Hyperparameter Optimisation of an Adversarial Neural Network in the tW channel at 13 TeV with ATLAS

Christian Kirfel

Masterarbeit in Physik
angefertigt im Physikalischen Institut

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

April 2019

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

Bonn, ...
Date Signature

1. Gutachter: Prof. Dr. Ian C. Brock
2. Gutachter: Prof. Dr. Klaus Desch

Contents

CHAPTER 1

Introduction

Der Weg ist das Ziel

One of my most influencing childhood experiences is Lego® [1]. Lego is a system of interlocking plastic bricks allowing to build a nearly unlimited variety of structures ranging from medieval castles to models of particle detectors [2]. Although the set of differently shaped bricks available to the player has greatly increased over the years, I still see Lego's appeal and charm in the originally simple concept of a limited set of elemental pieces, enabling an inspired user to construct almost anything he or she could think of.

This fascination for an elegant concept of elementary pieces that form everything has stayed with me through my life and education. I found it again in the Standard Model of Particle Physics (SMPP) [3]. The SMPP summarizes the set of elementary particles currently known and their interactions. They represent all the tools necessary to explain our world at least to a great extent. Carried by this inspiration I found my way into high energy collider physics and eventually, with this thesis, had the great opportunity of working with the ATLAS collaboration and taking part in an experiment extending the frontiers of modern physics.

When it comes to particle physics research, a conventional approach is to collide common particles at energies high enough to create new particles. Additionally, in order to filter collisions for event of interest, one needs to have a high rate of events to generate a sufficient sample size. It suggests itself to apply machine learning techniques on the task of filtering events at collider experiments. In this work, an adversarial neural network [32] is introduced. It adds the promise of minimising the sensitivity of a selection model for systematic uncertainties to the network's classification task.

At first, an introduction to the state of the art for particle physics is given. The SMPP is introduced and further information on physics incorporating top-quarks is presented thereby motivating the difficulties in separating the Standard Model process from its background and reducing its systematic uncertainties. Secondly the Large Hadron Collider and the ATLAS detector are introduced to explain how events are generated and reconstructed in the detector and how simulations for the experiment are generated. Then the concept of machine learning is described with a focus on neural networks and adversarial neural networks including their

hyper-parameters. Finally, the setup and training of a classifying neural network is described and then used as basis for the setup of an adversarial neural network. The results using an adversarial neural network are discussed based on the promises it might offer for an analysis highly dependent on systematic uncertainties.

Acknowledgements

I would like to thank ...

You should probably use `\chapter*` for acknowledgements at the beginning of a thesis and `\chapter` for the end.

CHAPTER 2

Theoretical basics

In this chapter the standard model with its elementary particles and interactions is introduced to allow discussing the fundamental kinematics in a particle detector. Furthermore the analysis in this thesis is motivated and the underlying problems are outlined. For this reason a brief introduction to the theory of particle production and decay is given.

2.1 The Standard Model of Particle Physics

Originally created in an attempt to unify the electromagnetic, weak ,and strong force under one theory the Standard Model of particle physics represents the status quo in this field summarizing the elementary particles and their interactions. The model is a gauge quantum field theory and its internal symmetry is the unitary product group $SU(3) \times SU(2) \times U(1)$ in which the interactions are represented by particles named gauge bosons. Figure 2.1 shows the Standard Model particles and their central properties which will be introduced in this chapter starting with the interactions and their mediators followed by a summary of the particles and lastly a section on the top-quark in addition to its properties directly relevant for the research in this thesis.

Although failing to answer open questions like the origin of dark matter or neutrino oscillations, the Standard Model has proven to be a powerful model being very successful in providing experimental predictions for decades.

2.1.1 Force carrier particles

There are three interactions represented in the Standard Model of Particle Physics, the electromagnetic interaction, the weak interaction, and the strong interaction, each represented by a integer-spin particle, called mediator boson. Gravity is usually not included in this model as it is neglectable at this scale.

The electromagnetic force is described by the theory of quantum electrodynamics.Its mediator is the photon, a massless particle that couples to the electric charge. The weak force, most commonly known as the interaction of the β-decay, couples to the weak charge, an intrinsic property of all fermions, and its mediators are the Z- and the two opposite-charged W-Bosons.

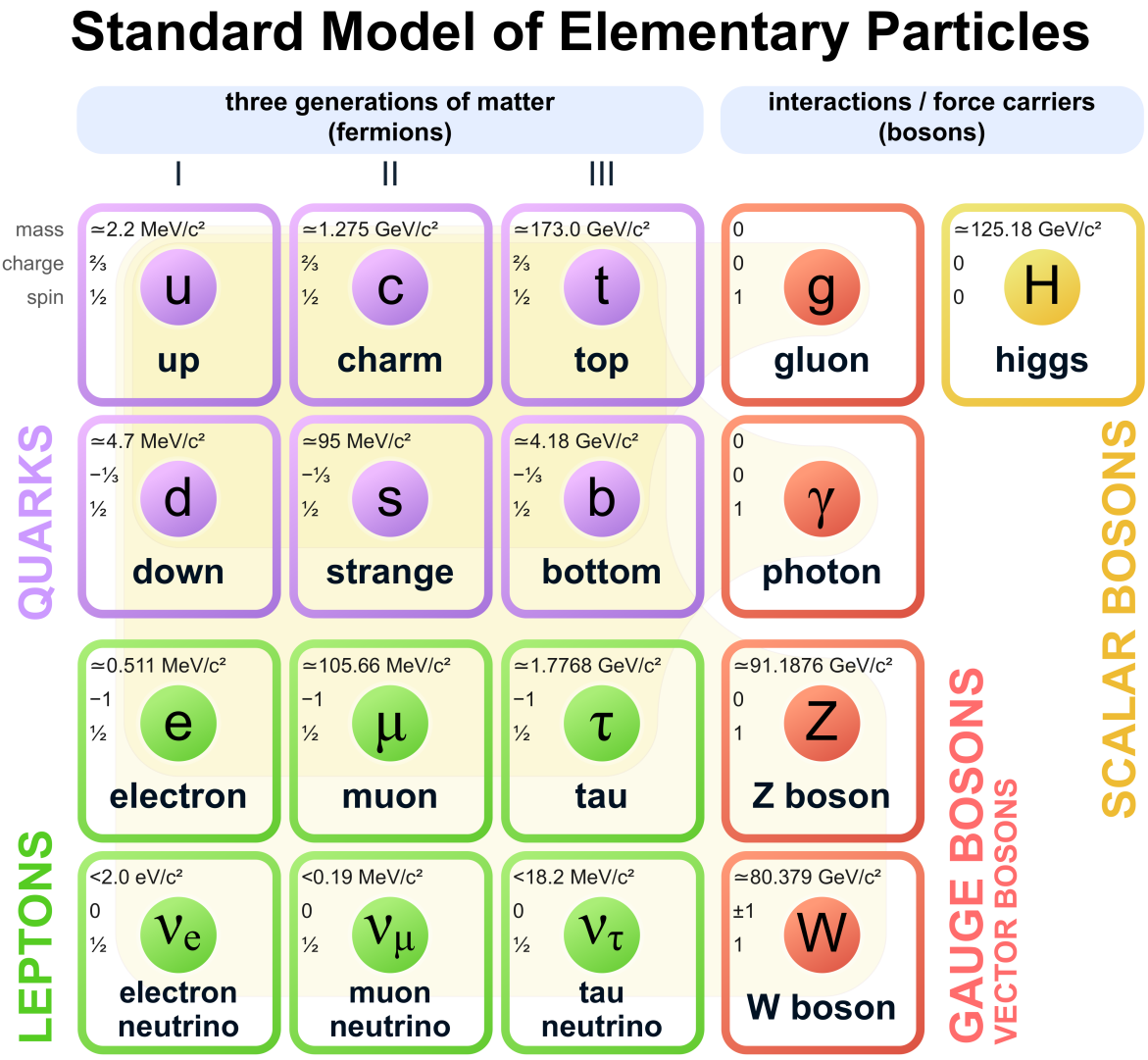


Figure 2.1: Summary table of the Standard Model particles and their properties. The sketch [4] was updated to PDG 2018 data [5]

The electromagnetic and the weak force can be unified as the electroweak interaction forming a $SU(2) \times U(1)$ symmetry.

The strong force is responsible for the binding of quarks in the nucleons. It is mediated by the 8 differently flavoured gluons. It only couples to particles with colour charge and is represented by the $SU(3)$ symmetry term in the Standard Model.

As already stated the gravitational force is not included in the standard model as its coupling strength at the scale is only of the order 10×10^{-37}. Although it is not a mediator, the Higgs boson was included in the Standard Model after its discovery in 2012.

2.1.2 Matter particles

The non-mediator particles in the model are called *fermions*, which are half-integer spin. They can be broadly classified into three generations of leptons and quarks

The first generation of particles forms the most common form of matter. The electron, the electron-neutrino, and the up- and down-quark, which are the main constituents of the proton and neutron. In the higher generations the defining quantum numbers stay the same while the mass of the particles increases. Moreover, there is an antiparticle for each particle with all quantum numbers reversed.

The neutrinos interact only via the weak interaction. There is one neutrino for each lepton-family. The non-neutrino leptons electron e^-,the muon $^-$, and the tauon $ø$ have electric charge in addition to their weak charge and therefore also interact via the electromagnetic force. Furthermore, the six quark flavours are up and down, strange and charm, bottom and top. The quarks are the only particles interacting with all three forces. They carry not only a weak and an electric charge but also colour charge enabling them to interact with gluons.

2.1.3 The CKM matrix

One aspect of the standard model worth looking at in particular is the CKM matrix which describes the probability of a quark of one flavour generation to be transformed into another quark of suitable charge via the charged weak interaction namely a W-boson. Each parameter V_{ij} represents the probability of a transition between flavour i to flavour j. The elements of the matrix predicted with the standard model theory show great agreement to the experimental results. [**pdg**]

$$\begin{pmatrix} d' \\ s' \\ b' \end{pmatrix} = \begin{pmatrix} V_{ud} & V_{us} & V_{ub} \\ V_{cd} & V_{cs} & V_{cb} \\ V_{td} & V_{ts} & V_{tb} \end{pmatrix} \cdot \begin{pmatrix} d \\ s \\ b \end{pmatrix}, \tag{2.1}$$

$$\begin{pmatrix} d' \\ s' \\ b' \end{pmatrix} = \begin{pmatrix} 9.97446 & 0.22452 & 0.00365 \\ 0.22438 & 0.97359 & 0.04214 \\ 0.00896 & 0.04133 & 0.999105 \end{pmatrix} \cdot \begin{pmatrix} d \\ s \\ b \end{pmatrix}. \tag{2.2}$$

Important properties not listed in figure xy are the cross-section and the branching fraction.

2.1.4 Top quark properties

The most essential aspects of particle interactions, underlying the events of interest of this work, are the production and decay processes of the top-quark. This section introduces the main properties of the top-quark. Chapter **??** then expands this explanation by discussing the production and possible decay modes of top-quarks in the Large Hadron Collider (LHC).

The top-quark, being the third generation up-type quark, is special because its mass exceeds the masses of the other quarks by orders of magnitude. Its mass is about $173\,\mathrm{GeV}/c^2$ which is higher than the masses of the weak mediators. It almost exclusively decays into a W-boson and a bottom-quark,

$$\frac{\Gamma_{t\to Wb}}{\Gamma_{t\to Wq}} = 0.957 \pm 0.034, \tag{2.3}$$

and has a lifetime of $\sim 5 \times 10^{-25}\,\mathrm{s}$ which is smaller than the typical hadronisation time, meaning that the top-quark forms no bound states with other quarks and instead decays. These essential properties of the heaviest quark lead to some interesting aspects and motivate one to research its properties. More information on the process involving a top-quark investigated in this is given in chapter **??**. For more information about the standard model see [3, 6]

2.2 Kinematics of collider physics

There are a few kinematic variables and experimental properties worth discussing because they are characteristic for collider experiments which will be briefly introduced in this section.

One of the most important attributes of a collider experiment is its centre-of-mass energy, $\sqrt{s}$. This denotes the available energy for particle analysis offered by the experiment. It is defined as:

$$\sqrt{s} = \sqrt{\left(\sum_i E_i\right)^2 - \left(\sum_i p_i\right)^2} \tag{2.4}$$

where E_i and p_i are the energies and momenta of the initial state particles. In the case of the Large Hadron Collider used for the simulation in this thesis and introduced in chapter **??** two proton beams of equal energy are brought to collision resulting in an energy formula of

$$\sqrt{s} = 2E_{beam} \tag{2.5}$$

assuming there are two beams with energies much higher than the particles' masses and therefore one can neglect them in the calculation.

The second property is the luminosity of an experiment which defines how many interactions an experiment can produce. For the colliding gaussian beams underlying the simulations in

this work it can be defined as:

$$L = f \frac{n_1 n_2}{4\pi\sigma_x\sigma_y} \tag{2.6}$$

where f is the frequency of the beams, n_i is the particle number per bunch and the σ_i values stand for the horizontal and vertical beam size respecively. The integrated luminosity is then used to estimate the total number of interactions for a certain period of measurement:

$$\mathcal{L} = \int L(t)dt \tag{2.7}$$

Multiplied with a cross-section the corresponding number of interactions to be expected, N, can be defined as $N = \sigma\mathcal{L}$.

CHAPTER 3

The LHC and the ATLAS detector

For most researches in modern particle physics there are two main constraints. The first one arises from the statistical nature of decay and creation processes in particle physics. Many of the most interesting events occur extremely rarely and call for large amount of data or more precisely high luminosity, to achieve statistically significant results. Secondly the energy scale of particle physics is enormously high and therefore the energy needed to allow breaking the structure of particles below the nuclear scale.

This work was carried out using simulations based on the ATLAS [7] detector at the Large Hadron Collider (LHC) [8] which offers both a record breaking center-of-mass energy and luminosity. This chapter summarises both machines and provides the knowledge necessary to understand the simulations used. First of all, a summary of the machines' parts and their technical details is given, followed by a description of how the detector detects particles and how their properties are measured. Simulations of detector events are explained and the event selection for this work is covered.

3.1 Large Hadron Collider

The Large Hadron Collider, located at the facilities of the European Organization of Nuclear Research (CERN) close to Geneva, was built to extend the frontiers of modern particle physics by delivering high luminosities and reaching unprecedented high energies thereby providing the data benefitting multiple particle physics experiments.

The LHC is a circular particle collider with a circumference of 26.7 km designed to accelerate and collide two counter-rotating proton beams. The protons are accelerated in bunches of up to 10^{11} protons at energies up to 6.5 TeV and a luminosity of $10^{34}\,\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$, achieving the record center-of-mass energy of up to 13 TeV. The bunches are pre-accelerated by a number of accelerators before being inserted in the last, so called, storage ring. An overview of the acceleration system is given in figure 3.1 and for more detailed information one can refer to the LHC design report. [8]. The four main interaction points, at which the beams are brought to collision, contain the main experiments of the LHC. Two of them are general purpose detectors, namely ATLAS [7] and CMS. [9] The third is the LHCb [10] which focuses on bottom-quark physics. Lastly, ALICE [11] is used for investigating heavy ion collisions. Figure 3.2 shows a

sketch of the LHC's location and the positions of the four main experiments.

