}}^{\mathrm{sys}}(o_1, \ldots, o_n)$, the magnitude of the vector sum of transverse momenta;
- $m(o_1, \ldots, o_n)$, the invariant mass of the system;
- $m_{\mathrm{T}}(o_1, \ldots, o_n)$, the transverse mass of the system;
- $\Delta p_{\mathrm{T}}(s_1, s_2)$, the p_{T} difference between the systems;
- $\Delta R(s_1, s_2)$, the distance between the systems in $\phi - \eta$ space;

where $o_1, \ldots, o_n$ stands for one of the objects $\ell_1, \ell_2, j_1, j_2, E_{\mathrm{T}}^{\mathrm{miss}}$ and s_1 and s_2 stand for complex systems of objects.

`1j1b`	`2j2b`
$m(\ell_1 j_1)$	$m(\ell_1 j_1)$
$m(\ell_2 j_1)$	$m(\ell_1 j_2)$
$p_{\mathrm{T}}^{\mathrm{sys}}(\ell_1 \ell_2 j_1 E_{\mathrm{T}}^{\mathrm{miss}})$	$m(\ell_2 j_2)$
$p_{\mathrm{T}}^{\mathrm{sys}}(j_1 E_{\mathrm{T}}^{\mathrm{miss}})$	$m(\ell_2 j_1)$
$p_{\mathrm{T}}^{\mathrm{sys}}(\ell_1 \ell_2)$	$p_{\mathrm{T}}^{\mathrm{sys}}(j_1 j_2)$
$C(\ell_1 \ell_2)$	$p_{\mathrm{T}}(j_2)$
$\Delta p_{\mathrm{T}}(\ell_1, \ell_2)$	$\Delta R(j_1, j_2)$
$m_{\mathrm{T}}(j_1 E_{\mathrm{T}}^{\mathrm{miss}})$	
$m(\ell_2 j_1 E_{\mathrm{T}}^{\mathrm{miss}})$	
$p_T(j_{\mathrm{S}})$	

Table 4.2: Variables used for training in the `1j1b` and `2j2b` regions respectively, taken from [58].

4.5 Performance metrics

Judging the performance of the ANN is more complex than for a simple classifier network. Consequently, multiple approaches are taken:

ROC curve / AUC Plotting the receiver operator characteristic (ROC) curve as well as calculating the area under the curve (AUC) is a good metric to judge the performance of a binary classifier network. It does however not give any insights on the effect of the adversary, which is why it is of

limited usefulness here. The ROC curve is used to ensure correct training of the classifier and avoid its performance from falling too much below the pure classifier network approach due to the effect of the adversary. It is also used to identify any overtraining in the classifier, which would be indicated by the ROC curves of training and test sample not agreeing.

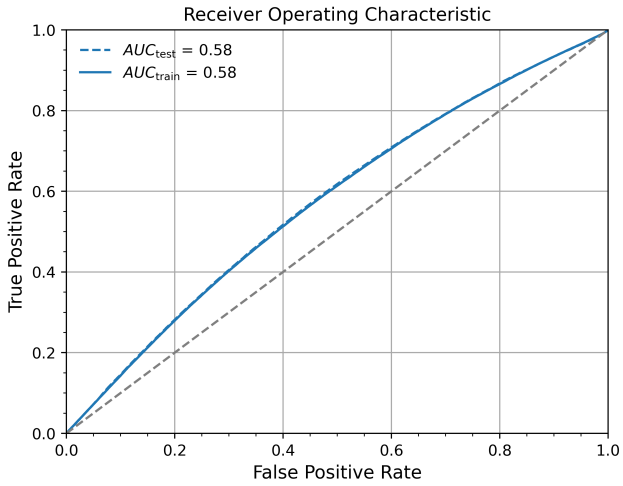


Figure 4.4: Example of the ROC curve plot. Good training is indicated by a smooth curve over the diagonal. Overtraining is indicated, if the solid train and dotted test curves do not agree.

Loss curves Probably the most important metric is the loss curve plot. Losses of the discriminator, adversary and combined networks are plotted together over the duration of the training. If the network is training correctly the discriminator and combined network loss curves should steadily fall, while the performance of the adversary worsens over time. If the networks do not balance out correctly it should become immediately obvious when looking at this plot. Additionally, by plotting losses of both the training and test sample, overtraining can be identified. An example of a loss curve plot for this network is shown in Figure 4.5.

Systematic impact/separation The ultimate goal of training this adversary neural network is to reduce the impact of systematic uncertainties on the classifier training. In order to judge this, the distribution of all four samples is plotted as a separation histogram. Additionally, the ratio of systematic to nominal samples in each bin is plotted. This ratio should quickly indicate if nominal and systematic samples are treated more similarly by the network. An example of this plot is shown in Figure 4.6.

4.6 Hyperparameter optimisation

With a network as complex as this ANN hyperparameter optimisation is a difficult task. Due to the nature of duelling networks it is much more sensitive to small changes as well as requiring more than double the number of parameters to be optimised.

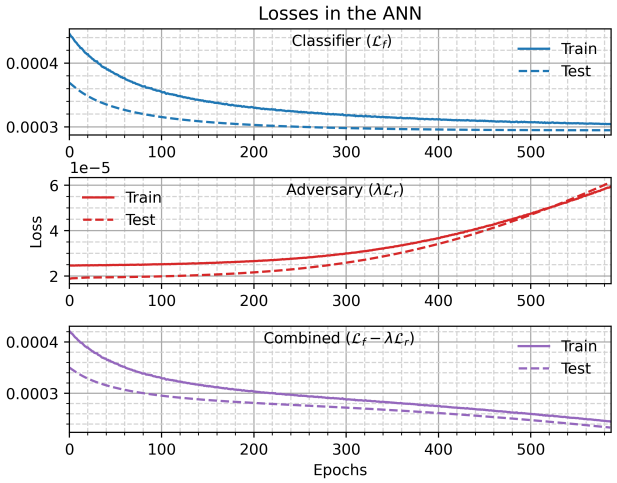


Figure 4.5: Example of loss curves in the ANN. The top curve shows the loss of the classifier, the middle curve plots the loss of the adversary. The bottom curve shows the loss of the combined network with $\mathcal{L}_f - \lambda\mathcal{L}_r$. In all cases the continuous curve shows the loss of the training sample, while the dotted curve shows the loss of the test sample. Some separation between these two is expected as dropout is only applied during the training, not during testing.

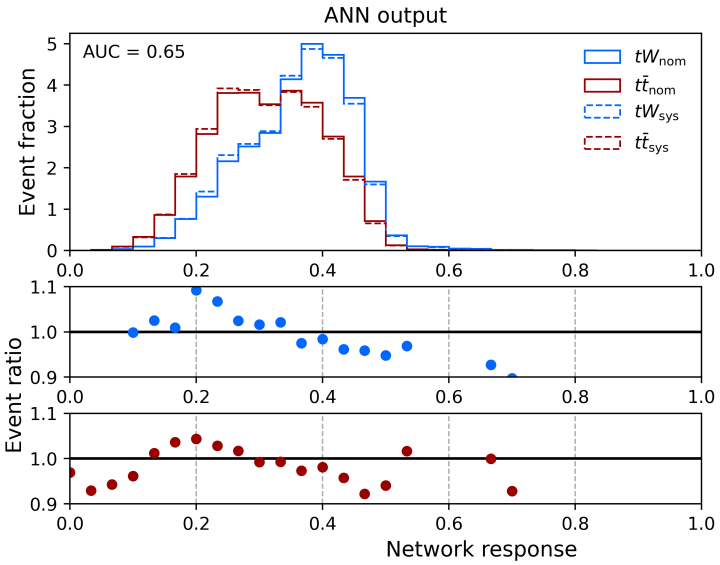


Figure 4.6: Example of a plot showing network response sample as well as the ratio of systematic and nominal sample below. If the impact of the systematic on training is reduced, the points of the ratio plot should be closer to 1 for bins with sufficient statistics.

To simplify the optimisation process somewhat, the following strategy is used: First, the adversarial is switched off by setting λ to 0 and only the discriminator network is trained. This way, the discriminator can be individually optimised to maximise its performance. Then, the adversary network is turned on and its parameters are tuned in order to achieve a situation where both networks are learning properly. During adversary optimisation, classifier parameters are also adjusted, as its optimisation only serves as a reasonable starting point of overall optimisation of the model.

In the following sections, the optimisation strategy for each parameter is described. The final optimised values are shown in Chapter 6.

Adversary behaviour The behaviour of the adversary falls into one of two categories. If it is too weak, no training of the adversary happens at all and the ANN acts like a regular classifier. Alternatively, if the adversary is capable of training based on the hyperparameter settings, it will always eventually overwhelm the classifier and invalidate the training. A behaviour "in between" those two extremes could not be found, so the best results were achieved by carefully tuning values that allowed the classifier to train for as long as possible before the adversary overtakes it and using early stopping. This way training is aborted at the optimal point, before the classifier performance drops off.

4.6.1 Random seed

Usually, when training neural networks, the random seed determining initial states of TensorFlow and related libraries does not have to be fixed. Only for the process of splitting the samples into training and validation samples a seed is necessary to ensure samples and their weights are split the same way.