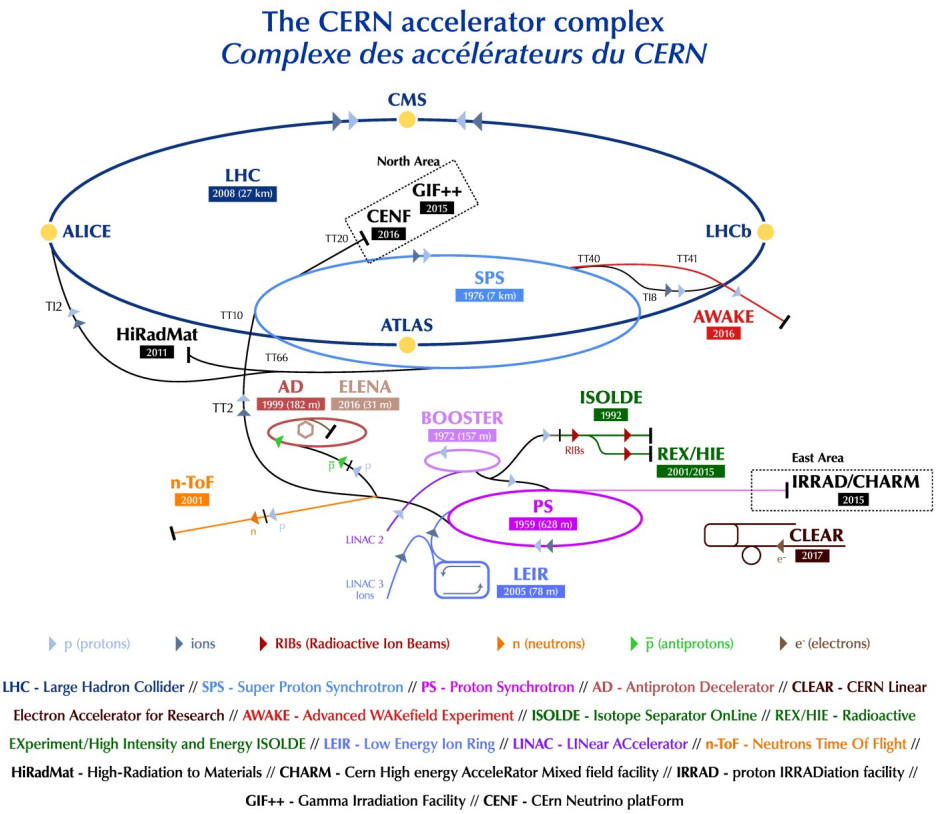


Figure 3.1: Sketch of the LHC accelerator complex showing the acceleration systems and the main storage ring with its experiments. [12]

3.2 The ATLAS detector

The ATLAS detector is a general purpose detector meaning it aims at covering a maximum number of final states, enabling reasearchers in may topics of particle physics to use its data.

ATLAS, "A Toroidal LHC Apparatus" has the distinguishing structure of a general purpose detector, its innermost part formed by tracking detectors directly surrounding the interaction point, followed by calorimeters, and a final layer for muon tracking. All the components are visualized in figure 3.3 including two humans to give an impression of the detector's size.

The innermost tracking detectors are summarized under the name Inner Detector (ID) and consist of two silicon detectors namely the Pixel Detector and the Semi Conductor Tracker as well as a straw detector named Transition Radiation Tracker. The Inner Detector allows for precise measurement of not only charged particles' position, and thus vertex information, but also for their charge and momentum.

The calorimeter system is divided into two components, being the electromagnetic calorimeter and the hadronic calorimeter, allow to measure the energy of particles by stopping them

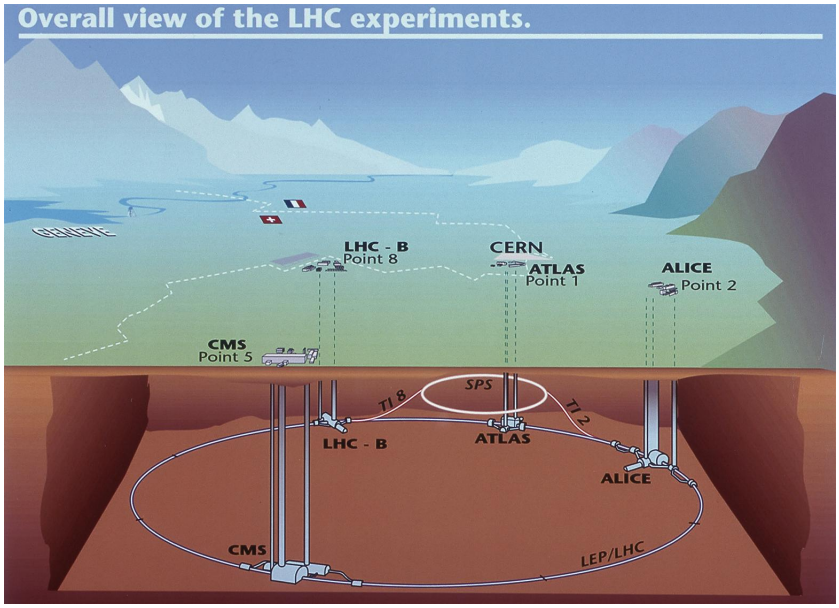


Figure 3.2: Sketch of the LHC ring, the position of the experiments and the surrounding countryside. The four big LHC experiments are indicated (ATLAS, CMS, LHC-B and ALICE) along with their injection lines (Point 1, 2, 4, 8). [13]

in the detector material.

The Muon Spectrometer is the last tracking detector which identifies particles crossing it as muons, as all other charged particles are usually stopped in the calorimeter system.

In the following the concept of each detector component is briefly introduced [14] to then summarize how particles can be detected and distinguished. The reconstruction of objects from the detector response is explained and an event selection for tW candidates introduced.

3.2.1 Tracking detectors

Tracking detectors are used to measure a charged particle's trajectory, momentum and charge value, of which two types are used in the ID of the ATLAS detector. The Pixel detector and the Semi Conductor Tracker (SCT) are silicon detectors and the Transition Radiation Tracker (TRT) is a straw-based tracking detector. For all detectors it holds true that they are surrounded by a magnetic field and cover a pseudorapidity range of $|\eta| < 2.5$. The magnetic field results in curved trajectories enabling an estimate of momentum and charge.[16]

Pixel detectors are based on ionisation of charged particles in the semiconductor material. The induced charge is picked up by the detector's pixels providing a position information. To provide a 3-dimensional trajectory the pixel-chips are ordered in 4 layers around the beam pipe where the layer closest to the point of interaction, called Insertable B-Layer (IBL), was added in 2015. It is located only 3.3 cm from the beam pipe and allows to detect vertices very close to the interaction point mainly originating from b-quarks giving the layer its name. [17]

The SCT as a silicon microstrip detector is the second silicon-based tracker immediately following on the pixel detector. It consists of modules of four silicon strip sensors organised in

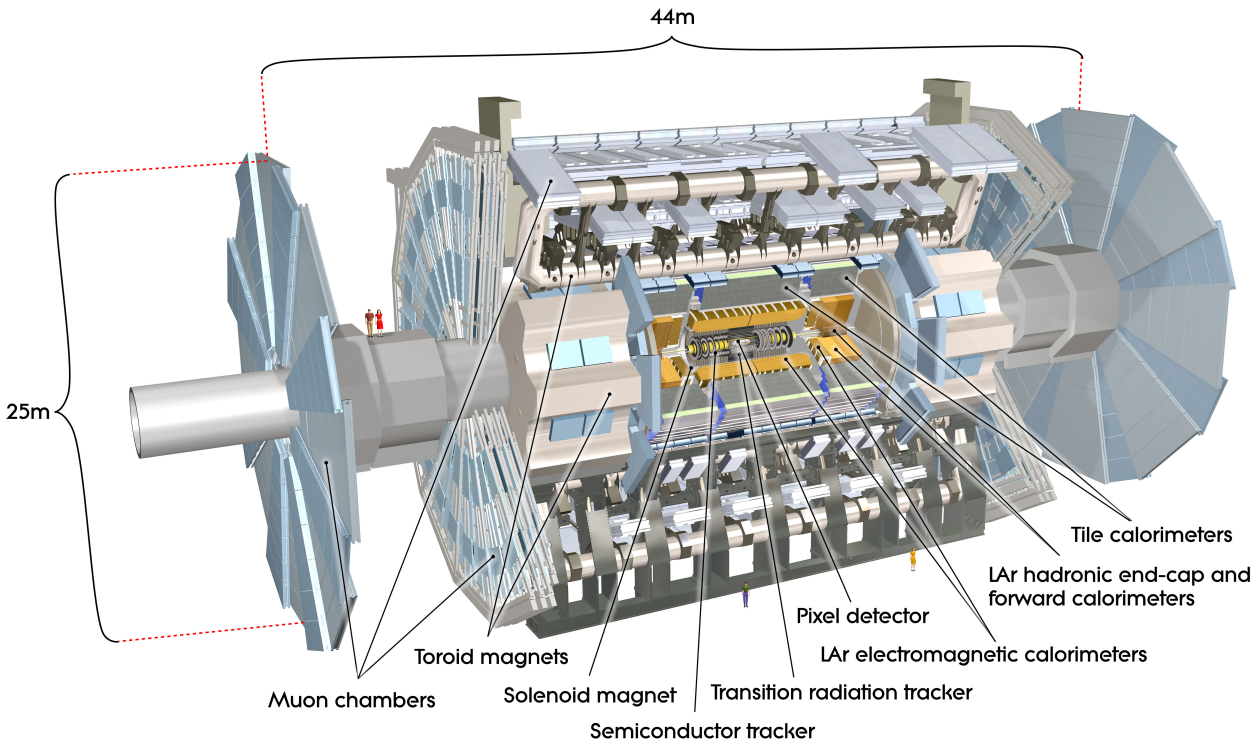


Figure 3.3: Sketch of the ATLAS detector and all its components including two average humans for scale. [15]

four barrel layers and eighteen planar endcap disks.

The TRT is structured in straw tubes each tube being an individual drift chamber with a strong potential difference due to negatively charged walls. The tubes are filled with a gas mixture (Xe or Ar) causing transversing charged particles to ionize and then by accelerated to the walls. A cascade is initiated and a measurable signal in the potential difference is measured. Between the tubes material is inserted resulting in transition radiation. This radiation has a cross section way higher for electrons than for other particles over a wide range of energy thus adding particle information to the track information provided by the TRT.

A particle being detected in a layer of the ID is called a hit. The record of hits gives an estimate on the particle's trajectory and can thereby also provide information on the vertex the particle originates from. This vertex information is worth mentioning as for a an experiment with an event count as large as the ATLAS experiment's events interfere and information from so called pileup events can affect the event's information. As pileup originates from different events it can be separated from the event of interest by separating the vertices.

3.2.2 The ATLAS calorimeter system

The ATLAS calorimeter system is divided into three main parts. The electromagnetic (EM) calorimeter, comprising a barrel and two end-caps, and the hadron calorimeter, built by a tile

calorimeter, consisting of a a barrel and two so called "extended barrels", and the hadron endcaps. The third part is the forward calorimeter which additionally focuses on electromagnetic interaction. The tile calorimeter is scintillator-based apart from that the main part of the calorimeter system is based on liquid argon. The components cover a pseudorapidity range of $|\eta| < 4.9$

Calorimeters determine a traversing particle's energy by exploiting the formation of particle showers. [14] Due to inelastic collisions in the detector's material, the energy of the original particle is distributed on a cascade of secondary particles finally stopped by ionization. The resulting charge or photons can be measured as an estimate of the initial energy.

Electromagnetic calorimeters exploit the energy loss of electromagnetically interacting particles in matter. Mainly photons and electrons loose their energy based on pair production and Bremsstrahlung respectively. The energy loss initializes a cascade of particle decays called an electromagnetic shower. The decay stops when the shower particles do not hold sufficient energy for a decay anymore. The energy of the final state shower particles is picked up by the detector representing the initial particle's energy. The ATLAS ECAL is a sampling calorimeter, built of two alternating layers of absorber and detection material. In the absorber the showers are induced to subsequently be detected in the detection layers.

As the ECAL uses electromagnetic showers the hadronic calorimeter depends on hadronic shower evolution. Hadronic showers are initialized due to ionisation or strong interaction with the material's nuclei. If the resulting particles still interact with the material a shower evolves. The hadronic tile calorimeter is made of alternating layers of steel absorbents and scintillators covering a pseudorapidity range of $|\eta| < 1.6$. The hadronic endcap calorimter (HEC) is liquid argon-based and covers $1.4 < |\eta| < 3.1$ Due to the larger size of hadronic showers the HCAL occupies more detector space than the ECAL.

3.2.3 The Muon spectrometer

The second tracking detector of ATLAS is the muon spectrometer which is the outermost part of the detector. The task of the spectrometer is to detect charged particles traversing the calorimeter without being stopped or deploying their complete energy, and to do both collect trigger information and information on trajectory and momentum. Due to these two tasks the spectrometer is bifid with the first part being the trigger chamber covering a range of $|\eta| < 2.4$, followed by the high-precision chamber with a range of $|\eta| < 2.7$. The main detector's support feet cause a further gap at about $\phi = 300°$ and $\phi = 270°$.

Normally the only charged particles left to be detected in the muon spectrometer are muons giving the component its name and allowing to provide good trigger information for researches interested in muons in the final event topology.

3.2.4 The ATLAS coordinate system

The ATLAS coordinate system is a right-handed and right-angled coordinate system with the z-axis pointing along the LHC's beam pipe. The corresponding transverse plane is defined by the x-axis pointing towards the ring's centre while the y-axis points upwards. The origin of the system is defined by the nominal point of interaction. The polar angle θ is the angle between

the z-axis and the x-y-plane and the azimuthal angle ϕ is the angle between the x- and the y-axis.

Alternatively, as in this work, an event's topology is described by the azimuthal angle ϕ, the pseudo-rapidity η, and the transverse momentum p_{T}. The pseudo-rapidity replaces the polar angle and is defined as

$$\eta = \frac{1}{2} ln \left[tan \left(\frac{\theta}{2} \right) \right]. \tag{3.1}$$

The transverse momentum is defined by

$$p_{\mathrm{T}} = \sqrt{p_x^2 + p_y^2} \tag{3.2}$$

where p_x and p_y are the momenta along the corresponding axes.

The angular variables are defined within

$$\eta \in [-\infty, \infty], \, \phi \in [-\pi, \pi]. \tag{3.3}$$

This cylindrical system makes use of the shape of the ATLAS detector and the momentum conversation of the transverse plane due to it being perpendicular to all initial beam momenta.

3.2.5 Particle Detection in the ATLAS detector

This section focuses on the detection and distinction of different particle types in the ATLAS detector. The capability and combined information of the detector components is introduced giving an explanation of the general working principle and also of the characteristics defining the events in this work. Figure 3.4 gives an overview of typical particle interactions and detections.

In order to reconstruct the particles in an event low level information is gathered using the direct detector output and then associated to the higher level particle information.

The information from the ID is called a track and contains not only the trajectory but also tells how consistently a track holds hits in every layer. A track offers momentum and charge information and can be associated with a vertex and a possible energy deposition in a calorimeter. The vertex reconstruction arising from the track information allows to define a primary vertex defined by the highest sum of squared transverse momenta while additional vertices are identified as pileup vertices. Secondary vertices originating from tracks connected to the original vertex can be collected to identify short-lived particles.

The calorimeter data is summarised in clusters. Clusters are neighboring calorimeter cells with energy depositions significantly higher than the expected noise. A cluster is formed around a high energy deposition and can be associated to hadrons or jets or even to a corresponding track.

In the following the higher order objects reconstructed from this basic information are introduced to then explain the decisions made in the event selection for the tW channel.

Electrons are constructed from energy deposits in the EM associated with ID tracks. To improve the decision rule, a likelihood object quantity is constructed from the shape and

the ratio of the calorimeter to tracker response, and a set of further variables suitable for a better discriminant. There are three settings for the likelihood object namely tight medium and loose depending on how restrictive the analysis is. Lastly an isolation quantity is defined based on cones around the track and the EM deposit to further decimate background and fake electrons. [18]

Jets are cones of particles originating from the common hadronisation of a quark or gluon. In the detector they are reconstructed using 3-dimensional topological clusters of calorimeter energy. [19] In addition to this there is further information that can be associated to jets, i.e., an ID track or a vertex using a jet-vertex-tagger to minimize the impact of pile-up events and to associate to secondary vertices. For reconstruction the anti-$k_t$algorithm was used. [20]

Muon reconstruction uses MS hits matched with ID tracks. The choice can be further specified by applying an identification cut based on MS/ID agreement and integrity of MS hit response. As for electrons isolation can be required. [21]

***b*-jets** are jets originating from the decay of a b-quark and therefor a strong discriminant for events containing a t decay. The process of identifying a b-jet is called b-tagging and uses a multivariate discriminant. The topology of b-jets is distinguishable from other jets due to for example clear secondary vertices, vertex alignment of a primary, a secondary b-vertex and a tertiary c-vertex, the decay length, and the characteristic energy scale. [22, 23]

Missing transverse momentum arises from momentum imbalance in the transverse plane. Momentum in the transverse plane should be preserved due to it being perpendicular to the beam axis and imbalance is an indicator for neutrinos escaping the detector. It is calculated using two contributions, one being signals from fully reconstructed and calibrated particles and the other one is information from reconstructed charged particle tracks. [24]

In addition to this the ATLAS trigger system has to be mentioned. Although potentially deserving a chapter of its own for this work it is sufficient to just state and briefly explain trigger information.

Triggers are used to filter events before the actual event selection is taken into account. Given the incredible luminosity of the LHC such a preselection is important to minimize the data actually processed by the more complicated analysis algorithms and selection schemes. The ATLAS trigger system consists of three triggers, namely the Level 1 (L1), Level 2 (L2), and the Event Filter (EF) where L2 and EF are generally referred to as the High-Level Trigger (HLT).