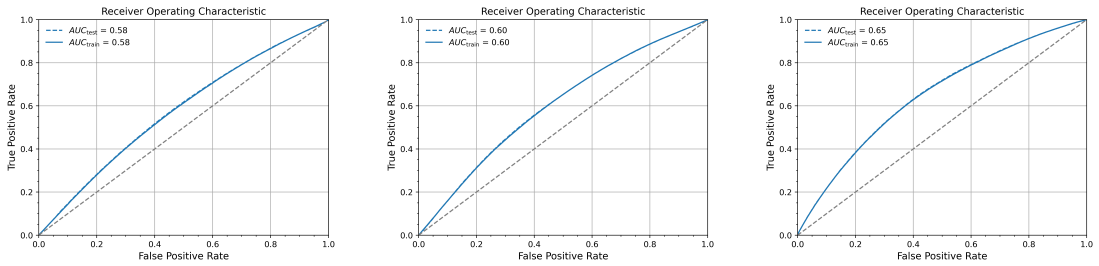
During this work, however, it has become clear that the ANN can be very unstable with respect to its initial state and other randomness in the training. Training multiple times with the exact same settings can yield vastly different results as the random seed is different in each training run, especially in the `1j1b` region with PS systematic. Due to this, the random seed is fixed for each region through the entire optimisation process. While this is generally not recommended, it is the only way to allow the optimisation process to go forward in the more unstable regions. As more stable states are reached, it might be possible again to remove this restriction and test for general applicability of results, regardless of the small variations introduced by a non-fixed random seed.

4.6.2 Nodes & layers

It has been shown that 128 nodes and 4–5 hidden layers work well [60]. A lower number of nodes and layers is tested, however, as Figure 4.7 shows, this only reduces network quality. Higher numbers do not bring improved results, so the value of 128 nodes is kept, as well as five layers for the classifier and four layers for the adversary.

4.6.3 Initialisers

Many possible initialisers are implemented in Keras. The classic `glorot_normal` initialiser, Keras' name for the Xavier normal initialiser introduced in Section 3.2.2, seems to work well. Other initialisers are tested, however they either have no noticeable influence on the training or cause the training to fail altogether.



(a) (b) (c)

Figure 4.7: Network training with the DR/DS systematic in the `1j1b` region with various number of nodes per layer in the classifier: (a) 32 nodes, (b) 64 nodes, (c) 128 nodes

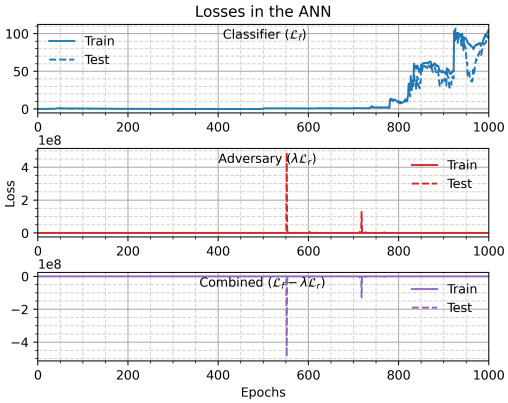


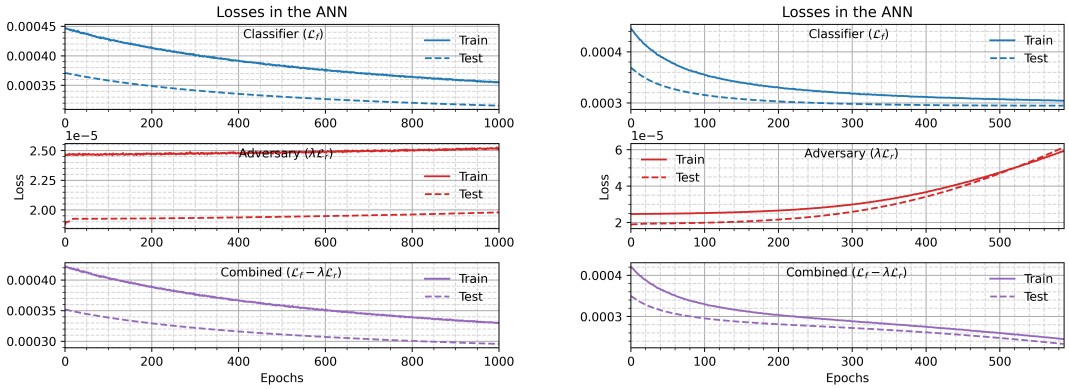
Figure 4.8: Losses using the *Adam* optimiser. In this case, the training fails entirely.

4.6.4 Optimiser

The classic *SGD* optimiser is used as default. Test runs with the more advanced *AdaGrad* and *Adam* optimisers show that the network seems to be too unstable to deal with these algorithms, so the default is kept. *Adam* may provide better results if run with completely different hyperparameters, however, even after some tuning it was not possible to find a stable configuration. Figure 4.8 shows an example of the network attempting to train using the *Adam* optimiser.

Learning rate

Learning rate has to be tuned very carefully for this adversarial neural network, as even small changes could cause training to shift out of balance. A large spaces of possible values is searched for each region. Every dataset works best with a different learning rate. In general it can be observed that very low learning rates will severely slow down training for no apparent benefit. This behaviour is shown in Figure 4.9.



(a) (b)

Figure 4.9: ANN training for different learning rates: (a) $\ell = 0.02$, (b) $\ell = 0.2$.

Momentum

A high momentum value should be able to speed up training considerably, unless obvious issues arise in training. Training using several momenta with the DR/DS systematic in the `1j1b` region is shown in Figure 4.10. No problems arise when increasing the momentum so a value of 0.8 is chosen.

Activation

Several options exist for the activation function for non-output layers. Three functions built into Keras are tested: `relu`, `elu` and `sigmoid`. The most consistent results can be achieved using `relu`, while the other two options tend to act less predictably. They can sometimes match the performance of `relu`, but will often cause training to not function properly or perform worse. Since no configurations are found where `elu` or `sigmoid` performs noticeably better, the more stable `relu` option is chosen. An example of `elu` and `sigmoid` not training properly, despite identical settings, can be seen in Figure 4.11

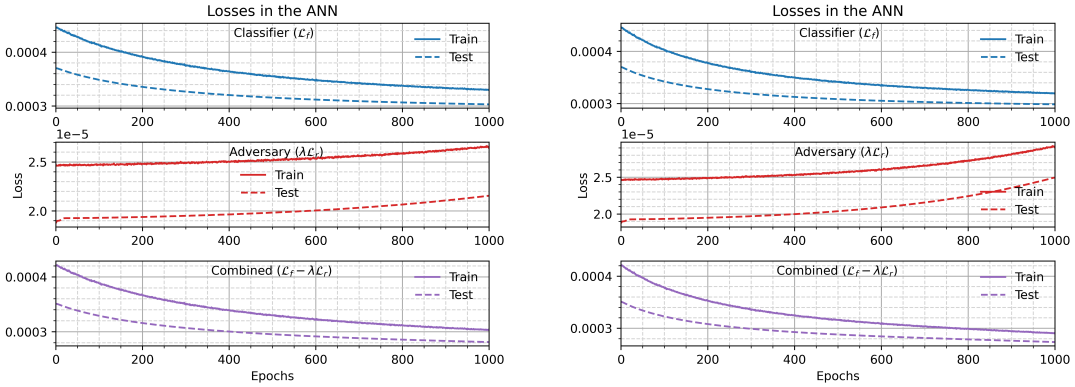
4.6.5 Regularisation

Batch normalisation

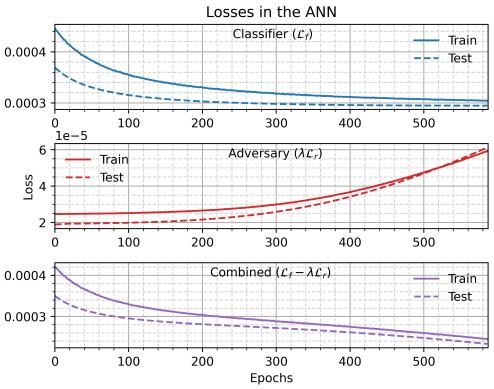
This network is quite deep and complex, so batch normalisation could help stabilise the network and make it slightly less sensitive to initialisation. The network is tested with batch normalisation on and off. As can be seen in Figure 4.12, performance is significantly improved with batch normalisation turned on.