The L1 is is completely hardware-based and its decision making process is motly based on information from the calorimeter and the muon trigger chambers. The decision step relies on high-p_{T} objects and their multiplicity in an event while also considering missing transverse momentum and the beam condition to provide a first and very broad event selection.

The HLT is based on software and takes the L1 events as input. L2 defines so called regions of interest (ROI) as those regions in the angular plane where the trigger objects for L1 were detected and applies further trgger cuts to these objetcs. The EF fully analyzes the event based on the complete information available.

This trigger information can be used for event selection making sure that certain final state objects are dominant in the topology and also to just apply a first, broad selection.

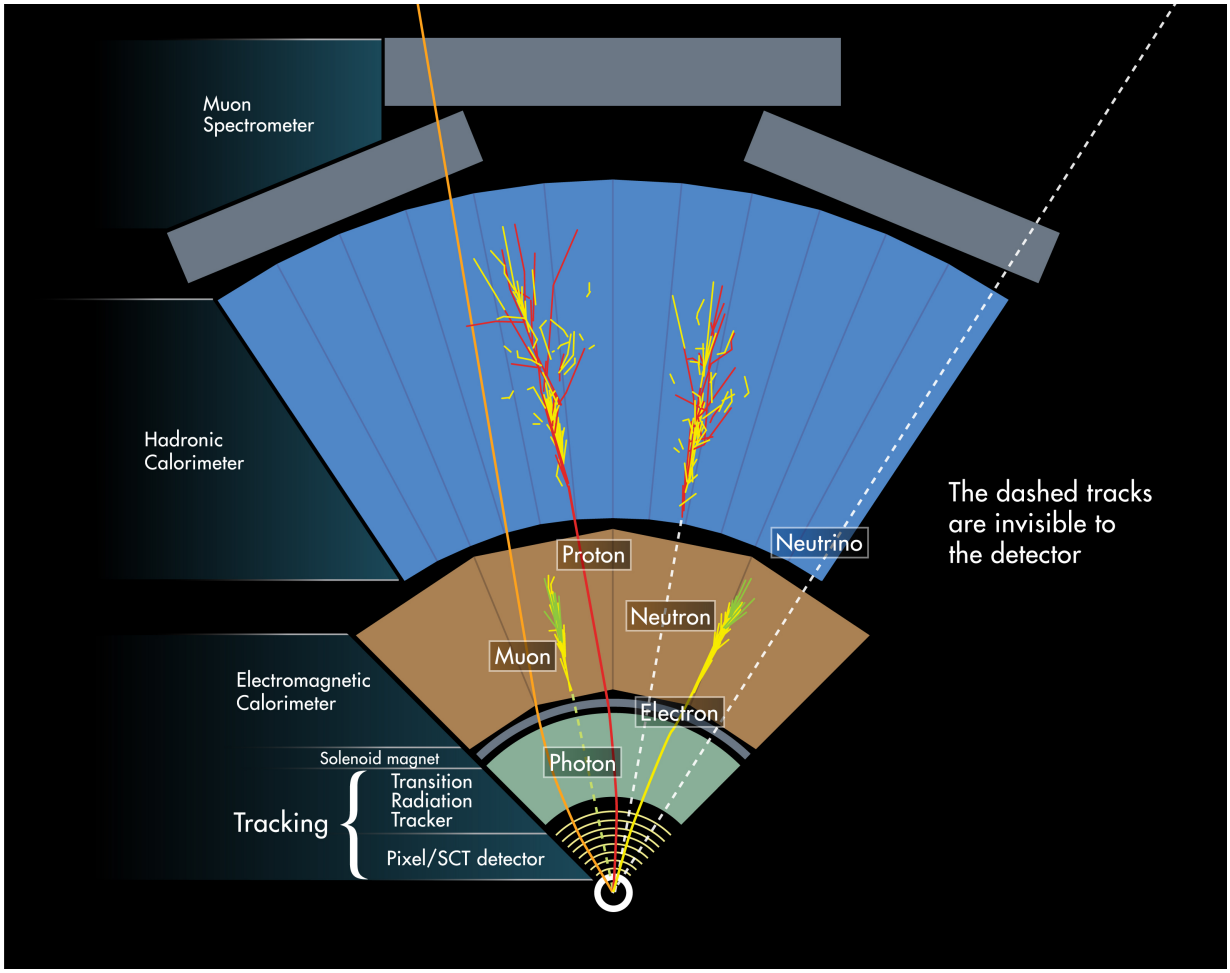


Figure 3.4: Scheme of the ATLAS-detector showing examples of typical particle detections. [15]

3.3 tW event selection

The process separation of interest for this work is tW $t\bar{t}$. Other backgrounds for the tW channel are reduced by applying an event selection:

- A single electron or muon trigger
- Electrons: tightly identified, isolated, $E_{\mathrm{T}} > 26\,\mathrm{GeV}$
- Muons: tight isolation, $p_{\mathrm{T}} > 26\,\mathrm{GeV}$
- Opposite-charge lepton pair
- Leading lepton $p_{\mathrm{T}} > 27\,\mathrm{GeV}$

- Veto for a third lepton $p_T > 20\,\text{GeV}$
- A lepton must match the trigger
- At least one jet with: $p_T > 25\,\text{GeV}$, $|\eta| < 2.5$, tagged at 77 % working point

After this preselection the events are categorized in regions based on the jet and b-jet multiplicities. For this work the region with exactly two b-tagged jets was used denoted as `2j2b`. This region has especially high impact from the NLO interference with a $t\bar{t}$ final state.

3.4 Monte Carlo simulation

A Monte Carlo simulation is a computer based stochastic calculation of a process that in principle could be deterministic but the problem and the amount of statistics suggest a stochastic approach.

Monte Carlo for the ATLAS detector is the simulation of detector events to calibrate the performance and generate an estimator to compare model and measurement. As for for this work the classifying tools and event selection rules are tested and tuned based on simulations to achieve a better understanding of what efficiencies are to be expected in a data analysis. This provides the truth information to an event that is needed for supervised learning, see section 5, and cannot be provided by data.

A simulation has to be based on quantities that can not only be calculated from theoretical models but also measured during the experiment because an interplay between simulation and actual data is needed to adjust the simulation to the experimental circumstances. In collider physics predictions are mainly based on the cross-section of an interaction stating how high the probability for certain interaction is. In addition to that the decay width of a particle is needed to describe how particles generated in an interaction behave in the detector system.

A proton-proton collision is modeled by describing the protons as a sea of partons, i.e., the gluons and quarks their momentum is carried by, and then calculating the cross-sections for these partons to interact individually. The technique of seeing the collision as an interaction of two individual partons while decoupling these interactions from the parton interactions in the individual protons is called factorization. For a parton carrying the fraction x of the proton's momentum for a center-of-mass energy of $\hat{s}$ the cross-section $\sigma(p_1 p_2 \to X)$ to create a final state X can be described as:

$$x_i = \frac{p_{parton,i}}{p_{proton}} \tag{3.4}$$

$$\hat{s} = x_1 x_2 s \tag{3.5}$$

$$\sigma(p_1 p_2 \to X) = \sum_{i,j=q,\bar{q},g} \int dx_i dx_j f_{i/p_1}(x_i, \mu^2) f_{j/p_2}(x_j, \mu^2) \cdot \hat{\sigma}_{ij}(ij \to X; \hat{s}, \mu^2) \tag{3.6}$$

There are two quantities in this equation not yet explained. The first on is the parton density function (PDF) of the proton denoted as $f_{i/p_m}(x_i, \mu^2)$. This is a function describing the probability that a parton with momentum fraction x_i in the proton k takes part in the hard scattering. The PDF is independent of the collision and has to be gained from experiments rather than through calculations.

The second quantity μ is the so called factorization scale. It describes to which degree the interactions of the partons in the proton can be neglected. It is an arbitrary scale determining the precision of the simulation to some degree.

3.4.1 ATLAS simulation

The MC simulation for the ATLAS detector is generated in two steps. [25]

In the first step the actual collision is simulated defining the particles in the final state. The underlying algorithm is called Monte Carlo event generator.

Secondly the response of the detector to these particles is simulated by the detector simulation allowing objects to be reconstructed from the event like from actual data gaining a comparison on every level of reconstruction.

Furthermore a set of different options and generators can be chosen for the simulation. Examples would be how a parton shower is simulated or to which degree multi parton interaction is taken into account.

3.4.2 Systematic uncertainties in Monte Carlo simulations

Simulations as described earlier have to be based on certain assumptions about the input parameters especially in those cases where the theory does not offer a precise result. This results in systematic uncertainties arising from the simulations and it is not obvious how good the actual data is represented by the assumptions made. In those cases different Monte Carlo samples are produced each representing a different assumption about the initial parameters showing first of all how the data should behave with these scaled paramters and allowing to test whether the systematic or the original sample fits better to the data. These samples of Monte Carlo will just be referred to as systematics in this work.

Usually the simulations use the leading order cross-section. When next-to-leading order diagrams become relevant the simulation has to be adapted resulting in different samples for different orders. This work focuses on the tW and $t\bar{t}$ events interfering at NLO as described in 4 The total NLO amplitude is the sum of the of singly resonant diagrams and all doubly resonant diagrams:

$$\mathcal{A} = \mathcal{A}_{tW} + \mathcal{A}_{t\bar{t}} \tag{3.7}$$

Calculating the cross-section requires the square of this amplitude:

$$|\mathcal{A}|^2 = |\mathcal{A}_{tW}|^2 + 2\mathcal{R}\{\mathcal{A}_{tW}\mathcal{A}_{t\bar{t}}^*\} + |\mathcal{A}_{t\bar{t}}|^2 \tag{3.8}$$

$$\equiv \mathcal{S} + \mathcal{I} + \mathcal{D} \tag{3.9}$$

As a nominal sample in this work the Diagram Removal approach, denoted as DR, is used. It fully removes the $\mathcal{A}_{t\bar{t}}$amplitude and thus both the pure $t\bar{t}$ contribution as well as the interference term vanish.

$$|\mathcal{A}_{DR}|^2 = \mathcal{S} \tag{3.10}$$

Alternatively the Diagram Subtraction or DS can be used which is in this work represented by the systematics sample. It introduces a gauge invariant term that cancels the $t\bar{t}$ contribution and in a simplified way can be written as

$$|\mathcal{A}_{DS}|^2 = \mathcal{S} + \mathcal{I} + \mathcal{D} - \widetilde{\mathcal{D}} \tag{3.11}$$

$$\approx \mathcal{S} + \mathcal{I}. \tag{3.12}$$

The nominal samples were produced using a full ATLAS detector simulation implemented in GEANT4 and the systematic samples were mostly simulated using ATLFAST2. [26, 27]. The systematic samples used in this thesis were produced using GEANT4 as a DS sample requires a full simulation.

The events were generated by POWHEGPYTHIA 8. [28]

CHAPTER 4

The tW Channel

For the first part of this chapter the decay of the top-quark is discussed. Then the production channels of the top-quark are diagrammed. Finally the production channel of interest and its final state are explained in the context of its background at the Large Hadron Collider (LHC). For more information about the LHC see chapter 3. The diagrams shown in this chapter were created using the code available at [**feyn_repo**].

4.1 Top-quark decay

4.2 Top-quark production

Most commonly top-quarks are created via the strong force in a top- anti-top-quark final state. Both gluon-gluon fusion and $q\,\bar{q}$-annihilation are possible production processes are possible. Figure 4.1 shows the processes at leading order (LO). Leading order means that no corrections for additional gluon emission and virtual corrections were taken into account.

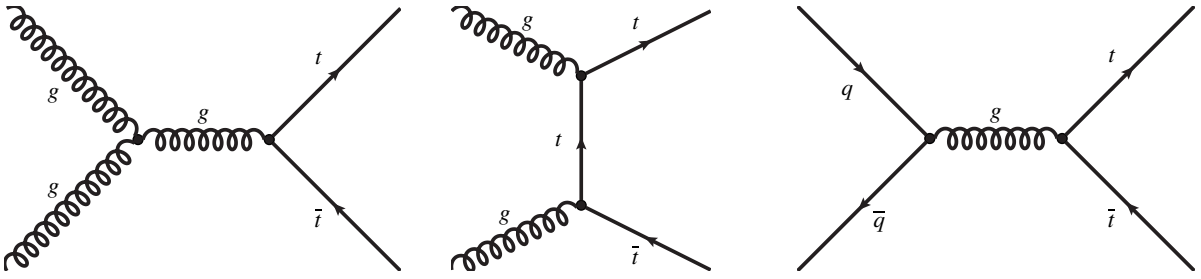


Figure 4.1: $t\bar{t}$ pair production feynman diagrams at LO.

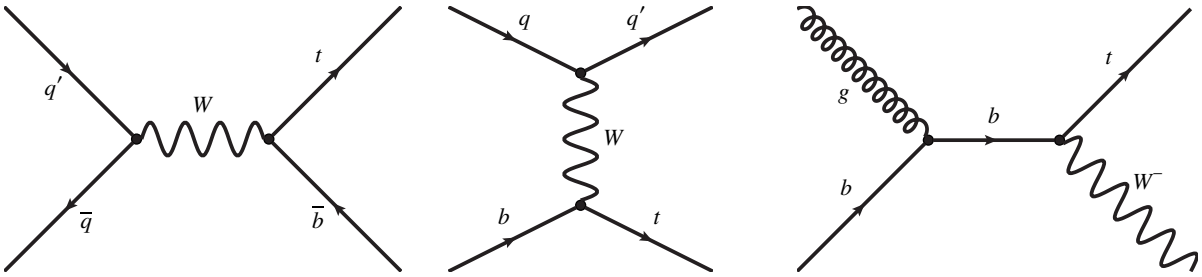
Additionally, top-quarks can be produced as single top-quarks via the electroweak interaction. The dominant process is the production through the interaction of a bottom-quark and a W-boson shown in figure 4.2(a). Also possible but the least common is the s-channel involving a virtual W-boson displayed in figure 4.2(b). Finally the channel of interest for this work is the tW-channel diagrammed in figure 4.2(c).

Figure 4.3 shows the final state of the tW decay with both W-bosons decaying leptonically at LO. Therefore the channel is named dilepton channel. At Next-To-Leading Order (NLO) a gluon splitting can result into a further bottom-quark in the final state. Figure 4.4 shows the $t\bar{t}$ final state in comparison to the NLO final state of the tW-channel. In the final state these channels these channels are not distinguishable, i.e., they interfere. These diagrams will be referred to as doubly resonant, in contrast to the singly resonant diagrams such as diagrammed in figure 4.1. Given that the $t\bar{t}$-cross-section is 10 orders larger than the tW-cross-section this gives rise to a NLO correction exceeding the actual LO cross-section. This results in the tW-channel not being well-defined at NLO. To allow treating tW as a separate process a workaround has to be used. There are two possible schemes for handling the interference in the calculation of the cross-section:

Diagram Removal (DR) removes all diagrams containing a second top-quark propagator that can be on-shell. It is used for the production of the nominal sample in this work.

Diagram Subtration (DS) only the $t\bar{t}$ contribution is cancelled when the top-quark is on-shell. This scheme is used for the systematics sample.

For more information on the schemes and their motivation see section 3.4.2,



(a) (b) (c) Picture 2

Figure 4.2: Single-top-production diagrams: s-channel (b), t-channel (a), tW-channel (c)

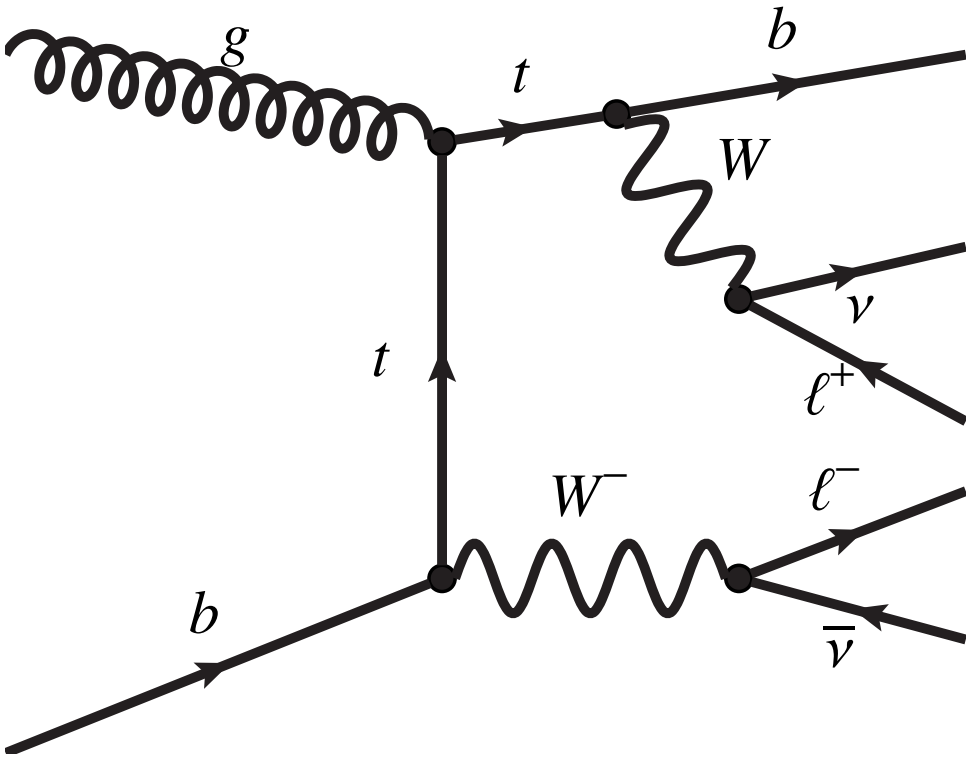
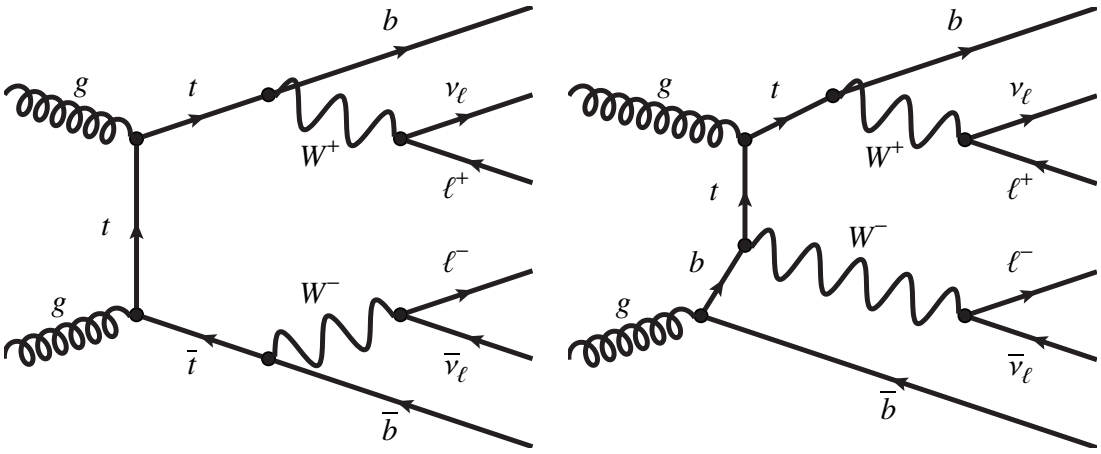


Figure 4.3: text



(a) (b)

Figure 4.4

CHAPTER 5

Machine Learning

5.1 The concept of machine learning

Over the last decades, computers have become indispensable tools of science; handling large amounts of data, completing tedious calculations, and controlling sophisticated experiments. For the most part, these machines were assigned discrete tasks and they followed step-by-step commands, designed beforehand by human users, and had expected outcomes. For particle physics in particular, computers have been used to select and process data from large samples, allowing the processing of these data at a speed beyond human capabilities. However, the selection rules always had to be generated by the user, therefore requiring an in depth understanding of the underlying system. In contrast, machine learning enables a program to establish its own decision rules, improving these over several iterations and thereby learning to solve the problem by itself.

There has been a great effort over the last decades trying to implement a way for machines to learn from known quantities. Thereby the machines would be enabled to analyse complex tasks ranging from voice recognition to object classification. The efficiency and validity of a machine learning model is highly dependent on the human understanding of the problem at hand. One prerequisite for a successful model is the tuning of the degrees of freedom and parameters to the complexity of the assignment. This is called *hyperparameter optimisation*, a task often proportional to the learning process itself.

Machine learning can be exemplified by drawing an analogy to human beings. In order to solve a problem, the machine needs to understand the system, to evaluate a decision step, and finally generate new decision steps. Understanding a system means to be aware of all its features and possibilities. Humans have their senses to easily break down observations into useful features and concepts that can then be processed for finding a decision. A computer has no such senses, and for most tasks, this means that the step of filtering information for a relevant subset of features has still to be done by humans or a good preprocessing algorithm. Once a system has been converted to a subset of features usable by a computer, the step of making its own decisions has to be implemented. This can be done by weighting and interconnecting the information using structures inspired by neurons and synapses in the human brain.

The structure and complexity of the network enables it to learn from data. In addition, a

metric is introduced that measures the quality of the model, called the *cost* or *loss function*. This function allows for iterative improvement as a decrease in its value is considered an improvement by the network. Combined with an optimiser, which suggests further steps, this function is a basis for a network to independently approach a good decision rule for a incompletely investigated topic.

A very commonly used machine learning technique is the*artificial neural network* which, on its own forms, an extended field that builds the base of this work. The essential concepts of machine learning will be explained in the context of neural networks.

5.2 Neural Networks

The artificial neural network, or just neural network, is one of the most commonly applied approaches to machine learning. Its structure and naming is inspired by the neurons forming the human nervous system.

todo

Instead of living neurons, a neural network consists of numerous very simple processors, called nodes. These nodes are usually structured into several layers as presented in figure. In addition, there are several ways to structure and connect these nodes, frequently matching a certain problem. In this explanation, only the most commonly way is explained. In that case each node of a middle layer is getting input from each node in its predecessor and is outputting to each node of the following layer. This obviously is only true in one direction for the extreme layers, input and output. A simple view of communication between nodes is shown in figure 5.1 describing the step between a node and the previous layer. This is called a feed-forward neural network, the math of which will be described in detail later.

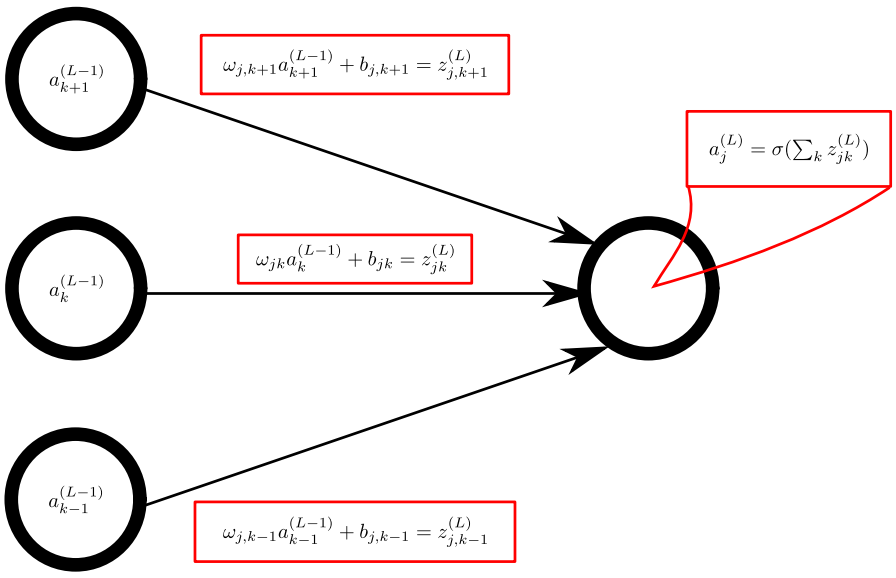


Figure 5.1: Network propagation from layer $(L-1)$ to layer L

The input layer

To understand a task and draw reasonable conclusions, the underlying system has to be understood at first, which means its features need to be determined and summarised. The human brain is capable of investigating unknown systems and learning the features that are the most unique or interesting ones. For that, the nervous system uses its senses to explore the system and process them later. To allow a machine to do something similar, the unknown system has to be represented in a way that it is clear for the network, what to look out for. This usually is the task that requires most preprocessing by the user. The simplest case is to submit a list of variables to the network. In particle physics, this could be kinematic variables of the final reconstructed objects in an event.

The input to a neural network is given to the input layer of nodes and subsequently processed through each following layer. For diverse tasks, different layers might deal with various parts of the information. However, in this work the linear way of giving all information used to an input layer and then processing it is used.

Decision making process

Computers representing nothing more than very powerful calculators, excel at performing high numbers of clearly defined calculations. This requires a precisely elaborated task containing no uncertainty. This is completely different for the human nervous system, which relies on a certain uncertainty when processing information through a net of neuron cells. In this net, the output of every neuron is taken as input for the surrounding neurons. The challenge of machine learning is to represent this fuzziness by many, somewhat discrete calculations. In a neural network, the neurons and their fuzzy interactions are represented by the nodes. Like neurons, each node can use input from many other nodes to create a new output signal. Thereby the sets of input information can be linked to each other in numerous ways. Combined with a weighting system, this allows one to create complex models and match a variety of problems.

$$z_j^L = \sum_{k=0}^{N} \omega_{jk}^L a_k^{L-1} + b_k \tag{5.1}$$

The input of every node is the weighted output of all previous nodes, as shown in equation (5.1). z_j^L is the input to the j-th node in the L-th layer. ω_{jk}^L is the weight from the k-th node in the previous layer to this node, a_k^{L-1} is that node's output, and b_k the relevant bias; representing a possible intercept of the functionality. The sum indicates that all k previous nodes contribute to the input to the j-th node.

The weight allows a network to predict which variables are correlated or allow for better decision rules when combined. In addition, the weights can just define the strongest variables and features. Furthermore, the output of each node is a non-linear combination of the input. This means the output can range from just one and zero to an exponential function. This is called the *activation-function* of a node. A common choice is the sigmoid function as presented in equation (5.2). [29]

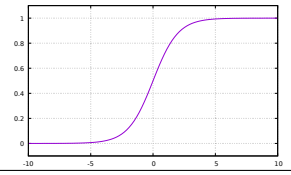
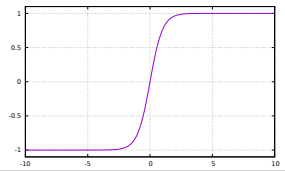
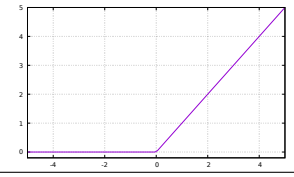
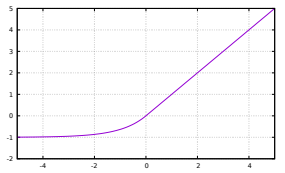
Name	Function	Plot
Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$	
Tangens Hyperbolicus	$f(x) = \frac{2}{1+e^{-2x}} - 1$	
Rectified Linear Unit, ReLU	$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$	
Exponential Linear Unit, ELU	$f(x) = \begin{cases} \alpha(e^x - 1) & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$	

Table 5.1: Selection of activation functions taken from the Keras documentation. [29]

$$a_j^L = \sigma(z_j^L) = \frac{1}{1 + e^{z_j^L}} \tag{5.2}$$

The sigmoid function has an output between 0 and 1, which is frequently desired for nodes. Especially for the final layer, as we are looking for predictions of outcome probabilities. A selection of further activation functions is shown in table 5.1.

For this thesis, mainly the exponential linear unit, or *elu*, and the rectified linear unit, or *relu*, were tested. *Elu* is suitable for converging the cost to zero relatively fast and provides the possibility of negative output while *relu* allows for the same benefit as sigmoid but requires less computational power for the simply linear output for positive values.

Of course the network does still not know the task assigned to it, but if we add a cost function to estimate the quality of a decision rule created by a certain combination of the input, we can easily make the network approach a better model in each step. Following the process of cost minimisation, the solution of the initial problem can be approximated. In this work, the cost function will be called the loss of the model.

In supervised learning, the network is trained with a set of labeled data. Each event in the training set has is assigned a label representing what process the network is looking at. This is referred to as *truth label* or just *label*. Comparing this truth information to the network output makes it very easy to calculate the loss as the deviation of the network output from the known label. A possible loss function is the *crossentropy* or just *binary crossentropy* for a binary output

result. As in this work the result is binary, signal and background, the binary crossentropy is the natural choice of loss function. Equation (5.3) shows the underlying function where p is the estimated probability for the prediction $\hat{y}$ and y is the truth label.

$$C = -(y \log p + (1 - y) \log(1 - p)). \tag{5.3}$$

The loss is the most important indicator for the training quality. This training quality must not be mixed up with the overall quality of generated model. Only after a test on a different sample or real data the model can be fully evaluated.

Optimisers - Choosing the next step

The probability interaction of the nodes combined with the loss function enables a network not only to create a model but also to evaluate it. The last missing part is an algorithm that can estimate a step to a model that further minimises the loss. One could certainly do this randomly until the network finds a very low costed decision rule if infinite computational power was provided, but that would neither be an efficient nor the desired learning process.

The output of each node in the final layer is defined by the weighted and biased information of the previous nodes, and lastly the activation function. For one connection, there are three variables that have impact on the loss, the weight, the bias, and the activation. Summarising this information for all nodes in a vector defines the *loss-vector*. The gradient of this vector is an estimator for the impact of each parameter on the overall loss and thereby gives a preferred direction for the model. Updating a network's parameters based on this gradient is called *backpropagation*. The algorithm works as follows:

1. A certain set of input variables is iterated through all layers of a network resulting in an estimator $\hat{y}$ at each output node.
2. The sum of deviations from the true value at all output nodes is determined as the loss C of the setup.
3. The gradient of the loss is calculated as the partial derivative of all network parameters using the following equation:

$$\frac{\partial C}{\partial a_k^{L-1}} = \sum_{j=1}^{N} \frac{\partial z_j^L}{\partial a_k^{L-1}} \frac{\partial a_j^L}{\partial z_j^L} \frac{\partial C}{\partial a_j^L}.$$

4. The parameters are then updated backwards through the layers following the negative loss gradient.

This backpropagation algorithm is the backbone of the neural network's learning process. The decision step based on the gradient defined above is specified by the networks optimizer and deserves a bit more attention.

There are different choices of optimisers, which try to accommodate different problems as well as some important parameters to tune for an effective training. The length of a learning step

has to match the problem's topology to properly let the model converge. First, we define the gradient, g, in a more general way. The batch size, m, stands for the amount of data processed to evaluate the next step. f is the network for a current configuration or model, θ, and the prediction $\hat{y}$. θ summarises all the parameters optimised by the network during the training. The truth label is y. Using these definitions the gradient becomes:

$$g = \frac{1}{m} \nabla_\theta \sum_j L(f(\hat{y}^j; \theta), y^j). \tag{5.4}$$

The configuration θ is then updated using the gradient and a multiplicative constant, η, called the *learning rate*, which determines the step size for each update following:

$$\theta\prime = \theta - \eta g. \tag{5.5}$$

Optimisation processes like these are gradient descent based optimisers and can be considered the basis of all optimisers. Depending on the choice, they might be based on the whole training sample or just a mini batch of the sample. The most basic form has only learning rate as its hyperparameter. A good learning rate should be small enough to avoid oscillations around minima but high enough to approach a minimum efficiently. A good estimate is given by the Robbins Monro condition:

$$\sum_k \eta_k = \infty, \tag{5.6}$$

$$\sum_k \eta_k^2 < \infty. \tag{5.7}$$

As the choice of learning rate will not be perfect for every part of the problem's topology, *momentum*, ν, can be introduced as a second parameter to the optimiser [29]. The effect desired is twofold; momentum should increase the learning rate in the direction of the minimum and lower it when approaching said minimum. Momentum scales each step by how aligned previous steps were, meaning it will allow avoiding local minima or moving slowly along a slope. This is accomplished by enlarging steps at the beginning of the training but diminishing them at the end close to the minimum. It promises to speed up the training with less risk of large oscillations, which is an effect of high learning rates. Momentum also takes a single scaling hyperparamter α and is updated each step in the following way:

$$\nu' = \alpha\nu - \eta \frac{1}{m} \nabla_\theta \sum_j L(f(\hat{y}^j; \theta), y^j), \tag{5.8}$$

$$\theta' = \theta + \nu'. \tag{5.9}$$

Alternatively one can use Nesterov momentum [29] which is a more advanced adoption of momentum and updates the step a further time after applying the gradient:

$$v' = \alpha v - \eta \frac{1}{m} \nabla_\theta \sum_j L(f(\hat{y}^j; \theta + \alpha \times v), y^j) \tag{5.10}$$

$$\theta' = \theta + v' \tag{5.11}$$

Finally it can be helpful to decrease the learning rate of the network stepwise while approaching a minimum to avoid oscillations or even missing the minimum completely. This can be accomplished by the hyperparameter of learning rate decay. It just decreases the learning rate in each iteration t by a small hyperparamter ϕ following the assumption that smaller steps are sufficiently close to the minimum. [29]

$$\eta' = \frac{\eta}{1 + \phi t} \tag{5.12}$$

Adaptive optimisers

In addition to the merely gradient based optimisers there are adaptive optimisers. Learning rate and momentum as previously described are difficult to tune to every part of the training process as the topology of the problem might rapidly change. Therefore adaptive optimisers update their parameters based on the training process. There is a set of adaptive optimisers. [29] Often the adaptive optimiser of choice is ADAM. [30] ADAM updates both, its learning rate and its momentum over the course of the training based on an exponentially decaying average of past gradients and past, squared gradients. The average makes sure that the parameters keep getting updated based on past steps. They should be decaying as otherwise the parameters would rapidly shrink. The decay of the averages is defined by a hyperparameter β.

$$\hat{g}^2 = \frac{\sum g^2}{1 - \beta_1^t} \tag{5.13}$$

$$\hat{g} = \frac{\sum g}{1 - \beta_2^t} \tag{5.14}$$

The model's parameters are then updated according to:

$$\theta' = \theta + \frac{\eta}{\sqrt{\hat{g}^2} + \epsilon} \hat{g} \tag{5.15}$$

ADAM is often considered a very good algorithm as it contains many corrections to hyperparamters during the training and thereby allows for easier optimisation but it also needs more computational power.