Dropout

Dropout is tested for rates between 0% and 80% for both networks as well as in the input layer. Relatively low rates of dropout work well at improving runtime without reducing performance. For high rates of dropout the network tends to collapse quickly, as can be seen in Figure 4.13. Based on

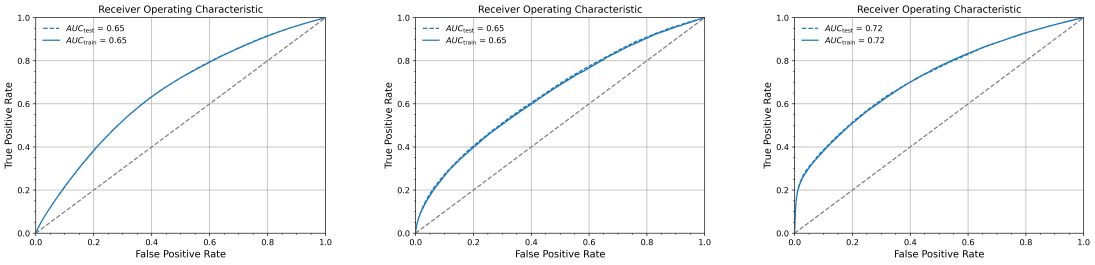


(a) (b)



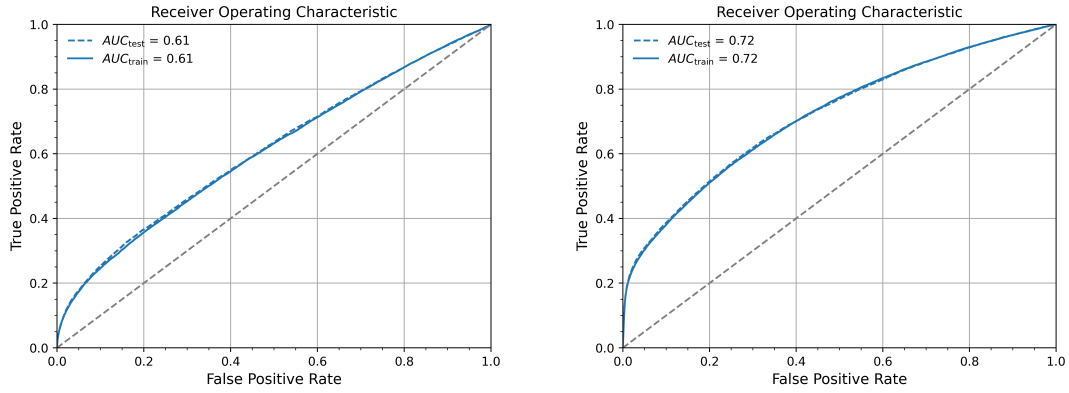
(c)

Figure 4.10: ANN training using several momenta: (a) $\alpha = 0.0$, (b) $\alpha = 0.3$, (c) $\alpha = 0.8$.



(a) (b) (c)

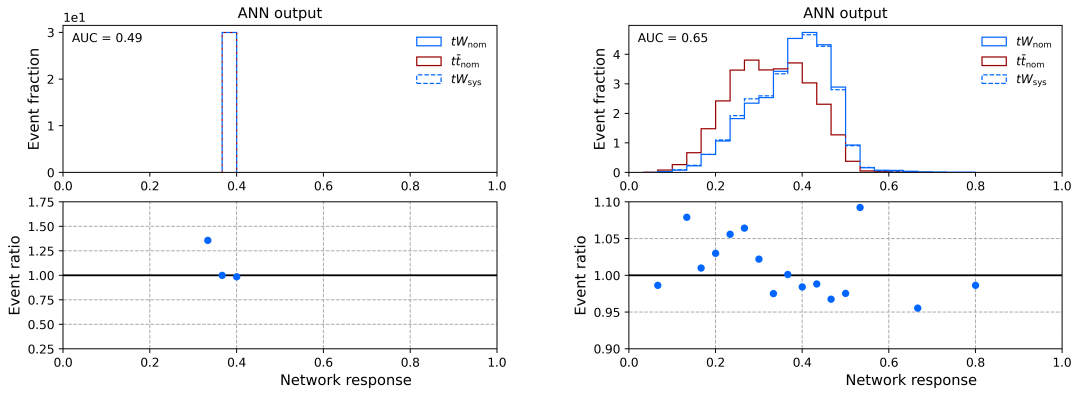
Figure 4.11: ROC curves with several activation functions: (a) `sigmoid`, (b) `elu`, (c) `relu`



(a) (b)

Figure 4.12: Training with the DR/DS systematic in the `2j2b` region with batch normalisation turned (a) off, (b) on.

these results a dropout of 30% is chosen for all hidden layers, as well as 10% for the classifier input layer.



(a) (b)

Figure 4.13: Network output training with DR/DS systematic in the `1j1b` region with (a) 80% dropout rate. The network collapses and produces no useful output. (b) 30% dropout rate, best performance is achieved.

Early stopping

In addition to dropout and batch normalisation, early stopping is used. There are several benefits that make this an important tool in this work. Firstly, monitoring the validation loss of the classifier network helps prevent it from overtraining. However, the more crucial use of early stopping in this work has been to prevent the adversary network from getting "out of control". Figure 4.14 shows an example of the training running for a full 1 000 epochs with high λ and no early stopping active. The classifier is quickly overwhelmed and the network behaves erratically. By monitoring the classifier

validation loss and stopping the training if it stops improving, this behaviour can be prevented and the training stopped at an appropriate point. In addition to this, early stopping is also useful to significantly save time during hyperparameter optimisation. For most parameters a large number of values has to be tested, many of which cause training to not work properly. With early stopping active on the validation losses of classifier and combined model, some of these runs will be prevented from training the full amount of epochs. This frees up the limited GPU slots quicker and hyperparameter optimisation can be performed more efficiently.

Early stopping is turned on for the classifier and combined validation losses, with a patience of 10 epochs.

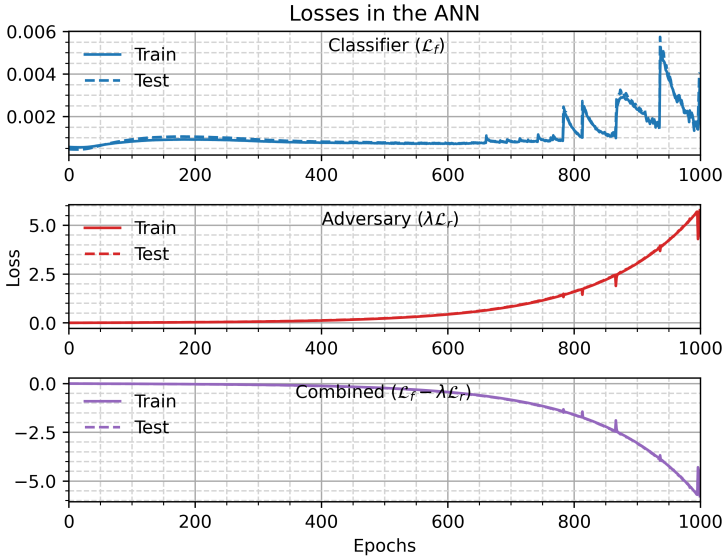
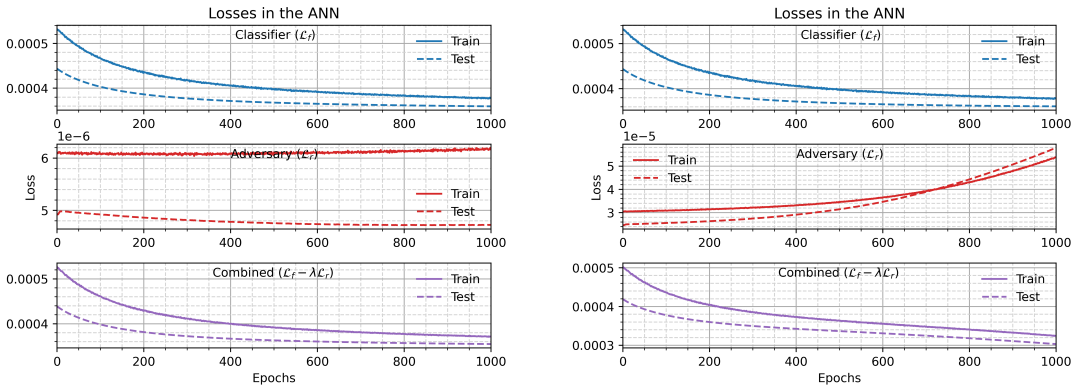


Figure 4.14: Example of a training run with $\lambda = 1$ and no early stopping applied. The adversary quickly dominates and training behaves erratically.

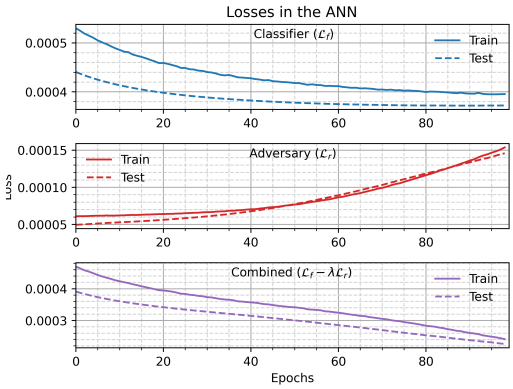
4.6.6 Lambda

λ is the most important hyperparameter in the network. It controls how much the loss of the adversary impacts the overall loss during training. It should be set to a value that allows the discriminator to train sufficiently, while the adversary gradually loses performance. A typical behaviour seems to be that the adversary slowly begins dominating the classifier, leading to an early stop due to lacking improvement of the classifier. Lack of early stopping will cause the network to get out of control and return nonsensical results.

To find the optimal value of λ, a large range of values is scanned and results checked for the desired behaviour described above. It appears to vary greatly between different datasets. Training behaviour for various values of λ is shown in Figure 4.15.



(a) (b)



(c)

Figure 4.15: Network behaviour for several values of λ: (a) $\lambda = 0.01$, the adversary is barely training; (b) $\lambda = 0.05$, both networks remain balanced for the full training duration; (c) $\lambda = 0.1$, the adversary quickly begins dominating and training is automatically stopped to prevent the discriminator loss curve from reversing.

CHAPTER 5

Runtime Optimisation

One of the most limiting factors in the previous work done on ANNs in the tW dilepton channel [60] was the poor training performance. Due to the complex model, large dataset and slow learning it can take many hours, sometimes more than a full day, to train the network once. This leads to great difficulties when trying to complete the delicate task of optimising the hyperparameters of this ANN.