5.3 Regularisation and Optimisation

Fluctuations and noise in the training sample can be a big problem for a model trained on the sample as a neural network might pick noise and random fluctuations up as features of its decision rule which basically is the process of overfitting.

The network observes way more features to work with than those actually present in reality or even in a different test sample. The most extreme scenario is that your network is large and deep enough to pick up every single feature in the training sample. If that happens the training error becomes very low and indicates a very good decision rule. For a different sample this decision rule is at most very unreliable but probably strictly wrong resulting in a high test sample error. The network just picked up and remembered every single feature in the training sample instead of general correlations and thus becomes a mask of the sample.

A possible way to solve this issue is to stop the training early or to find a good way when to stop the training. This way noisy features might not yet have been picked up by the training. This might in turn also lead to a suboptimal result of the overall training as one cannot be sure that the correct features always get picked up first.

More sophisticated approaches are called regularisations of a neural network. The most commonly used solution is a so called Dropout layer described in the subsection 6.6.1. Additionally a batch normalisation can have an effect of regularisation and is therefore introduced in this section as well.

5.3.1 Dropout

Dropout can be introduced as an additional layer attempting to hinder the network from relying on less dominant features too much by removing different nodes in each iteration. This forces the network to build models that are not based on strong correlations between nodes making the weights become less interdependent. To simplify it significantly it means training several neural networks depending on which nodes are turned on during a training epoch which it keeps the training in motion for a large number of epochs.

Dropout is added to each layer of a network and can also be restricted to a subset of layers. It slows down the training as the additional motion slows down the process of finding a minimum but it also accelerates each epoch slightly as it simplifies the network architecture.

5.3.2 Batch normalization

In supervised learning the training result is strongly dependent on the set of data the network is trained on. This means that the performance might change dramatically when the test data is very different. Imagine a classifier distinguishing between pictures that show cars and pictures that do not show cars. If the training set contains predominantly green cars the colour green might end up as a strong indicator for the classification car. In general the colour green will not be as dominant and the network will perform slightly worse when trying to classify cars of a different colour. Formally such a change of input is called a covariance shift.

A way to reduce the effect of covariance shift is batch normalization. The output of nodes in general and the weight of a connection in a neural network is not necessarily limited allowing

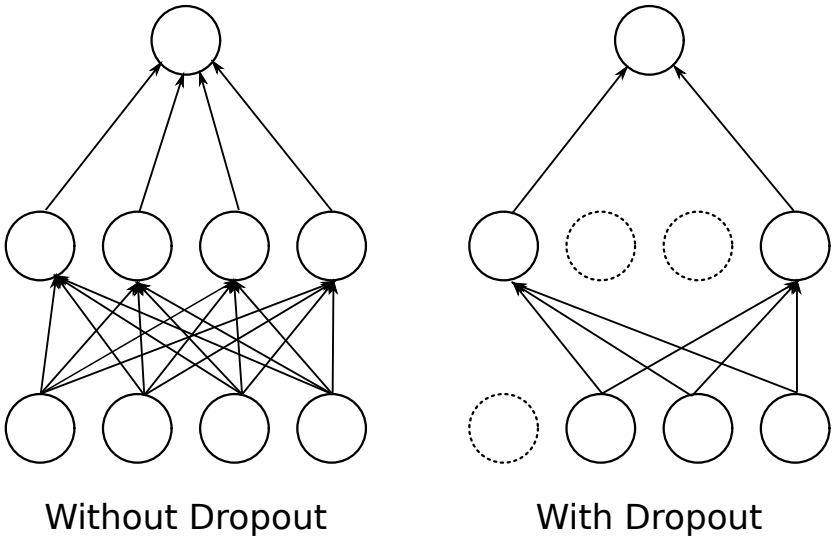


Figure 5.2: Sketch of network before and after the inclusion of dropout. On the left hand side dropout is not applied and all nodes are connected. On the right hand side the dashed circles are nodes excluded by dropout and therefor not connected to the other nodes.

for certain connections to be really dominant and overshadowing less dominant features. We want to avoid this as the dominance of some features might just be present in the training sample. This can be achieved by normalizing the output of each layer in the network to the total output and thereby minimising the effects of strongly overrepresented features. This is done by normalizing each output to the mini-batch mean μ_B and the mini-batch standard deviation σ_B^2.

$$\mu_B = \frac{1}{m}\sum_i x_i \tag{5.16}$$

$$\sigma_B^2 = \frac{1}{m}\sum_i (x_i - \mu_B)^2 \tag{5.17}$$

$$x_{i,norm} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \tag{5.18}$$

5.4 Adversarial Neural Networks

This main part of this work is the examination and training of an adversarial neural network. An adversarial neural network consists of a classifier and a second network that tries to regularise the output of the first classifying network.

In this section the concept of an adversarial neural network is motivated and the underlying mathematics as originally stated in paper [31] are presented. For more information about an approach directly tested on physics see the paper "Learning to Pivot with Adversarial Networks". [32]

5.4.1 The adversarial neural network

Neural networks have been very successful in classifying tasks but less successful for generative tasks. This was the original problem that gave birth to the idea of a generative adversarial network. Generative networks often produce output that is very easy to distinguish from real samples. The solution suggested is adding a classifier that tries to distinguish between generated samples and real samples. As long as this adversary is able to accomplish this task the first network fails at its generative task. Training the two networks against each other disincentives the generative network from using the features not dominant in real samples.

In this work the first network is not a generator but a classifier separating signal events from background events in a Monte Carlo simulation. These simulations contain systematic uncertainties and different samples represent a set of plausible data generation processes. The classifier should not be too dependent on variables with high systematic uncertainties as the network cannot account for differences in the training and testing sample or even in real data. If the classifier has these strong dependencies on systematic uncertainties it might lead to a high co-variance shift.

Instead of a generated sample and a truth sample, a so called nominal sample and systematic samples are used as input for the second network. Systematic samples have slightly different distributions than the original samples because of changes to the variables with the systematic uncertainties. Training the second network on determining whether it is looking at a nominal or a systematic sample allows to estimate how strongly the model depends on variables with high systematic uncertainties. Training the classifier against the adversarial network promises to reduce the effect of systematic uncertainties on the model. If the topology of the problem allows for it this should render the model generated by the classifier pivotal. That means it does not depend on the unknown values of the nuisance parameters.

Mathematically this comes down to a minimax decision rule or a competition between two neural networks. Let us call the classifier $Net1$ and the adversary $Net2$ then the problem becomes:

$$\min_{Net1} \max_{Net2} V(Net1, Net2) = \mathbb{E}_{x\sim\rho_{data}}[logNet1(x)] + \mathbb{E}_{z\sim\rho_{sys}}[log(1 - Net2(z))] \tag{5.19}$$

$V(Net1, Net2)$ is the combined value function for the two adversary networks. The first network is trained to be an optimal classifier represented by $logNet1(x)$ while the second network is trained to distinguish between the nominal and systematics distribution $z \sim \rho_{sys}$ represented by $log(1 - Net2(z))$.

In theory the first classifier should be trained slowly and kept close to its optimum while the second network slowly learns and allows the first network to adapt to it. This is achieved by training the two networks successively over multiple iterations using a combined value function. As this value function a combined loss function is used. It is just the difference between the two separate loss functions with a hyperparameter λ to control the impact of the adversary as shown in equation (5.20).

$$\mathcal{L} = L_{net1} - \lambda L_{net2} \tag{5.20}$$

In the first step of each iteration the first network is trained using the combined loss function $\mathcal{L}$. In the second step the second network is trained using its simple loss function λL_{net2}. Each of the networks has the usual set of hyperparameters to optimise explained in detail in the previous sections missing.

In this work the adversarial network is set up by building a classifying network. The information of the classifier is then fed into both the classifier's output layer and the adversarial network creating a second model based on the first network's model. The networks are then trained successively controlling the combined and separate losses. Figure ?? shows a sketch of the setup.

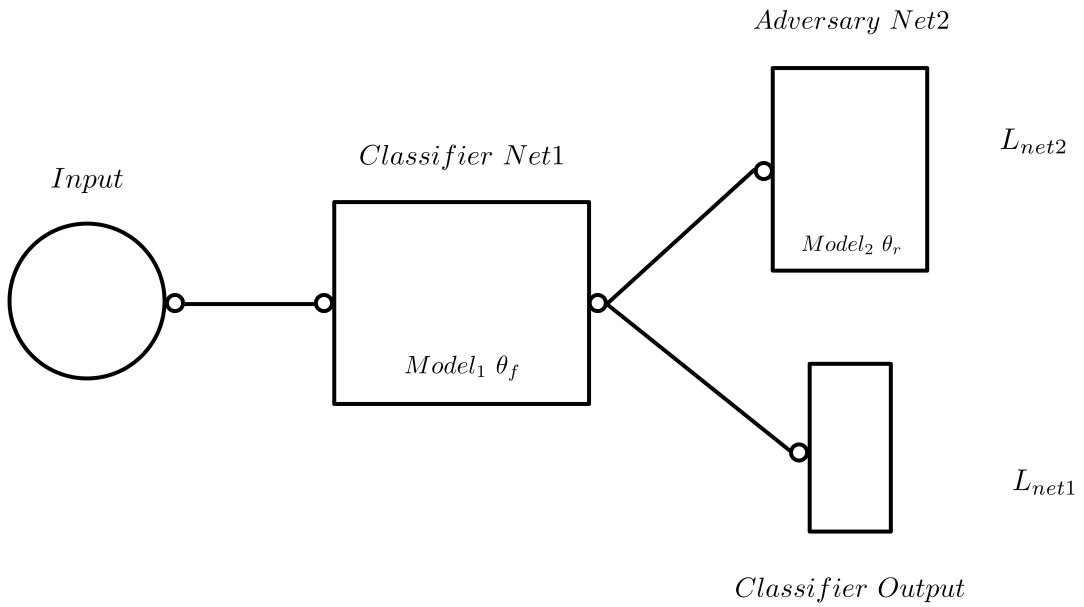


Figure 5.3: Sketch of the network setup. The main part is the classifier. The model generated is fed into both the second network and the classifier's output. This way the two losses L_{net1} and L_{net2} are generated. Furthermore the training of the classifier immediately affects both output models.

The hyperparameter λ is set to tune the impact of the second network on the model. A large λ leads to a very pivotal model but can also decrease the overall quality of the classifier.

In order to better understand the figures shown in chapter 7 the three loss values of an adversarial neural network will be introduced as the last part of this chapter. The first loss belongs to the classifier and is expected to first increase as the adversary finds a good model and to decrease once a more pivotal model is found. The second loss displays the adversary's performance ideally decreasing at first to then increase and saturate as a pivotal model renders it impossible to extract any information. Lastly the combined loss is displayed showing the overall performance and decreasing as good models are found for both the classifier and the adversary. Figure 5.4 shows an example for this loss functions taken from the paper "Learning

to Pivot with Adversarial Networks". [32]

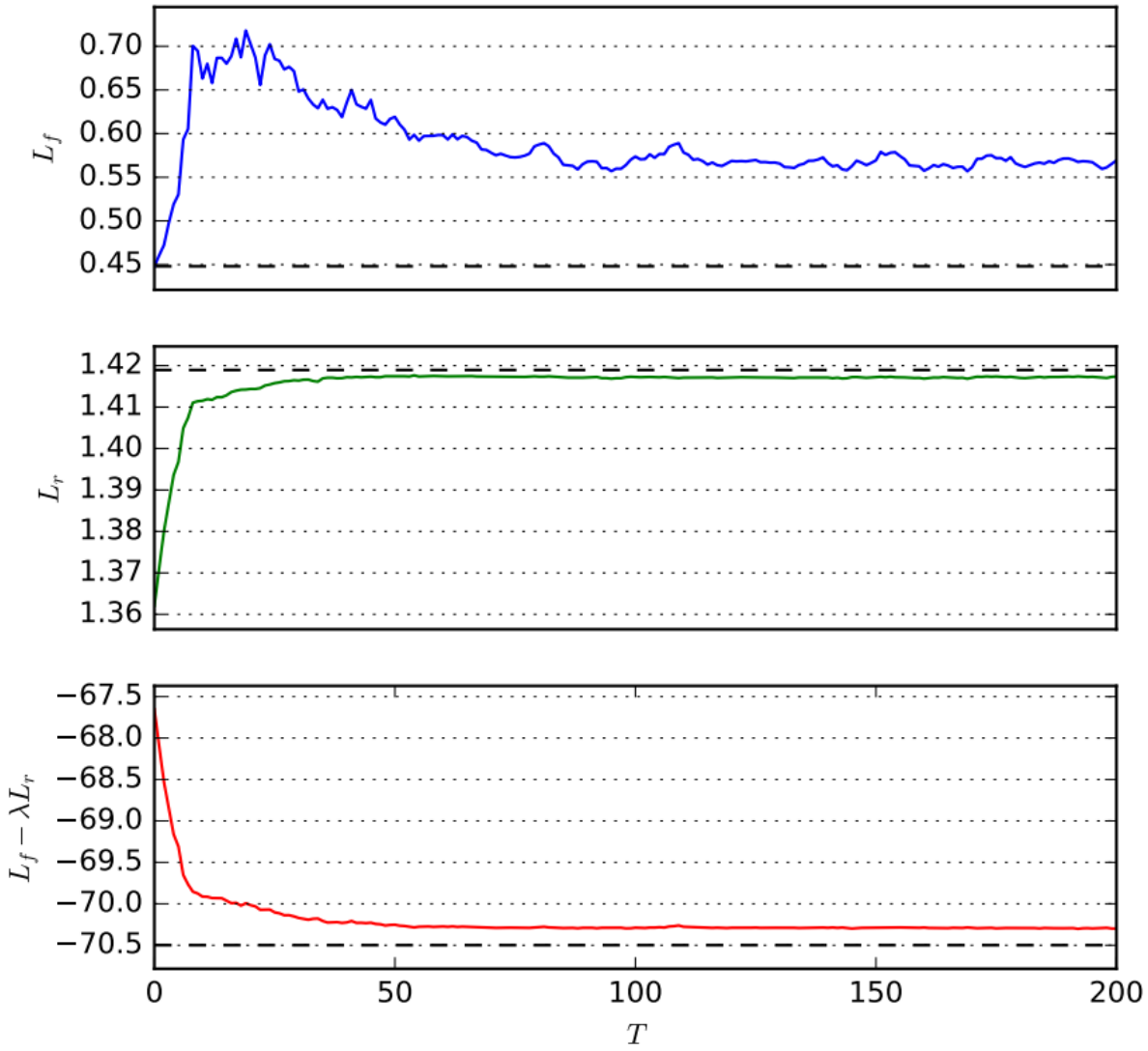


Figure 5.4: Three losses of an adversarial network for $\lambda = 10$ taken from a toy example. [32] From top to bottom the classifier, adversary and combined loss is presented. T is the number of training iterations.