A major goal of this work is to significantly improve the performance of the tW ANN and make hyperparameter optimisation a feasible task. This chapter will describe how this goal was reached. First, the differences between CPUs and GPUs are introduced in Section 5.1. Next, Section 5.2 lists the setup and performance metrics used to measure the performance improvement. In Section 5.3, 5.4 and 5.5 the process of heavily improving the performance is detailed. Lastly, Section 5.5 aims to measure how the ANN impacts performance compared to a regular classifier network.

5.1 Introduction to CPUs and GPUs

Neural network training is mostly composed of performing millions of matrix multiplication operations. If large enough batches sizes are chosen, many of these operations can be performed simultaneously without affecting the training.

Previously, regular processors were used to train neural networks. Large networks of processor nodes can run many training instances at once, allowing for easy hyperparameter optimisation. However, processors are inherently limited in their parallelity. Typical CPUs work with 4–32 logical cores, allowing only few calculations to be performed at once respectively [64]. These cores are optimised for general computing tasks that are rarely highly parallel. Additionally, a lot of transistors in a CPU are devoted to caching and control instead of processing, further limiting raw training performance.

A growing trend in deep learning has been the use of graphics processing units (GPUs) for training and inference. GPUs are optimised for rendering 3D scenes, a highly parallelisable task that involves performing millions of vector calculations. This has led to GPUs getting optimised for very high parallelity and containing often thousands of low performance cores. This feature can also be used in neural network training, potentially improving performance by orders of magnitude if the size of data batches can be increased significantly. Figure 5.1 exemplifies the difference in parallel data processing capability between CPUs and GPUs.

TensorFlow natively supports NIVIDA's CUDA and cuDNN architectures, so training can easily be done on an installed NVIDIA GPU if available [65].

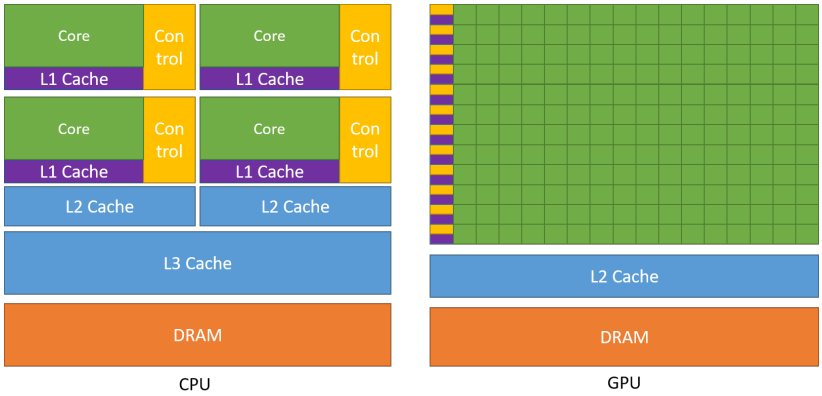


Figure 5.1: Architecture differences between a CPU and a GPU. The GPU contains many more cores and has more space dedicated to processing [64].

5.2 Performance metrics & setup

To accurately measure performance differences between different hardware, the following setup is used for every run:

- Region: `1j1b`
- Systematic: PS
- 100 epochs
- No early stopping

Initialisation and pretraining are performed for every run. Timing starts with the start of the main adversarial training using Python's built in `time.time()` function and ends after training is completed. From the measured time the epochs per hour are extrapolated.

5.2.1 Batch size

The amount of data processed in each training step, the *batch size*, can have a major impact on the performance of the network. if the batch size is too small, performance will be severely reduced as the network is not able to exploit available resources. However if the batch size is large enough to be a significant part of the overall data, training quality can be reduced. Additionally, when using a batch size that is too large to be handled by the hardware at once, diminishing returns will also cause training performance to decrease.

For each hardware configuration, multiple batch sizes are tested to find the optimal point of highest performance without running into memory constraints or reducing training quality.

5.3 Performance on CPU

Classically, networks are trained on regular computer processors. As mentioned in Section 5.1, parallelity is limited on CPUs, with typically only 4–32 calculations being run at once.

In this case, the network is trained on the worker nodes of the Bonn Analysis Facility (BAF2) computing cluster, powered mostly by Intel® Xeon® E5-2680 v4 processors, running at a base clock of 2.40 GHz. The raw performance of this CPU, if all 28 logical cores are used, peaks at about 1.3 TFLOPs[1]. Performance is measured with 8 and 16 logical cores. Note that due to the architecture of this computing cluster, actual runtime can vary by a significant amount, influenced by extraneous circumstances such as general load of the cluster, choice of node and many more. Due to this, performance shown here is only meant to be an approximation and not a true benchmark.

Performance over several batch sizes is shown in Figure 5.2. In all cases, performance seems to peak around a batch size of 4 096 to 16 384 events with diminishing returns above that. While there are differences between using 8 and 16 cores, they do appear to be rather small, suggesting that I/O or caching are the main bottleneck here. This test was performed with the standard installation of TensorFlow and other libraries. Intel suggests the use of an optimised version of TensorFlow on their CPU, however a quick test showed that there was no visible performance uplift. Since GPUs promise to be significantly faster, investigating CPU optimisation further does not seem to be necessary.

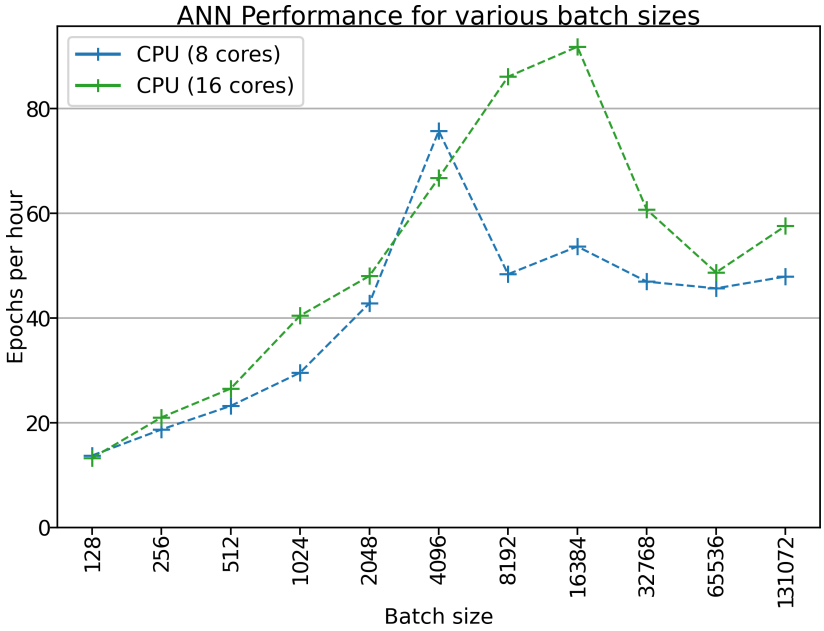


Figure 5.2: Performance of CPU with 8 and 16 cores allocated across several batch sizes

The performance achieved with higher batch sizes ranges between 75–100 epochs/hour, a 3–4x uplift from the previously used 512 batch size. At this speed, a full 1 000 epoch training run would take about 13 hours to complete. While that is a significant improvement, it still is likely too long to reasonably perform hyperparameter optimisation using CPU training.

[1] FLOPs: floating point operations per second, a common measure of raw computer performance

5.4 Performance on GPU

The use of graphics cards for training neural networks promises significant performance uplift that could make optimising this ANN a realistic task. For the following tests the newly installed GPU node in the BAF2 cluster is used. It is equipped with four NVIDIA GeForce® GTX 1080Ti graphics cards, delivering a raw performance of 11.3 TFLOPs each. Each card also has 11 GB of integrated memory which is enough to store even larger datasets. The node is powered by an Intel® Xeon® E5-2620 v4 processor, running at 2.10 GHz.

Performance is measured using the standard TensorFlow 2.1.0 GPU-optimised binaries, with the XLA compilation enabled. Batch sizes from 1024 up to the maximum of 262 144 are considered. Figure 5.3 shows that performance using these GPUs is significantly higher than on CPUs, peaking at almost 1 400 per hour. This means that a total training time well below one hour can be achieved. The sweet spot of batch size appears to be reached at 65 536, with another doubling only bringing a negligible improvement in performance.

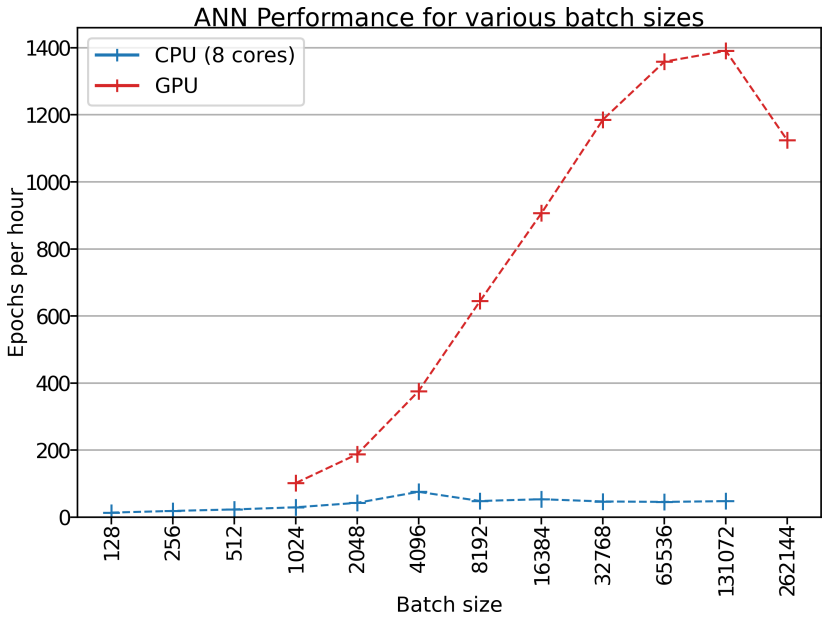


Figure 5.3: Performance of the ANN on CPU and GPU across multiple batch sizes

With this significant uplift in performance more effective optimisation should become viable, however at the current time availability of GPUs on the BAF2 cluster is still limited.