CHAPTER 6

Hyperparameter optimisation of a classifying neural network

The adversarial neural network is based on a common classifying neural network trained on signal/background separation. The output of that network is then used as input for the adversary to create a negative back-feed for the classifier. Before the adversarial, second network is added, the classifying network is optimised on its own to make sure that its setup is sufficient for the classification task. During the adversarial training this setup can be updated if the structure is not optimised for the additional task of a model less sensitive to systematic uncertainties.

This chapter describes the hyper-parameter optimisation of the first network starting with the motivation of the input information. The second section explains the choice of the architecture followed by the step-wise setup of the optimiser. Lastly regularisation of the network is tested and described. The overall aim of the description is to provide some understanding on the hyper-parameters available and their correlations. The impact of the hyper-parameters on the training results is presented not only to introduce their function but also to motivate possible solutions later on.

To train the network and to test its performance all its parts need to be in place. Therefore all hyper-parameters had to be initialized with values assumed to represent a good setup. Initially this was accomplished by starting with a very simple network using a minimal set of hyper-parameters which were then elaborated to a more and more optimised network. The results shown in this work are going to be based on the final choice of hyper-parameters where only one parameter is then varied at a time to explain the impact of the particular tuning. For that reason section 6.2 will already introduce the final network structure which is then step by step to be explained and motivated.

The structure was achieved within reason and the computational power constraints. There certainly are hyper-parameters that deserve more attention and in addition to that there are alternative setups for the whole network that were not tested. A further investigation of network optimisation is without a doubt very promising.

6.1 Technical details

The artificial neural networks in this thesis were created using the Keras python library [29]. Keras is an application programming interface written in python and able to run on Tensorflow, CNTK or Theano. It was developed by google and summarizes the necessary calculations for running a deep neural network training in fast and easy modules. The backend is the package responsible for the underlying vector calculations needed for the network setup and training. In this work the Tensorflow package was used as a backend [33].

6.2 Final setup of the network

This section describes the fully optimised classifier which is then parameter-wise varied during this chapter to motivate the particular choices. The hyperparamters are listed and the loss curve, the ROC curve and the final separation are introduced as they are the main tools a training performance is evaluated by in this thesis. For the sake of completeness the agreement between nominal and systematic response is also shown as it will be a standard inclusion for the plots during the adversarial training.

- Input: 14 variables motivated by a BDT variable scan.
- Hidden layers: 6 ELU layers $\times$ 128 nodes each
- Output layer: 1 SIGMOID node
- Optimisation: SGD, LR $= 0.06$, momentum $= 0.3$, no nesterov, no decay
- Duration: 600 epochs

6.3 The input variables

There were two sets of input parameters tested for the classifier. The first one is a set of simple kinematic variables. This is tested to exploit a neural network's ability to deduce all further information from the complete basis of a system. The second set of variables uses more complex variables based on the most significant variables for a boosted decision tree usage on the same problem. The variable sets used were:

Figure 6.2 shows a comparison of the separation and the loss curves for both variable sets. It is visible that for the set of simple variables the network runs into overtraining after about 200 epochs 6.2(d) because $Loss_{train}$ and $Loss_{test}$ start to diverge. This also results in a bad separation with a strong disagreement between the two samples. There is an argument for testing a simple set of variables for a shorter training duration. However, this optimisation process would be a project of its own. Furthermore, in this work one has to keep in mind that the network is supposed to be used in adversarial setup later. This demands for a longer training period and a large number of good features to be available. Otherwise the probability of finding a

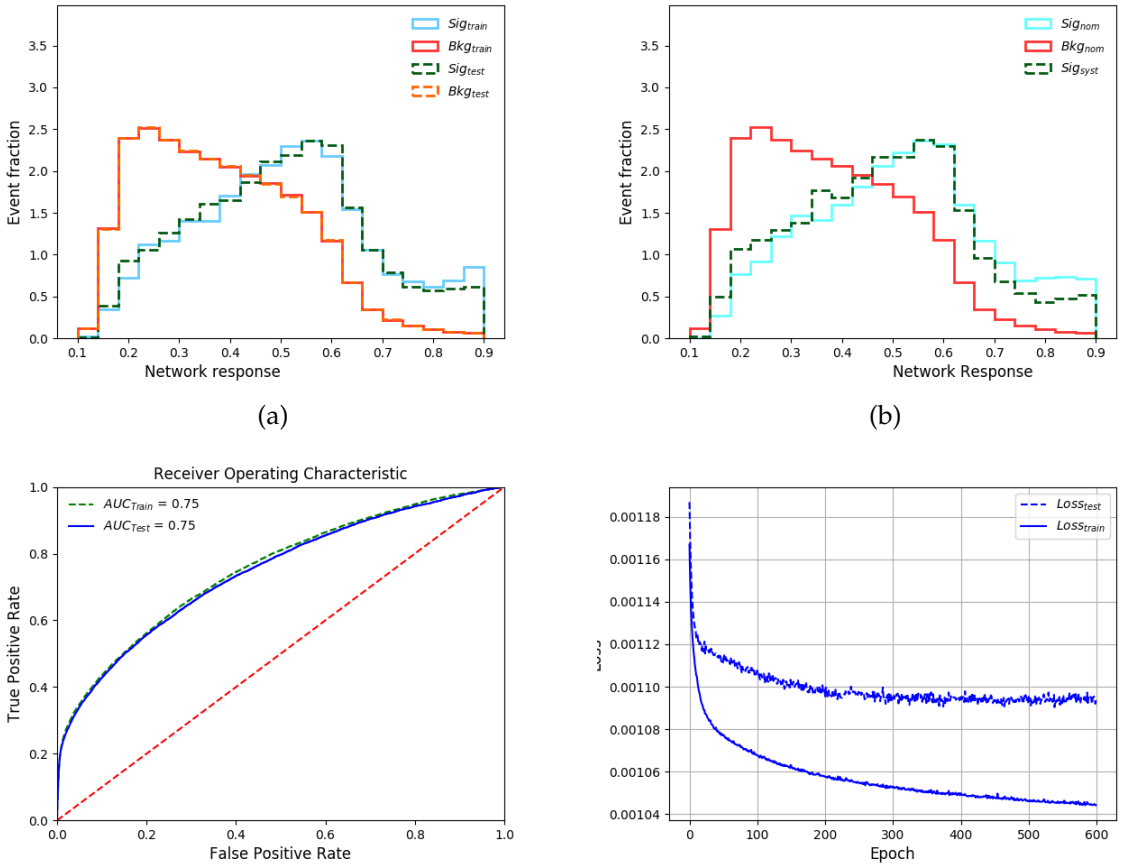


Figure 6.1: Figure (a) shows the response of the classifier. The solid lines represent the training sample and the dashed lines the test sample. Figure (b) shows the difference for response in systematic and nominal samples. The solid lines represent the nominal samples and the dashed line the systematic sample. Figure (c) shows the ROC-curve. The dashed line is the training curve and the solid line is the test curve. Figure (d) diagrams the losses with dashed for test and solid for training.

Table 6.1

Variable	Branch
$p_{\mathrm{T}}^{\mathrm{miss}}$	met
$\lvert\eta_{jet1}\rvert$	absEta_Jet1
$\lvert\eta_{jet1}\rvert$	absEta_Jet2
$\lvert\eta_{lep1}\rvert$	absEta_Lep1
$\lvert\eta_{lep2}\rvert$	absEta_Lep2
$p_{\mathrm{T}\,jet1}$	pT_jet1
$p_{\mathrm{T}\,jet2}$	pT_jet2
$p_{\mathrm{T}\,lep1}$	pT_lep1
$p_{\mathrm{T}\,lep2}$	pT_lep2
ϕ_{jet1}	phi_jet1
ϕ_{jet2}	phi_jet2
ϕ_{lep1}	phi_lep1
ϕ_{lep2}	phi_lep2

Table 6.2

Variable	Branch
	mass_lep1jet2
	pTsys_lep1lep2met
	pTsys_jet1jet2
	mass_lep1jet1
	deltapT_lep1_jet1
	deltaR_lep1_jet2
	deltaR_lep1lep2_jet2
	mass_lep2jet1
	pT_jet2
	deltaR_lep1_jet1
	deltaR_lep1lep2_jet1jet2met
	deltaR_lep2_jet2
	cent_lep2jet2
	deltaR_lep2_jet1

classifier less sensitive to a systematic uncertainty decreases and the behaviour of the losses has to supervised very carefully.

Showing an overall good performance the set of complex variables as suggested by the boosted decision tree variable scan performed in the xy analysis was chosen. Please note that the set of variables in the paper referred to does not fully agree with the variables used in this work. The reason being that the variables were adopted while the research was still ongoing.

6.4 The network architecture

The architecture of the neural network is formed by its nodes and layers. The choice of the architecture is nontrivial and, as a lot of aspect's of machine learning, not an exact science. However, one can make some assumptions about the appropriate architecture. First of all the complexity of the model should about match the complexity of the task assigned. Although it usually is not trivial to find an estimator for a task's complexity and even less to match it to a certain architecture, a test series often leads to a good estimate. Another possible thought is the amount of variables necessary to fully describe a system resulting in the minimum variable number necessary to input in the network. This also gives a first estimate on how large the architecture should at least be. Both the depth and the overall size of the model play a role. A simplified way of explaining these two properties is by saying that the depth defines how often the input is processed while the number of nodes is the number of features that can be kept during each step of processing.

In general an architecture that is too deep and wide will pick up too many features too fast and overtrains before it gets to a good minimum. This can be seen in an early divergence between the training loss and the validation loss. An architecture too simple is not able to pick up the features of the task at all resulting in no learning process. The loss stays constant or

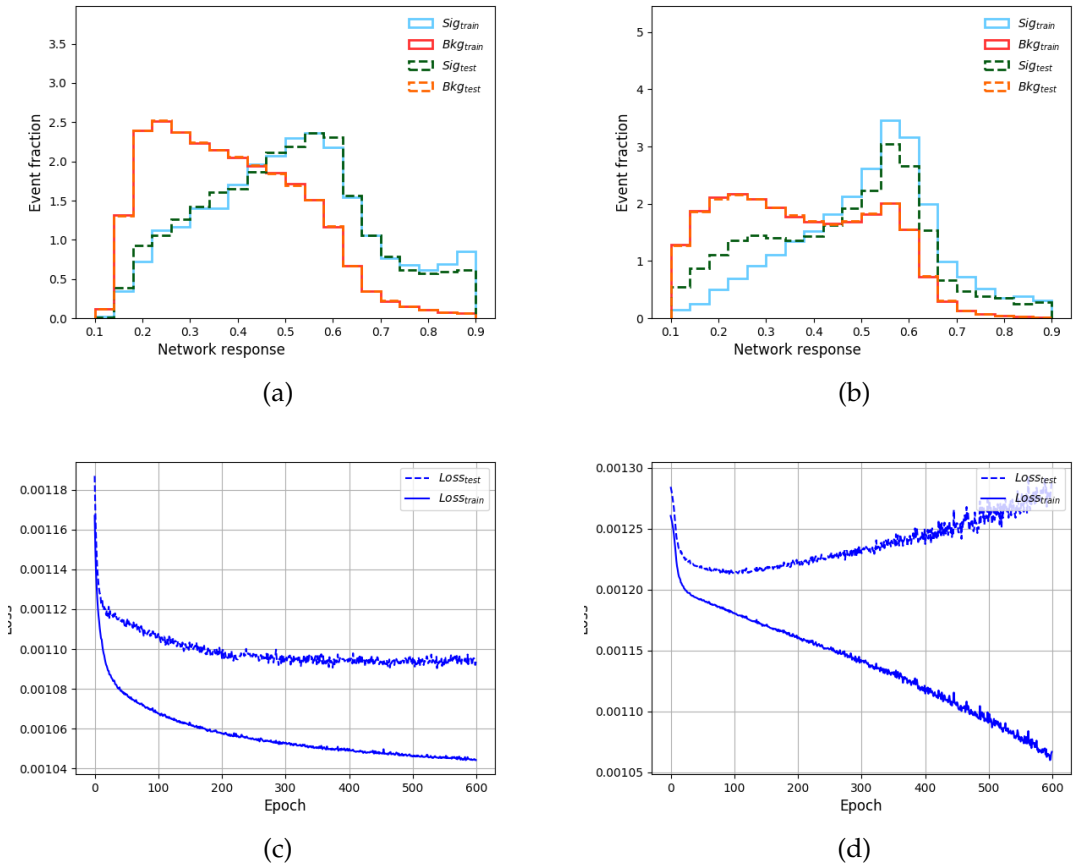


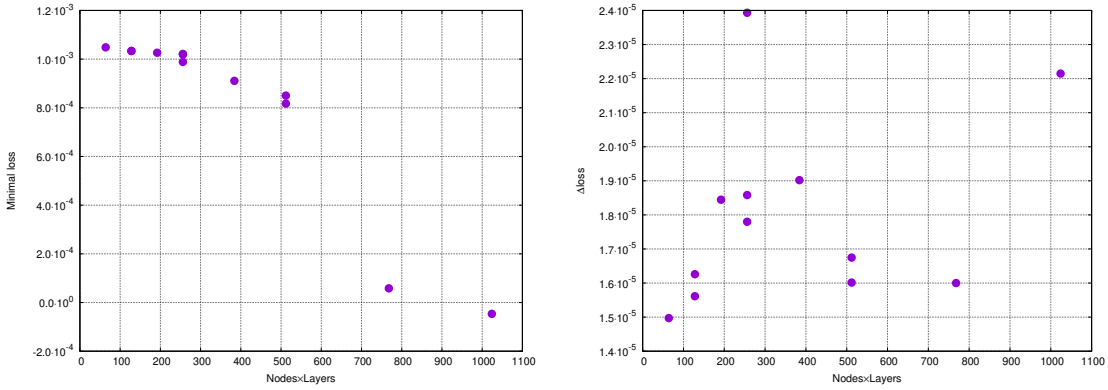
Figure 6.2: Figure (a) and figure (b) show the separation for the complex and simple variables respectively. The losses for the variable sets are diagrammed in figure (c) and figure (d).

changes very slowly.

In this work a test-series was performed, training a network for a wide range of combinations of nodes and layers. For the sake of simplicity the number of nodes per layer was kept constant during each training. Two variables were then plotted against the size of the architecture. First the overall smallest loss the model achieved during the training was plotted. The other variable was the minimal difference between the training and the validation loss. To keep it simple the complexity of the architecture was defined as the product of nodes and layers. These are certainly not the most sophisticated indicators for the model's complexity and its performance. However, the plots do allow for some sophisticated guesses for a good choice of architecture.

Figure 6.3(a) shows that more complex architectures also achieve smaller loss values. This unfortunately is not a clear sign for an overall good performance as heavily overtrained networks will achieve low loss values as well. Therefore, a second estimator was taken into account. Figure 6.3(b) shows the minimal difference between $Loss_{train}$ and $Loss_{test}$ achieved during the whole training process. There are two regions showing a small difference. The first is the region of very simple architectures. This however, is a result of a slow learning process. No features are being picked up that differentiate between the two sets used for loss evaluation.

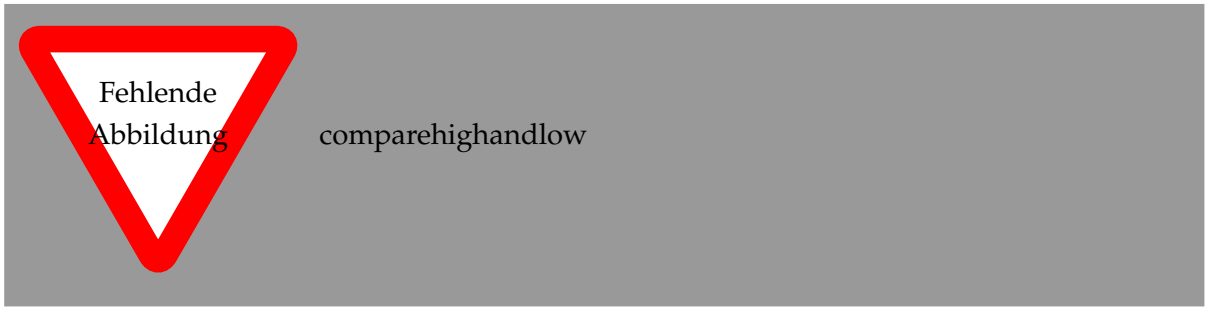
The second region, 500 to 800, was used as the region of choice before heavy overtraining occurs. Finally, a more complex architecture in this region, comprising 6 layers with 128 nodes each, was chosen. Although this choice seems fairly arbitrary at this point, there is arguments for a complex architecture as it can always be regulated using dropout if need be.



(a) Minimal loss

(b) Minimal loss difference

Figure 6.3: Figure (a) shows the minimal overall loss achieved during the training for the range of architectures. Figure (b) shows the minimal difference between $Loss_{train}$ and $Loss_{test}$ achieved.



In addition to the choice of the architecture the activation function for the layers has to be set. Two different activation functions come to use in the neural network. The main function connects the nodes in the hidden layers while the last one converts the output to a value between 0 and 1. For the output the sigmoid function is a natural choice and no other activations were tested. For the hidden layers ELU and RELU were tested. Figure 6.4 shows a comparison of the ROC-curve and the loss development for both activation-functions. For RELU a strong disagreement between training- and test-sample occurs and ELU became the activation function of choice. Further activation-functions were not tested due to time constraints.

Furthermore the activation function for the hidden layer is elu the activation function

6.5 Setup of the optimisation

For the most part of the optimisation SGD was used as the optimiser and its tuning is shown in detail. Adam was also tested out but not fully tuned due to constraints on time and computational resources. In subsection 6.5.1 a brief overview of the performance using Adam is given. Section 6.5.2 describes the fine tuning of the SGD optimiser in more detail.

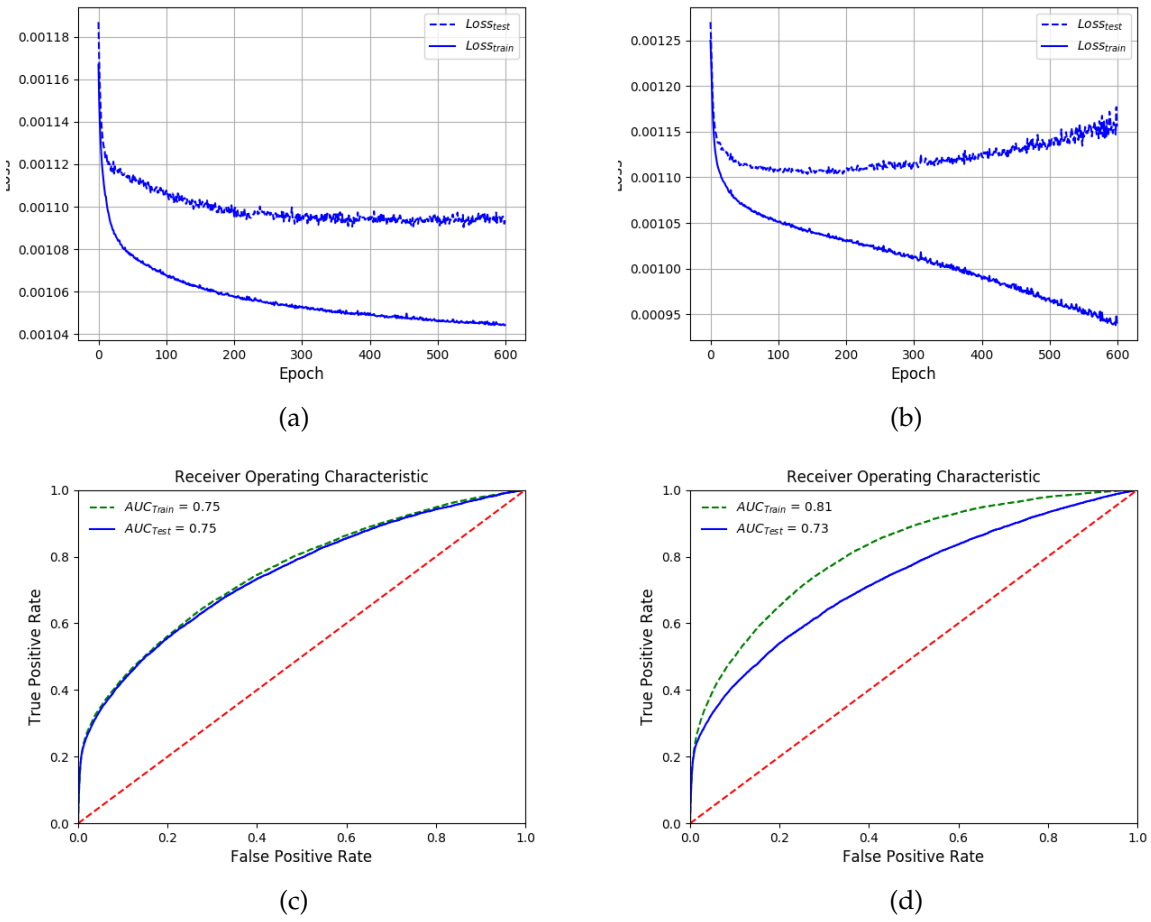


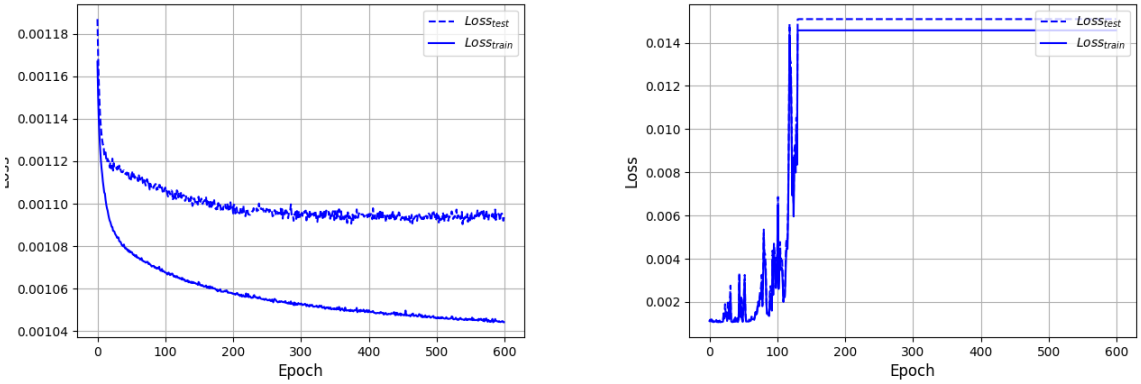
Figure 6.4: Figure (a) and figure (b) show the losses for SGD and Adam respectively. The ROC-curves for the different activation-functions are diagrammed in figure (c) and figure (d).

6.5.1 Choice of the optimiser

A classic gradient-based optimiser, SGD, is compared to adam as an adaptive optimiser. Figure 6.5 compares the loss development for both optimisers. It is obvious that adam behaves strangely and it was therefore discarded. It is possible that adam could outperform SGD for a better tuning of its hyper-parameters and it certainly is worth testing out if the computational power is provided.

6.5.2 Tuning the optimiser

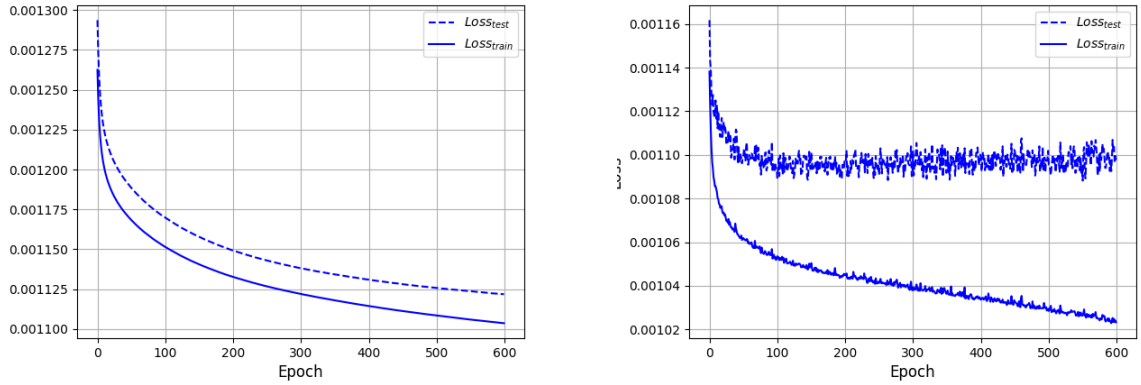
As described in section5.2 an optimiser has several hyper-parameters of its own. In this section the learning rate, the momentum with the option of Nesterov, and the decay parameter will be probed. For the learning rate a range of values between a small learning rate of 0.001 and high learning rates up to 0.5 were tested to investigate the behaviour. The choice was based on how high the oscillations are and how efficiently the training converges. High learning rates were



(a) (b)

Figure 6.5: Loss curves for SGD (a) and adam (b).

discarded for an increased probability of overtraining and a lot of unwanted oscillations while small learning rates made the training unnecessarily slow and inefficient. Figure 6.6 shows a comparison of the network losses for a small learning rate of 0.001 6.6(a) and a relatively large learning rate of 0.2 6.6(b). For the small learing rate there are no signs of overtraining and no oscillations visible. However, the training converges slowly. For a high learning rate the $Loss_{train}$ and $Loss_{test}$ show larger divergence and each curve shows more oscillations individually. In the end an intermediate learning rate of 0.06 is chosen.



(a) (b)

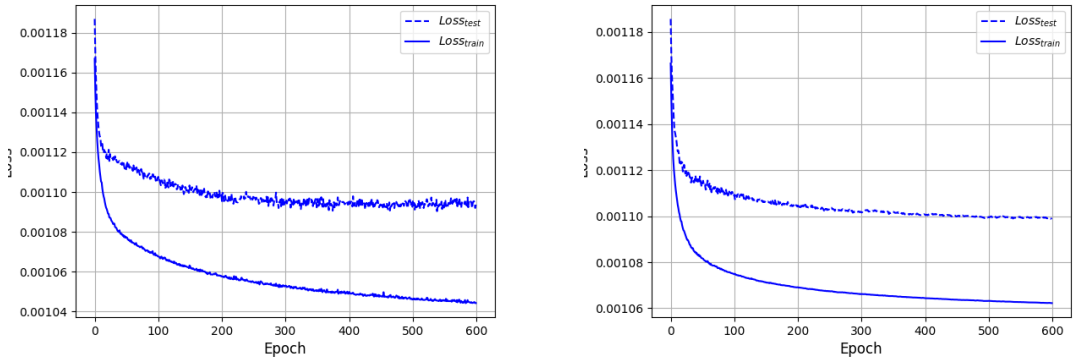
Figure 6.6: Loss behaviour for learning rates of different scales. Figure (a) shows losses for 0.001 and figure (b) for 0.2.

In this setup dropout is used. That makes it a bit more difficult to see the clear signs of overtraining. For a plot that shows it see section 6.6.1.

Like the learning rate the momentum was scanned over a wide range of values. Again he risk of overtraining and the amount of oscillations was evaluated against the overall training

efficiency. This resulted in a final value of 0.3. In addition, the test of using Nesterov momentum did not lead to any significant effects. It was therefore not utilized.

The decay parameter was tested to see its effect on the loss behaviour. Figure 6.7 shows the standard loss behaviour compared to losses for a decay value of 1×10^{-6}. When decay is used the losses 6.7(b) become slightly more stable and plateau a bit earlier and for a higher value. Overall no significant improvement is visible. For that reason decay is not included in the final setup of the classifier. It was kept as an option for the later combination of the classifier and the adversary where it might help keeping the model close to the optimum while slowly updating to a model less sensitive to systematics.



(a) (b)

Figure 6.7: Comparison of usual loss behaviour and loss behaviour including a decay paramter. Figure (a) shows the losses without decay. Figure (b) shows the losses for a decay parameter of 1×10^{-6}

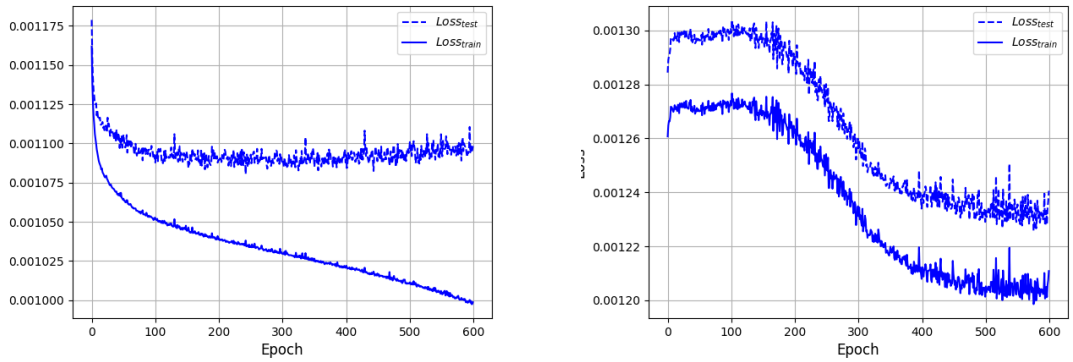
The final setup for the optimiser is:

6.6 Regularisation

The two regularisation tools tested were dropout-layers and batch-normalisation layers. Dropout has the main purpose of regularising a network while batch-normalisation focuses on keeping all outputs on the same order which indirectly also helps the regularisation of the network. Dropout is tested for a range of values defining the percentage of nodes turned off per iteration step. Batch-normalisation is just tested in the default setup to investigate its effects.

6.6.1 Dropout

A dropout layer is added to each hidden layer of the network. Figure 6.8 shows the impact of dropout on the behaviour of the losses. For only 1 percent dropout compared to the default 10 percent overtraining occurs relatively early. For a very high dropout of 80 percent the losses behave strangely. The curve becomes more linear and converges less efficiently. In conclusion, dropout is a valuable addition to the network. It makes changing the architecture and the optimiser more pardoning and is a simple way to adjust a network without rearranging all hyper-parameters.



(a) (b)

Figure 6.8: Comparison of usual loss behaviour and loss behaviour including high dropout. Figure (a) shows the losses for a dropout of 0.01. Figure (b) shows the losses for a dropout of 0.8

makes clear that full structure is very interdepndent
overtraining very visible
more linear and less focused

6.6.2 Batch Normalisation

CHAPTER 7

Adversarial Neural Network

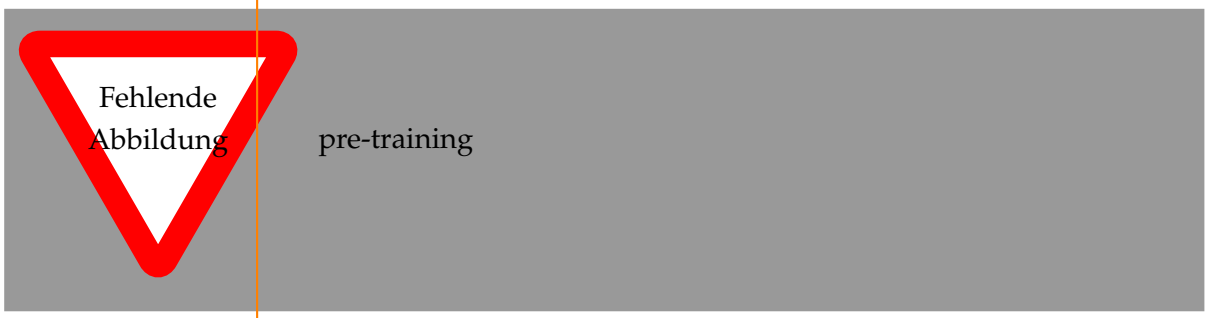
In this chapter the setup and training of the Adversarial Neural Network is described. The Network did not achieve the desired results in its initial configuration. For this reason different configurations for the implementation of the network structure are presented in addition to the hyper-parameter investigations. In total three main approaches to the adversarial neural network were tested. The first one uses the understanding earned from training the classifier and the original approach introduced in the paper "Learning to pivot with Adversarial Networks" [32]. The second approach uses a hidden layer as input for th adversary instead of using the output of the classifier. Lastly the information of the hidden layer is compressed to a smaller layer before being transferred to the adversary.

The first section of the chapter focuses on the initial run of the ANN using the base network presented in chapter **??**. The results are investigated and the hyper-parameters are adapted using an initial setup for the second network. The problems with this setup are explained and possible reasons are listed. The second section describes the approach of using hidden layers as input. The results are compared to the initial approach and conclusions are drawn. The third approach deals with further adapting the information from the hidden layer and is described in the third section. Lastly the approaches are compared and possible further steps are listed.

Prerequisites

The technical details for the setup are the same as for the classifying network as introduced in section 6.1. The adversarial setup uses the classifier trained and tuned in chapter **??** as a basis. For the classical setup this output is used as input for the adversarial network.

In general both the classifier and the adversary are pre-trained. That means they are trained on their tasks without using a combined loss-function to generate better starting conditions for the adversarial training iterations. In addition to that, observing the losses of the networks during the pre-training indicates whether the networks influence each other.