5.5 Miscellaneous improvements

Model switching via callbacks

Previously, training the different models was switched manually. For each iteration, one epoch was trained per model with weights being frozen in between. While this is simple to implement and allows

for great flexibility, it requires TensorFlow to restart the training with every step, adding overhead.

In order to remove this overhead, TensorFlow's callback system can be used. This allows the user to call functions during training depending on certain conditions. For the simplest case of training each network for only one epoch, a simple function can be used that flips the model to be trained on with each epoch.

Accelerated Linear Algebgra (XLA)

An important improvement to performance can be the TensorFlow library itself. Upgrading to TensorFlow 2.1 improved performance by several percent. Another small uplift in performance was brought by activating the Accelerated Linear Algebra (XLA) compilation mode. XLA can compile TensorFlow graphs to make better use of the specific hardware in the system, thereby improving its efficiency.

Newer versions of TensorFlow may further add to this performance improvement, as certain calculations are optimised more for specific sets of hardware.

5.6 Comparison with standard classifier network

Lastly, in order to gauge the true impact of the adversarial network setup on performance, it is compared to a standard classifier. In order to ensure a fair comparison, the same code is used for both tests.

For the pure classifier test the code is simply modified: The adversarial and combined models are not built and the training is changed to only train the single classifier model. Pretraining of the classifier is left unchanged, then it is trained for 100 epochs. Because the classifier model is updated only every other epoch during ANN combined training, the ANN is trained for 200 epochs instead and the "Classifier epochs per hour" metric is used. All other settings are left unchanged to ensure a fair comparison.

The results are shown in Figure 5.4. The raw classifier achieves about 3 400 epochs/hour, while the full ANN can only train the classifier model for 740 epochs/hour. This is a performance reduction of 4.6x if the ANN is used. Note that this does not account for differences in optimal hyperparameters in the Adversary Neural Network training.

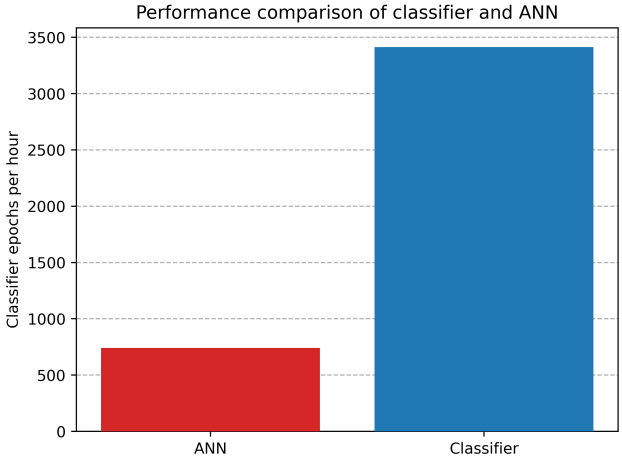


Figure 5.4: Performance comparison between pure classifier and ANN training.

CHAPTER 6

Results

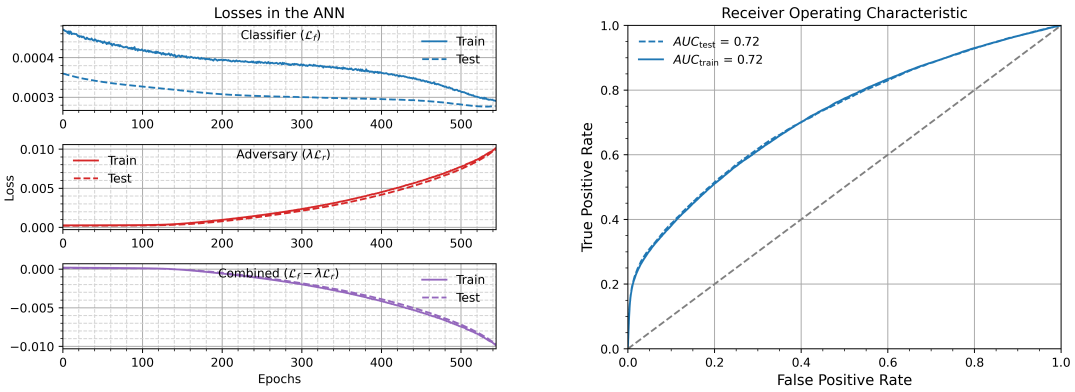
In this chapter the results obtained from optimising the adversarial neural network in the tW dilepton channel are presented. For each dataset, general behaviour of the network with optimised hyperparameters and discussed first. Next, the network response is analysed with respect to the goal of reducing impact of systematic uncertainties. Results from training with the DR/DS systematic in the `2j2b` and `1j1b` regions, respectively, are detailed in Sections 6.1 and 6.2. Results with the PS systematic in the `1j1b` region is shown in Section 6.3.

6.1 ANN training with DR/DS systematic in the `2j2b` region

The network appears to act relatively stable when using the DR/DS systematic in the `2j2b` region, likely due to the relatively high separation compared to other regions, as well as the strong separation of the more $t\bar{t}$-like DS sample from the nominal DR sample. Best training is achieved with a comparably low learning rate of $\ell = 0.05$ but high lambda $\lambda = 0.5$. The loss curves and ROC curve are shown in Figure 6.1. Both networks appear to be training as expected. The training is cut off around 550 epochs by early stopping to avoid a significant reduction in performance caused by a dominating adversary. The ROC curve shows decent performance of the discriminator training and no significant overtraining. Final hyperparameter values are shown in Table 6.1. Note that the optimiser settings and epoch in the discriminator and adversary columns refer to the values used in pretraining.

6.1.1 Systematics impact

Figure 6.2 shows the network response for each sample, with and without active adversary. Both signal and background are well behaved. The effect of the DS sample being more $t\bar{t}$-like is clearly visible in the ratio plot, as the event fraction of tW_{sys} to tW_{nom} is tilting heavily towards lower network responses. The shape of network response clearly changes when turning on the adversary, however the ratio plot does not show any improvement of systematics impact. The exact reason for this is unclear, further tuning in this region may improve this, but it is also possible that the topology of the `2j2b` region is not well suited for this particular approach.

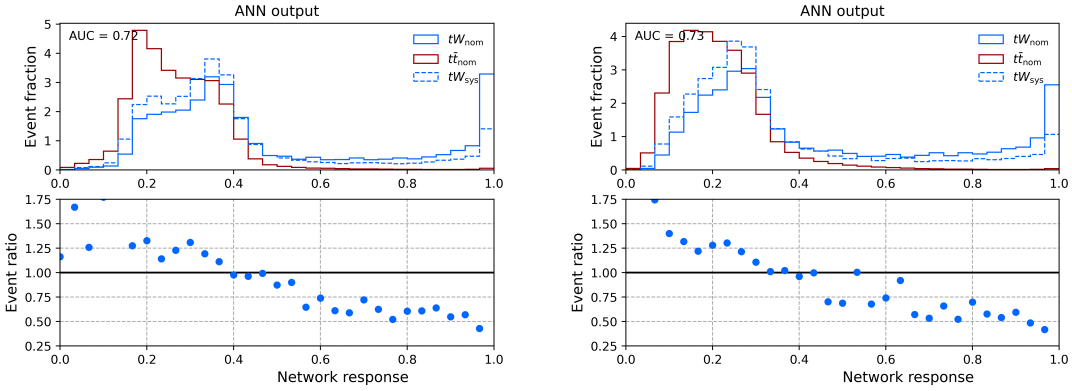


(a) (b)

Figure 6.1: (a) Loss curves, (b) ROC curve for the best training of the DR/DS systematic in the `2j2b` region

	Discriminator	Adversary	Combined
Layers	5	4	
Nodes	128	128	
Epochs	10	10	≈ 550
Optimiser	SGD	SGD	SGD
Learning rate	0.2	0.01	0.05
Momentum	0.8	0.8	0.8
Activation	ReLU	ReLU	ReLU
Input dropout	0.1		
Dropout	0.3	0.3	
λ	0.01		

Table 6.1: Final values for hyperparameters for the DR/DS systematic in the `2j2b` region

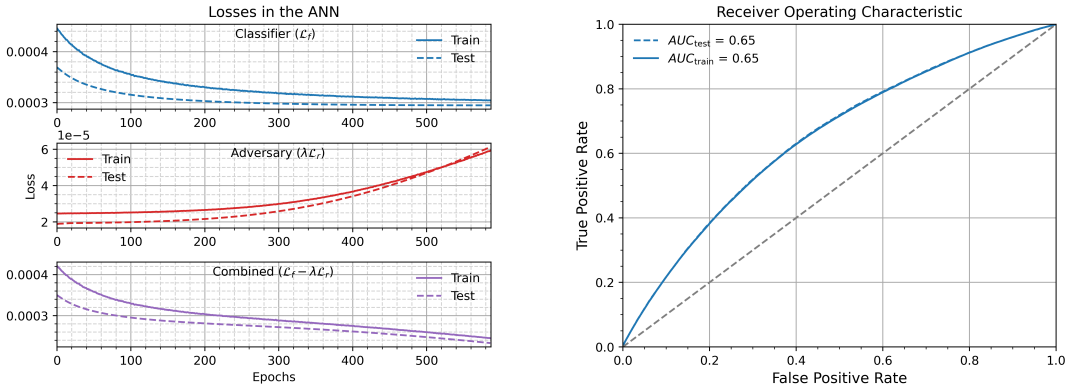


(a) (b)