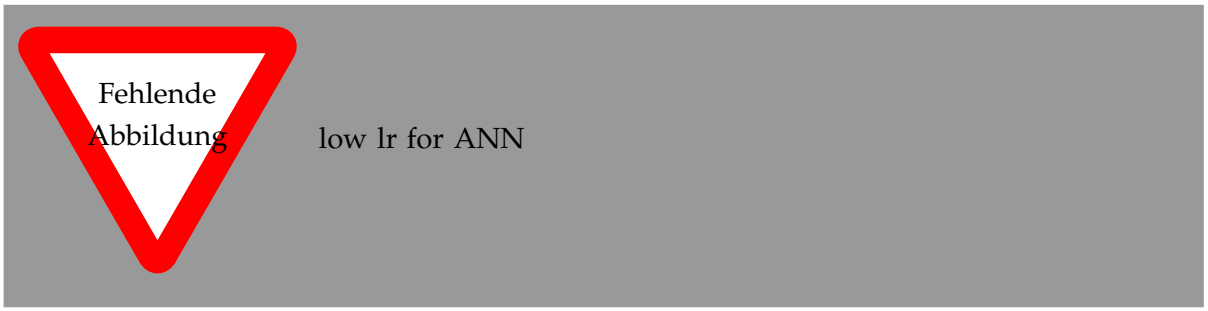
7.1 Approach I: classical neural network

Show results

The initial setup uses the classifier as trained in chapter 6 and an adversary inspired by this structure. Optimisation and activation stayed the same while the architecture was simplified to factor the one-dimensional input into the system. It becomes clear that the optimised hyper-parameters for the classifier are not suitable for the task of training an adversarial neural network. This is an important insight because it means that the adversary is not something one can just add to a model to improve certain features. Instead, it is an integral aspect of a whole training procedure of its own. The next steps taken during the analysis presented here try to investigate further what the performance of the combined network is strongly correlated with and how to understand what the network suffers from.

A hyper-parameter scan focused on the optimisation was performed. The results for a very low learning rate and a non-existing momentum are shown indfsfesefef. With a slow optimisation the training becomes more stable and both networks start learning. This is assumed to be due to the fact that in an adversarial training not just a minimum is sought. Instead, the classifier is first moved close to a minimum and then supposed to stay close that model of decent classification while also slowly adapting to features less sensitive to the systematic uncertainty.

Unfortunately even with these updated hyper-parameters no model is found that is sufficiently independent on systematics. On its own both networks improve their performance but in combination the classifier does not find a model that significantly downgrades the performance of the adversary. The worst case scenario is that this is due to no such model existing. It is very well possible that for this particular task there is no training that is independent of systematics. A simple change would be the set of input parameters. However, this would cause a whole new training problem. For that reason another aspect, the input of the adversary, was looked into. At this point the adversary is just feed the sigmoid-output of the single, final node of the classifier. As mentioned earlier the amount of variables and the complexity of the architecture should be scaled with the task. Possibly the single output is enough to see a systematic dependency but not sufficient to improve the model using this insight. Approach II is based on this thought.

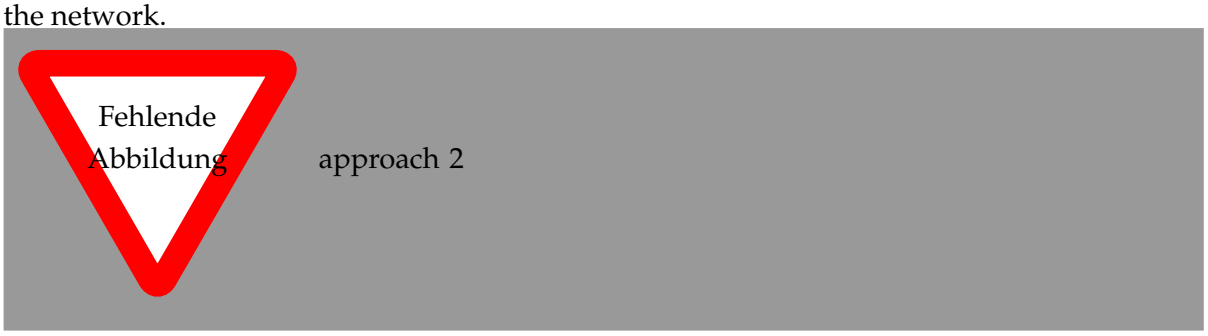


Inhalt... Inhalt...

7.2 Approach II: hidden layer input

The amount of information given to the adversary is the output of one single node. This does not justify the usage of complex architecture and it allows now insight in the deeper dependencies of the classification model. Alternatively one can use the last hidden layer of the classifier which can be described as the final model the sigmoid-decision is based on. This is not fully correct as all dependencies of the model created in the last step are excluded this way. Nevertheless, it is a viable approximation worth testing.

This approach simply inputs the last hidden layer into the adversary without changing any other hyper-parameters. A hyper-parameter scan is then performed to teest the behaviour of the network.

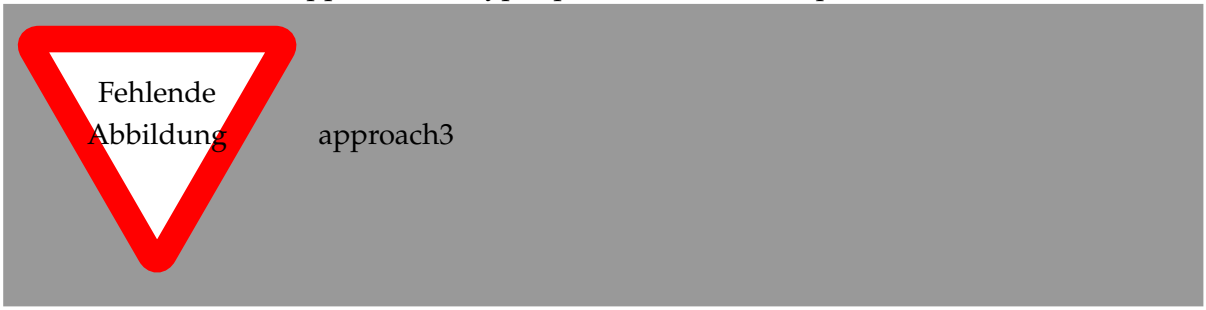


Using this setup the losses behave as desired. At one point the adversary fails to gain any information while the classifier keeps on learning. The combined loss decreases. On the hindsight the model is very unstable and for many hyper-parameter setups it just stops working properly. Furthermore, the improvement on the sensitivity does not visibly improve. It might be possible to further research whether small achievements were made but now only the behaviour and not the effect of an adversarial network was created.

7.3 Approach III: compressed hidden layer input

After using only a single input node in the classical approach and 128 nodes in the second approach, an intermediate number of inputs is tested. For approach III a second to last layer with only 16 nodes is added to the classifier and then fed into the adversary. This way the input dimension is closer to the dimension of the classifier. In addition to that it promises to control the instability of the second approach.

Parallel to the other approaches a hyper-parameter scan was performed.



7.4 Summary

In conclusion the approach for an adversarial neural network as suggested in the paper "Learning to pivot with Adversarial Networks" [32] has not been able to achieve the promised results for an tW-$t\bar{t}$-separation. It was however possible to show the desired behaviour of the two adversarial networks. It has been shown that the input to the adversary is essential and a part of the analysis that would deserve more research. Furthermore, the adversarial neural network has proven to be a structure of its own rather than just two arbitrary classifiers combined. They have to be built around an unhurried optimisation process allowing both networks to slowly learn from different areas of the system's topology.

CHAPTER 8

Conclusions

The work that has been presented in this thesis is split into two. First of all the behaviour of neural networks and adversarial neural networks for classification tasks in particle physics has been investigated. Influence of the hyper-parameters on network performance for different setups has been tested in order to understand how a classifier should be set up and how it can be upgraded to an adversarial neural network.

Secondly the possibilities offered by adversarial neural networks to limit the influence of systematic uncertainties on a classification model has been discussed. In particular, the technique of an adversarial network has been tested on the tW-$t\bar{t}$ separation and the systematic uncertainty arising from the different simulation approaches of DR and DS for the interference term at NLO.

A fundamental knowledge of neural networks and a set of hyper-parameters has been gathered. It has been shown that a good set of parameters is not only dependent on the topology of the problem but also heavily depends on the setup of the network structure. A good classifier does not generally also make a good basis for an adversarial network structure. Instead an adversarial network requires a low learning rate and overall a slow optimisation because more features of the problem need to be taken into account. This is due to the sensitivity to a certain systematic being added to the primary task of classification. The initial optimum that allowed for straightforward and fast training is not the desired model anymore.

Moreover, the adversarial neural network tested in this work is relatively unstable and the outcome changes heavily when hyper-parameters are slightly adjusted. This effect was not observed for the classifying network and resulted in a laborious optimisation process. This made the constraint of computational power even more significant. The most important feature of the adversarial network, researched with the three approaches presented in chapter 7, is the dimension of input it is provided with. There is a strong argument that only one node, as classically suggested, does not justify a complex adversarial network architecture. In addition to that the general information lost when looking only at the final node should not be neglected during the setup. The performances achieved were strongly dependent on the shape of input.

Lastly a significant improvement on the sensitivity of the classifier for the systematic sample could not be achieved. Although the classifier was able to generate a model that rendered its adversary unable to learn no significant differences were visible in the distribution. The

shapes changed but at the stage of this work no sophisticated argument can be made for an improvement. In conclusion, switching from the single-node-input to a more complex input for the adversary seems to be the more promising approach. Combined with a different set of variables and a detailed hyper-parameter scan one might be able to observe a reduction of sensitivity while keeping the quality of the overall classification.

Neural networks in general are a promising technique for researchers in particle physics. The variety of highly different structures and hyper-parameters of the networks makes them a potentially powerful tool for different scopes of duty. An improvement on a sensitivity to systematics has been achieved for different researches using adversarial network approaches. Unfortunately in this work no significant improvements were achieved. Although this might be due to the topology of the problem and the desired model not existing, with a more in-depth optimisation or a generally different setup there may well be improvements not found in this work.

Bibliography

[1] *The Lego Group*, https://www.lego.com/de-de, Accessed: 2019-03-22 (cit. on p. 1).

[2] Sascha Mehlhase, https://build-your-own-particle-detector.org/models/atlas-lego-model-mini/?lang=de, Accesed: 2019-03-23 (cit. on p. 1).

[3] M. Thomson, *Modern Particle Physics*, 6th ed., Cambridge University Press, 2016 (cit. on pp. 1, 8).

[4] Wikipedia contributors, *Elementary particles included in the Standard Model*, [Online; accessed 03-March-2019], 2006, URL: https://commons.wikimedia.org/wiki/File:Standard_Model_of_Elementary_Particles.svg (cit. on p. 6).

[5] M. Tanabashi 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 p. 6).

[6] D. Griffiths, *Introduction to Elementary Particle Physics*, 2nd ed., Wiley-VCH, 2008 (cit. on p. 8).

[7] G. Aad et al., *The ATLAS Experiment at the CERN Large Hadron Collider*, JINST **3** (2008) S08003 (cit. on p. 11).

[8] L. Evans and P. Bryant, *LHC Machine*, JINST **3** (2008) S08001 (cit. on p. 11).

[9] S. Chatrchyan et al., *The CMS Experiment at the CERN LHC*, JINST **3** (2008) S08004 (cit. on p. 11).

[10] A. A. Alves Jr. et al., *The LHCb Detector at the LHC*, JINST **3** (2008) S08005 (cit. on p. 11).

[11] K. Aamodt et al., *The ALICE experiment at the CERN LHC*, JINST **3** (2008) S08002 (cit. on p. 11).

[12] E. Mobs, *The CERN accelerator complex - August 2018. Complexe des accélérateurs du CERN - Août 2018*, (2018), General Photo, URL: http://cds.cern.ch/record/2636343 (cit. on p. 12).

[13] J.-L. Caron, "Overall view of LHC experiments.. Vue d'ensemble des experiences du LHC.", AC Collection. Legacy of AC. Pictures from 1992 to 2002., 1998, URL: https://cds.cern.ch/record/841555 (cit. on p. 13).

[14] N. Wermes and H. Kolanoski, *Teilchendetektoren Grundlagen und Anwendungen*, 1st ed., Springer Verlag, 2016 (cit. on pp. 13, 15).

[15] J. Pequenao, "Computer generated image of the whole ATLAS detector", 2008, URL: http://cds.cern.ch/record/1095924 (cit. on pp. 14, 18).

[16] W. R. Leo, *Teqhniques for Nuclear and Particle Physics Experiments*, 2nd ed., Springer Verlag, 1994 (cit. on p. 13).

[17] Y. Takubo, *The Pixel Detector of the ATLAS experiment for the Run2 at the Large Hadron Collider*, JINST **10** (2015) C02001, arXiv: 1411.5338 [physics.ins-det] (cit. on p. 13).

[18] *Electron efficiency measurements with the ATLAS detector using the 2015 LHC proton-proton collision data*, tech. rep. ATLAS-CONF-2016-024, CERN, 2016, URL: https://cds.cern.ch/record/2157687 (cit. on p. 17).

[19] G. Aad et al., *Topological cell clustering in the ATLAS calorimeters and its performance in LHC Run 1*, Eur. Phys. J. **C77** (2017) 490, arXiv: 1603.02934 [hep-ex] (cit. on p. 17).

[20] 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. 17).

[21] G. Aad et al., *Muon reconstruction performance of the ATLAS detector in proton–proton collision data at $\sqrt{s}$ =13 TeV*, Eur. Phys. J. **C76** (2016) 292, arXiv: 1603.05598 [hep-ex] (cit. on p. 17).

[22] ATLAS Collaboration, *Performance of b-Jet Identification in the ATLAS Experiment. Performance of b-Jet Identification in the ATLAS Experiment*, JINST **11** (2015) P04008. 127 p, 111 pages plus author list + cover page (128 pages total), 55 figures, 17 tables, submitted to JINST, All figures including auxiliary figures are available at http://atlas.web.cern.ch/Atlas/GROUPS/PHYSICS/PAPERS/PERF-2012-04/, URL: http://cds.cern.ch/record/2110203 (cit. on p. 17).

[23] *Optimisation of the ATLAS b-tagging performance for the 2016 LHC Run*, tech. rep. ATL-PHYS-PUB-2016-012, CERN, 2016, URL: https://cds.cern.ch/record/2160731 (cit. on p. 17).

[24] 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. **C78** (2018) 903, arXiv: 1802.08168 [hep-ex] (cit. on p. 17).

[25] G. Aad et al., *The ATLAS Simulation Infrastructure*, Eur. Phys. J. **C70** (2010) 823, arXiv: 1005.4568 [physics.ins-det] (cit. on p. 20).

[26] S. Agostinelli et al., *GEANT4: A Simulation toolkit*, Nucl. Instrum. Meth. **A506** (2003) 250 (cit. on p. 21).

[27] T. Y. and, *The ATLAS calorimeter simulation FastCaloSim*, Journal of Physics: Conference Series **331** (2011) 032053, URL: https://doi.org/10.1088%2F1742-6596%2F331%2F3%2F032053 (cit. on p. 21).

[28] *Summary of ATLAS Pythia 8 tunes*, tech. rep. ATL-PHYS-PUB-2012-003, CERN, 2012, URL: https://cds.cern.ch/record/1474107 (cit. on p. 21).

[29] F. Chollet et al., *Keras*, https://keras.io, 2015 (cit. on pp. 29, 30, 32, 33, 40).

[30] D. P. Kingma and J. Ba, *Adam: A Method for Stochastic Optimization*, arXiv e-prints, arXiv:1412.6980 (2014) arXiv:1412.6980, arXiv: 1412.6980 [cs.LG] (cit. on p. 33).

[31] I. J. Goodfellow et al., *Generative Adversarial Networks*, arXiv e-prints, arXiv:1406.2661 (2014) arXiv:1406.2661, arXiv: 1406.2661 [stat.ML] (cit. on p. 35).

[32] G. Louppe, M. Kagan and K. Cranmer, *Learning to Pivot with Adversarial Networks*, (2016), arXiv: 1611.01046 [stat.ME] (cit. on pp. 1, 35, 38, 49, 52).

[33] G. B. Team, *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*, Software available from tensorflow.org, 2015, URL: http://tensorflow.org/ (cit. on p. 40).

APPENDIX A

Useful information

In the appendix you usually include extra information that should be documented in your thesis, but not interrupt the flow.

List of Figures

List of Tables