Figure 6.2: Separation of tW and $t\bar{t}$ network response and ratio plot of nominal and systematic sample responses, training with the DR/DS systematic in the `2j2b` region: (a) with adversary, (b) without adversary

6.2 ANN training with DR/DS systematic in the 1j1b region

1j1b is the main signal region, so significantly more tW data is expected compared to the 2j2b region, while the amount of $t\bar{t}$ data should be lower. A consequence of this is that the impact of the DR/DS systematic in this region is much lower. It is not clear though how this would affect ANN training. During the optimisation process, it has been observed that this region is significantly less stable and hyperparameters are more difficult to optimise. A good configuration can be found for a learning rate of $\ell = 0.2$ and $\lambda = 0.05$ as well as additional parameters shown in Table 6.2. ROC and loss curves can be seen in Figure 6.3. All loss curves look very good and smooth for all networks. After about 600 epochs it can be seen that the early stopping algorithm stops the training due to the classifier validation loss no longer improving. This fairly long training time leads to a good AUC value for 0.65. The ROC curve is also smooth and shows no potential issues.



(a) (b)

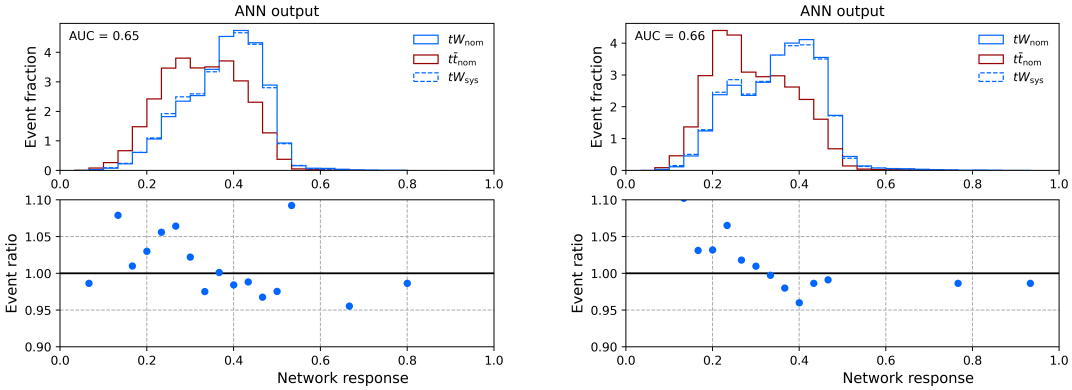
Figure 6.3: (a) Loss curves, (b) ROC curve for the best training of the DR/DS systematic in the 1j1b region

	Discriminator	Adversary	Combined
Layers	5	4	
Nodes	128	128	
Epochs	10	10	≈ 590
Optimiser	SGD	SGD	SGD
Learning rate	0.2	0.01	0.2
Momentum	0.8	0.8	0.8
Activation	ReLU	ReLU	ReLU
Input dropout	0.1		
Dropout	0.3	0.3	
λ	0.05		

Table 6.2: Final values for hyperparameters for the DR/DS systematic in the 1j1b region

6.2.1 Systematics impact

The separation of tW and $t\bar{t}$ samples including the tW_{sys} sample alongside with the ratio plot is shown in 6.4. A significant change of shape is visible and the AUC value worsens as the adversary is turned on. However, as the ratio plot shows, the ratio of tW_{sys} to tW_{nom} is visibly reduced. The total difference between bins $\sum_{\text{bins}} |tW_{\text{sys}} - tW_{\text{nom}}|$ is reduced by about 20%, suggesting that the impact of the DR/DS systematic uncertainty is reduced. It is unclear however if the reduction of AUC value compared to regular training is worth it for this amount of improvement.



(a) (b)

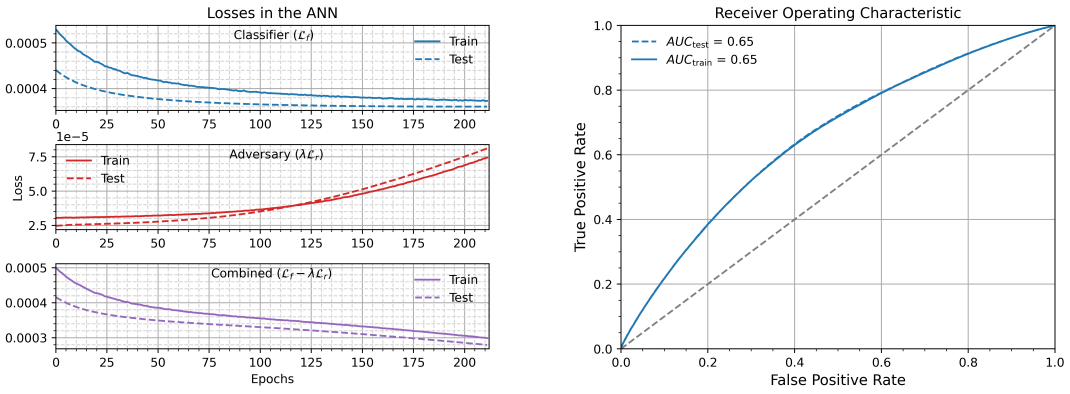
Figure 6.4: Separation of tW and $t\bar{t}$ network response and ratio plot of nominal and systematic responses in the `1j1b` region with DR/DS systematic: (a) with adversary, (b) without adversary

6.3 ANN training with PS systematic in the `1j1b` region

The PS systematic is interesting compared to DR/DS due to it affecting both the tW and $t\bar{t}$ samples. Network training behaves similarly to DR/DS in the `1j1b` region once a stable configuration of hyperparameters is found with $\ell = 0.05$ and $\lambda = 0.1$. ROC and loss curves for this configuration can be seen in Figure 6.5. All three loss curves behave as desired with the classifier learning well, while the adversary worsens over time. There may be a small amount of overtraining in the adversary, however it does not become problematic by the time the training is stopped around 210 epochs. The ROC curve also looks as expected and no overtraining in the classifier is visible.

6.3.1 Systematics impact

The separation and ratio plot is shown in Figure 6.6. The worse separation with the adversary turned on is clearly visible with the large $t\bar{t}$ peak around 0.2 missing completely. The ratio plot shows no improvement of the systematic with the $t\bar{t}$ sample, however for the tW sample the improvement in network response to nominal and systematic samples is clearly visible.

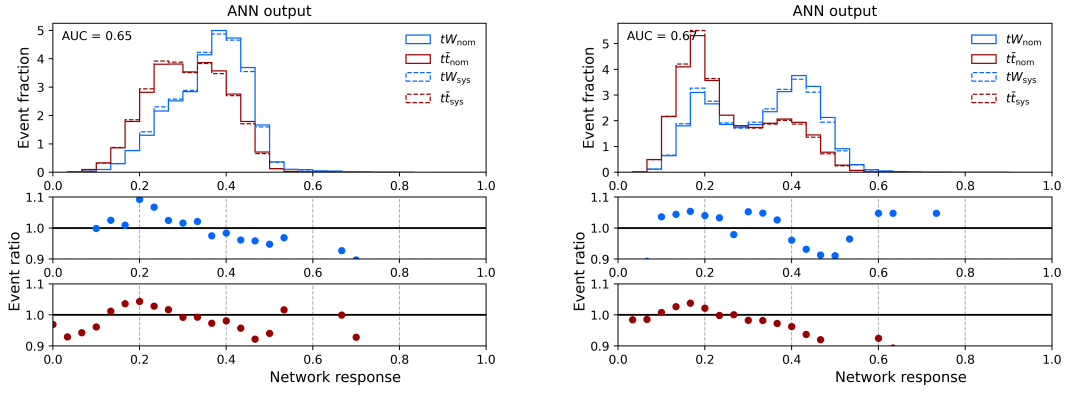


(a) (b)

Figure 6.5: (a) Loss curves, (b) ROC curve for the best training of the PS systematic in the 1j1b region

	Discriminator	Adversary	Combined
Layers	5	4	
Nodes	128	128	
Epochs	10	10	≈ 210
Optimiser	SGD	SGD	SGD
Learning rate	0.2	0.01	0.05
Momentum	0.8	0.8	0.8
Activation	ReLU	ReLU	ReLU
Input dropout	0.1		
Dropout	0.3	0.3	
λ	0.1		

Table 6.3: Final values for hyperparameters for the PS systematic in the 1j1b region



(a) (b)

Figure 6.6: Separation of tW and $t\bar{t}$ network response and ratio plot of nominal and systematic responses in the 1j1b region with PS systematic: (a) with adversary, (b) without adversary

CHAPTER 7

Conclusion

This work focused on improving on some major issues present with the adversarial neural network in the tW dilepton first tried in [60]. The issues included the very long training time and the inherent instability of the network. In addition to dealing with these issues, the network was tested with an additional region and systematic uncertainty.

The runtime was successfully decreased using several techniques. The implementation of the ANN was optimised and additional tools were used to get the maximum available performance and make efficient use of training steps. An optimal, increased batch size was found to make more use of hardware capabilities. The largest improvement was achieved by using a GPU for the training process and further optimising batch size for a peak performance of about 1 400 epochs/hour. This lowered the total training time from 20 hours to 20–40 minutes, despite using about three times as much training data. These massive improvements in runtime make further studying of this network reasonably possible, as the time and computing resources required to conduct many training runs is no longer crippling.

Training was performed for three setups: The DR/DS systematic uncertainty was trained in the `1j1b` and `2j2b` regions. Additionally, the parton showering and hadronisation systematic uncertainty, which applies to both signal and background samples, was trained in the `1j1b` region.

Concerns about the inherent instability of this network structure could not be dispelled. With the additional performance however, a somewhat stable configuration with desired behaviour of each network was found for all regions. A strong equilibrium between classifier and adversary networks could not be found. It appears that the adversary loss will always accelerate over time, so the use of early stopping is crucial to achieve a reasonable result. The classifier could consistently be observed to perform worse if the adversary was turned on. This is a behaviour expected as the classifier becomes more robust.

In the `1j1b` region a reduction in the difference of network response to nominal and systematic tW was found for both tested systematics. However, no difference was found for the tW sample in the `2j2b` region, as well as the $t\bar{t}$ sample when training with the parton showering systematic. Further work will however be necessary to judge whether these reductions in systematic dependency of the network output can outweigh the reduced overall separation between tW and $t\bar{t}$.

With the improved performance additional aspects of the network may be available to be tuned now, such as finding an optimal set of variables for this ANN. However, the limited availability of GPU resources still has to be considered.

This network remains an interesting technique that could improve the tW analysis in the future. As computing resources increase and particle physicists strive for more powerful machine learning methods, adversarial networks have the potential to become an important tool for advanced data analysis.

Bibliography

[1] P. O'Grady, *Thales of Miletus*, URL: https://www.iep.utm.edu/ (visited on 16/11/2020) (cit. on p. 1).

[2] R. S. Westfall, *Isaac Newton*, URL: https://www.britannica.com/biography/Isaac-Newton (visited on 16/11/2020) (cit. on p. 1).

[3] C. Domb, *James Clerk Maxwell*, URL: https://www.britannica.com/biography/James-Clerk-Maxwell (visited on 16/11/2020) (cit. on p. 1).

[4] *The Standard Model*, (2012), URL: http://cds.cern.ch/record/1997201 (cit. on pp. 1, 3).

[5] MissMJ, *Standard Model of Elementary Particles*, [Online; accessed October 2020], 2006, URL: https://commons.wikimedia.org/wiki/File:Standard_Model_of_Elementary_Particles.svg (cit. on p. 4).

[6] D. H. Perkins, *Introduction to High Energy Physics*, 4th ed., Addison-Wesley, 2000 (cit. on p. 4).

[7] M. T. et al, *Review of Particle Physics*, Phys. Rev. D **98** (3 2018) 030001, URL: https://link.aps.org/doi/10.1103/PhysRevD.98.030001 (cit. on pp. 5, 12, 13).

[8] P. W. Higgs, *Broken Symmetries and the Masses of Gauge Bosons*, Phys. Rev. Lett. **13** (16 1964) 508 (cit. on p. 5).

[9] *The Large Hadron Collider*, (2014), URL: http://cds.cern.ch/record/1998498 (cit. on p. 5).

[10] The ATLAS Collaboration, *The ATLAS Experiment at the CERN Large Hadron Collider*, Journal of Instrumentation **3** (2008) S08003 (cit. on pp. 6–9).

[11] The CMS Collaboration, *The CMS experiment at the CERN LHC*, Journal of Instrumentation **3** (2008) S08004 (cit. on p. 6).

[12] The ALICE Collaboration, *The ALICE experiment at the CERN LHC*, Journal of Instrumentation **3** (2008) S08002 (cit. on p. 6).

[13] The LHCb Collaboration, *The LHCb Detector at the LHC*, Journal of Instrumentation **3** (2008) S08005 (cit. on p. 6).

[14] J. Haffner, *The CERN accelerator complex. Complexe des accélérateurs du CERN*, (2013), General Photo, URL: http://cds.cern.ch/record/1621894 (cit. on p. 6).

[15] "LHC Guide", 2017, URL: http://cds.cern.ch/record/2255762 (cit. on p. 6).

[16] ATLAS Outreach,
"ATLAS Fact Sheet : To raise awareness of the ATLAS detector and collaboration on the LHC", 2010, URL: https://cds.cern.ch/record/1457044 (cit. on pp. 8, 10).

[17] C. Grupen and B. A. Shwartz, *Particle Detectors*, 2nd ed., Cambridge University Press, 2008 (cit. on p. 8).

[18] G. Ripellino, *The alignment of the ATLAS Inner Detector in Run 2*,
tech. rep. ATL-INDET-PROC-2016-003, CERN, 2016,
URL: https://cds.cern.ch/record/2213441 (cit. on p. 8).

[19] ATLAS Collaboration, ed.,
ATLAS detector and physics performance: Technical Design Report, 1,
Technical Design Report ATLAS, 1999, URL: https://cds.cern.ch/record/391176 (cit. on p. 9).

[20] A. Ruiz-Martinez and A. Collaboration, *The Run-2 ATLAS Trigger System*,
tech. rep. ATL-DAQ-PROC-2016-003, CERN, 2016,
URL: https://cds.cern.ch/record/2133909 (cit. on p. 10).

[21] J. Pequenao and P. Schaffner,
"How ATLAS detects particles: diagram of particle paths in the detector", 2013,
URL: https://cds.cern.ch/record/1505342 (cit. on p. 11).

[22] M. Aaboud et al., *Performance of the ATLAS Track Reconstruction Algorithms in Dense Environments in LHC Run 2*, Eur. Phys. J. C **77** (2017) 673, arXiv: 1704.07983 [hep-ex] (cit. on p. 10).

[23] M. Aaboud et al., *Reconstruction of primary vertices at the ATLAS experiment in Run 1 proton–proton collisions at the LHC*, Eur. Phys. J. C **77** (2017) 332,
arXiv: 1611.10235 [physics.ins-det] (cit. on p. 10).

[24] G. Aad et al.,
Topological cell clustering in the ATLAS calorimeters and its performance in LHC Run 1,
Eur. Phys. J. C **77** (2017) 490, arXiv: 1603.02934 [hep-ex] (cit. on pp. 10, 11).

[25] G. Aad et al., *Muon reconstruction performance of the ATLAS detector in proton–proton collision data at $\sqrt{s}$ =13 TeV*, Eur. Phys. J. C **76** (2016) 292, arXiv: 1603.05598 [hep-ex] (cit. on pp. 10, 11).

[26] M. Aaboud et al., *Electron reconstruction and identification in the ATLAS experiment using the 2015 and 2016 LHC proton-proton collision data at $\sqrt{s}$ = 13 TeV*,
Eur. Phys. J. C **79** (2019) 639, arXiv: 1902.04655 [physics.ins-det] (cit. on p. 10).

[27] M. Cacciari, G. P. Salam and G. Soyez, *The anti-k_t jet clustering algorithm*,
JHEP **04** (2008) 063, arXiv: 0802.1189 [hep-ph] (cit. on p. 11).

[28] *Pile-up subtraction and suppression for jets in ATLAS*, tech. rep. ATLAS-CONF-2013-083, CERN, 2013, URL: https://cds.cern.ch/record/1570994 (cit. on p. 11).

[29] *Optimisation of the ATLAS b-tagging performance for the 2016 LHC Run*, (2016) (cit. on p. 11).

[30] M. Aaboud et al., *Performance of missing transverse momentum reconstruction with the ATLAS detector using proton-proton collisions at $\sqrt{s} = 13$ TeV*, Eur. Phys. J. C **78** (2018) 903, arXiv: 1802.08168 [hep-ex] (cit. on p. 12).

[31] F. Abe et al., *Observation of Top Quark Production in pp Collisions with the CDF Detector at Fermilab*, Phys. Rev. Lett. **74** (1995) 2626, arXiv: hep-ex/9503002 [hep-ex] (cit. on p. 12).

[32] S. Abachi et al., *Observation of the top quark*, Phys. Rev. Lett. **74** (1995) 2632, arXiv: hep-ex/9503003 [hep-ex] (cit. on p. 12).

[33] D. Collaboration, *Useful Diagrams of Top Signals and Backgrounds*, URL: https://www-d0.fnal.gov/Run2Physics/top/top_public_web_pages/top_feynman_diagrams.html (visited on 14/10/2020) (cit. on p. 12).

[34] *NLO single-top channel cross sections*, URL: https://twiki.cern.ch/twiki/bin/view/LHCPhysics/SingleTopRefXsec (cit. on p. 13).

[35] M. H. Seymour and M. Marx, "Monte Carlo Event Generators", *69th Scottish Universities Summer School in Physics: LHC Physics*, 2013 287, arXiv: 1304.6677 [hep-ph] (cit. on p. 15).

[36] S. Höche, *Introduction to parton-shower event generators*, 2015, arXiv: 1411.4085 [hep-ph] (cit. on p. 16).

[37] G. Aad et al., *The ATLAS Simulation Infrastructure*, The European Physical Journal C **70** (2010) 823, ISSN: 1434-6052, URL: http://dx.doi.org/10.1140/epjc/s10052-010-1429-9 (cit. on p. 15).

[38] C. Oleari, *The POWHEG BOX*, Nuclear Physics B - Proceedings Supplements **205-206** (2010) 36, ISSN: 0920-5632, URL: http://dx.doi.org/10.1016/j.nuclphysbps.2010.08.016 (cit. on p. 16).

[39] T. Sjöstrand, *The Pythia event generator: Past, present and future*, Computer Physics Communications **246** (2020) 106910, ISSN: 0010-4655, URL: http://dx.doi.org/10.1016/j.cpc.2019.106910 (cit. on p. 16).

[40] S. Agostinelli et al., *Geant4—a simulation toolkit*, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **506** (2003) 250, ISSN: 0168-9002 (cit. on p. 16).

[41] C. D. White et al., *Isolating Wt production at the LHC*, JHEP **11** (2009) 074, arXiv: 0908.0631 [hep-ph] (cit. on p. 16).

[42] *The Herwig Event Generator*, URL: https://herwig.hepforge.org/ (cit. on p. 17).

[43] P. A. Freiberger and M. R. Swaine, *ENIAC*, URL: https://www.britannica.com/technology/ENIAC (visited on 23/10/2020) (cit. on p. 19).

[44] K. Albertsson et al., *Machine Learning in High Energy Physics Community White Paper*, J. Phys. Conf. Ser. **1085** (2018) 022008, arXiv: 1807.02876 [physics.comp-ph] (cit. on p. 19).

[45] D.-A. Clevert, T. Unterthiner and S. Hochreiter,
Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs), 2016,
arXiv: 1511.07289 [cs.LG] (cit. on p. 22).

[46] I. Sutskever et al., "On the importance of initialization and momentum in deep learning",
vol. 28, Proceeding of Machine Learning Research 3, PMLR, 2013 1139 (cit. on p. 24).

[47] D. P. Kingma and J. Ba, *Adam: A Method for Stochastic Optimization*, 2017,
arXiv: 1412.6980 [cs.LG] (cit. on p. 24).

[48] X. Glorot and Y. Bengio,
Understanding the difficulty of training deep feedforward neural networks,
Journal of Machine Learning Research - Proceedings Track **9** (2010) 249 (cit. on p. 25).

[49] Chabacano, *Overfitting*, [Online; accessed October 2020], 2008,
URL: https://commons.wikimedia.org/wiki/File:Overfitting.svg (cit. on p. 25).

[50] N. Srivastava et al., *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*,
Journal of Machine Learning Research **15** (2014) 1929,
URL: http://jmlr.org/papers/v15/srivastava14a.html (cit. on p. 26).

[51] S. Ioffe and C. Szegedy, *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*, (2015), arXiv: 1502.03167 [cs.LG] (cit. on p. 26).

[52] *tf.keras.callbacks.EarlyStopping*, URL: https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/EarlyStopping
(visited on 03/11/2020) (cit. on p. 27).

[53] *Early stopping*, URL: https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-deep-learning-tips-and-tricks (visited on 15/11/2020)
(cit. on p. 27).

[54] T. Fawcett, *Introduction to ROC analysis*, Pattern Recognition Letters **27** (2006) 861
(cit. on p. 28).

[55] I. Goodfellow et al., "Generative Adversarial Nets",
Advances in Neural Information Processing Systems 27, ed. by Z. Ghahramani et al.,
Curran Associates, Inc., 2014 2672 (cit. on p. 28).

[56] G. Louppe, M. Kagan and K. Cranmer, "Learning to Pivot with Adversarial Networks",
Advances in Neural Information Processing Systems 30, ed. by I. Guyon et al.,
Curran Associates, Inc., 2017 981 (cit. on p. 29).

[57] T. Chen, *Introduction to Boosted Trees* (cit. on p. 31).

[58] K. D. Finelli et al., *Measurement of the cross-section for the production of a W boson in association with a top quark at 13TeV*, tech. rep. ATL-COM-PHYS-2019-222, CERN, 2019,
URL: https://cds.cern.ch/record/2667560 (cit. on pp. 32, 35).

[59] N. Boeing, *Comparison of Multivariate Techniques in the Wt Single Top-Quark Production Channel at ATLAS*, 2017 (cit. on p. 32).

[60] C. Kirfel, *Hyperparameter Optimisation of an Adversarial Neural Network in the tW channel at 13 TeV with ATLAS*, 2019 (cit. on pp. 31, 35, 38, 45, 57).

[61] F. Chollet et al., *Keras*, https://keras.io, 2015 (cit. on p. 33).

[62] Martin Abadi et al., *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*, 2015, URL: https://www.tensorflow.org/ (cit. on p. 33).

[63] F. Chollet et al., *Keras Functional API*, URL: https://keras.io/guides/functional_api/ (cit. on p. 33).

[64] nVidia, *CUDA Toolkit Documentation*, [Online; accessed November 2020], URL: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html (cit. on pp. 45, 46).

[65] *GPU Support*, [Online; accessed November 2020], URL: https://www.tensorflow.org/install/gpu (cit. on p. 45).

APPENDIX A

Datasets

The full set of datasets is listed here.

user.ddavis.mc16_13TeV.410646.PhPy8EG_Wt_DR_t.SGTOP1.e6552_a875_r9364_p3832.ll.v29.20
user.ddavis.mc16_13TeV.410646.PhPy8EG_Wt_DR_t.SGTOP1.e6552_a875_r10724_p3832.ll.v29.20
user.ddavis.mc16_13TeV.410646.PhPy8EG_Wt_DR_t.SGTOP1.e6552_a875_r10201_p3832.ll.v29.20
user.ddavis.mc16_13TeV.410647.PhPy8EG_Wt_DR_tbar.SGTOP1.e6552_a875_r9364_p3832.ll.v29.20
user.ddavis.mc16_13TeV.410647.PhPy8EG_Wt_DR_tbar.SGTOP1.e6552_a875_r10724_p3832.ll.v29.20
user.ddavis.mc16_13TeV.410647.PhPy8EG_Wt_DR_tbar.SGTOP1.e6552_a875_r10201_p3832.ll.v29.20

Table A.1: List of datasets used for the tW_{nom} sample

user.ddavis.mc16_13TeV.410656.PhPy8EG_Wt_DS_2l_t.SGTOP1.e6615_s3126_r9364_p3832.ll.v29.0
user.dfrizzel.mc16_13TeV.410656.PhPy8EG_Wt_DS_2l_t.SGTOP1.e6615_s3126_r10201_p3832.ll.v29.0
user.ddavis.mc16_13TeV.410656.PhPy8EG_Wt_DS_2l_t.SGTOP1.e6615_s3126_r10724_p3629.ll.v29.SP09
user.ddavis.mc16_13TeV.410657.PhPy8EG_Wt_DS_2l_tbar.SGTOP1.e6615_s3126_r9364_p3832.ll.v29.0
user.ddavis.mc16_13TeV.410657.PhPy8EG_Wt_DS_2l_tbar.SGTOP1.e6615_s3126_r10201_p3832.ll.v29.1
user.ddavis.mc16_13TeV.410657.PhPy8EG_Wt_DS_2l_tbar.SGTOP1.e6615_s3126_r10724_p3832.ll.v29.1

Table A.2: List of datasets used for the tW_{sys} (DR/DS) sample

user.ddavis.mc16_13TeV.411038.PhH7EG_Wt_DR_2l_t.SGTOP1.e6702_a875_r9364_p3832.ll.v29.20
user.ddavis.mc16_13TeV.411038.PhH7EG_Wt_DR_2l_t.SGTOP1.e6702_a875_r10724_p3832.ll.v29.20
user.ddavis.mc16_13TeV.411038.PhH7EG_Wt_DR_2l_t.SGTOP1.e6702_a875_r10201_p3832.ll.v29.20
user.ddavis.mc16_13TeV.411039.PhH7EG_Wt_DR_2l_tbar.SGTOP1.e6702_a875_r9364_p3832.ll.v29.20
user.ddavis.mc16_13TeV.411039.PhH7EG_Wt_DR_2l_tbar.SGTOP1.e6702_a875_r10724_p3832.ll.v29.20
user.ddavis.mc16_13TeV.411039.PhH7EG_Wt_DR_2l_tbar.SGTOP1.e6702_a875_r10201_p3832.ll.v29.20

Table A.3: List of datasets used for the tW_{sys} (PS) sample

user.ddavis.mc16_13TeV.410472.PhPy8EG_ttbar_hdamp258p75_2l.SGTOP1.e6348_a875_r9364_p3832.ll.v29.20
user.ddavis.mc16_13TeV.410472.PhPy8EG_ttbar_hdamp258p75_2l.SGTOP1.e6348_a875_r10201_p3832.ll.v29.20
user.ddavis.mc16_13TeV.410472.PhPy8EG_ttbar_hdamp258p75_2l.SGTOP1.e6348_a875_r10724_p3832.ll.v29.20

Table A.4: List of datasets used for the $t\bar{t}_{\text{nom}}$ sample

user.ddavis.mc16_13TeV.410558.PhH7EG_ttbar_hdamp258p75_2l.SGTOP1.e6366_a875_r9364_p3832.ll.v29.20
user.ddavis.mc16_13TeV.410558.PhH7EG_ttbar_hdamp258p75_2l.SGTOP1.e6572_a875_r10201_p3531.ll.v29.20
user.ddavis.mc16_13TeV.410558.PhH7EG_ttbar_hdamp258p75_2l.SGTOP1.e6366_a875_r10724_p3832.ll.v29.20

Table A.5: List of datasets used for the $t\bar{t}_{\text{sys}}$ (PS) sample

List of Figures

List of Tables