Extension of machine learning methods for the observation of longitudinal vector boson scattering with the ATLAS detector

Bachelor-Arbeit
zur Erlangung des Hochschulgrades
Bachelor of Science
im Bachelor-Studiengang Physik

vorgelegt von

José Antolín Neumann
geboren am 01.12.2001 in Jena

Institut für Kern- und Teilchenphysik
Fakultät Physik
Bereich Mathematik und Naturwissenschaften
Technische Universität Dresden
2023

Eingereicht am 11. Dezember 2023

1. Gutachter: Dr. Frank Siegert
2. Gutachter: Prof. Dr. Arno Straessner

Abstract

English

In the standard model (SM) the Higgs mechanism gives vector bosons (VB) their mass via the electroweak symmetry breaking (EWSB). Massive VB have the possibility of being longitudinally polarized. Because of that, the existence of massive longitudinal polarized VB is closely linked to the Higgs mechanism. In order to study the longitudinal polarization, the scattering of two massive gauge bosons is looked at. Since the longitudinal polarization component is rare compared to its background, an important part of the analysis is the identification of this polarization component. For this task Neural Networks (NN) are used. In order to improve the quality of the used NNs, a framework called OPTIMA is to perform a hyperparameter optimization.
In this thesis the successful implementation of Lightning in OPTIMA and the exploration of different possibilities to use multiclass classifier and regression models to find the longitudinal polarization component in the scattering of two same-charged $W^{\pm}$ bosons is described. The tested possibilities show that for the discrimination of the longitudinal polarization component, multiclass classifier outperform regression models. However, from the observed behaviour it seems promising to combine multiclass classifier and regression model in order to achieve better performances. Furthermore, it is shown that performing the SR event selection after training the NN instead of before, results in a significantly better performance on the SR. Another result of this thesis is that using the same events multiple times but with different target labels also improves the performance of the NN significantly. However, most of these effects are dependent on the size of the available dataset and therefore need further testing with differently sized datasets.

Deutsch

Im Standardmodell (SM) verleiht der Higgs-Mechanismus den Vektorbosonen (VB) ihre Masse über die elektroschwache Symmetriebrechung (EWSB). Massive VB haben die Möglichkeit, longitudinal polarisiert zu sein. Aus diesem Grund ist die Existenz massiver longitudinal polarisierter VB eng mit dem Higgs-Mechanismus verknüpft. Um die longitudinale Polarisation zu untersuchen, wird die Streuung von zwei massiven Eichbosonen betrachtet. Da die longitudinale Polarisationskomponente im Vergleich zu ihrem Hintergrund selten ist, ist ein wichtiger Teil der Analyse die Identifizierung dieser Polarisationskomponente. Für diese Aufgabe werden Neuronale Netze (NN) verwendet. Um die Qualität der verwendeten NNs zu verbessern, wird mit einem Framework namens OPTIMA eine Hyperparameter-Optimierung durchgeführt.
In dieser Arbeit wird die erfolgreiche Implementierung von Lightning in OPTIMA und die Erforschung verschiedener Möglichkeiten zur Verwendung von Multiklassen-Klassifikatoren und Regressionsmodellen zur Bestimmung der longitudinalen Polarisationskomponente in der Streuung zweier gleichgeladener $W^{\pm}$-Bosonen beschrieben. Die getesteten Möglichkeiten zeigen, dass für die Unterscheidung der longitudinalen Polarisationskomponente Multiklassen-Klassifikatoren den Regressionsmodellen überlegen sind. Das beobachtete Verhalten lässt es jedoch vielversprechend erscheinen, Multiklassenklassifizierer und Regressionsmodelle zu kombinieren, um bessere Leistungen zu erzielen. Darüber hinaus wird gezeigt, dass die Durchführung der SR-Ereignisauswahl nach dem Training des NN statt vor dem Training zu einer deutlich besseren Leistung auf der SR führt. Ein weiteres Ergebnis dieser Arbeit ist, dass die mehrfache Verwendung desselben Ereignisses, jedoch mit unterschiedlichen Zielklassen, die Leistung des NN ebenfalls deutlich verbessert. Diese Effekte sind jedoch meist von der Größe des verfügbaren Datensatzes abhängig und müssen daher mit unterschiedlich großen Datensätzen weiter getestet werden.

Table of Content

1 Introduction

In physics there are four fundamental forces in the universe and the goal is to find a theory that can explain all four. At the moment such a theory has not yet been found, but there is a theory that can explain three out of the four fundamental forces with only the gravitational interaction missing, called the Standard Model (SM). The SM is one of the most successful theories, since its predicted cross sections have been observed experimentally with very high precision. However, there are many things the SM can not explain, for example gravity (as described by general relativity) or the neutrino masses.
To test the SM and alternative theories, the Large Hadron Collider (LHC) was build at CERN. The LHC is currently being used mainly by the four big experiments ATLAS, CMS, ALICE and LHCb. After the experimental discovery [1] of the last missing particle predicted by the SM, the Higgs boson, efforts shifted to make more precise measurements with higher center of mass energies. The hope is to detect new or very rare processes in order to find clues to new physics.
The ATLAS experiment, among other things, studies the scattering of longitudinally polarized vector bosons. The existence of this process for massive bosons and is, thus, a direct consequence of the electroweak symmetry breaking (EWSB) of the Higgs mechanism. Since the Higgs mechanism plays an important role in developing the SM and possibly finding alternative theories, the polarized vector boson scattering is a very promising process for investigation.
In order to study the scattering of two longitudinally polarized vector bosons, one must first be able to distinguish the longitudinal polarization component of the process from other polarization components that are also measured. The fact that this is a classification task with high dimensional input data motivates the use of machine learning methods. In this work, the scattering of two same-charged, longitudinally polarized $W^{\pm}$ bosons, denoted $W^{\pm}_L$, and their separation from the other polarization components by using artificial Neural Networks (NN) is studied. Since the contribution of the $W^{\pm}_L W^{\pm}_L$ component is much smaller than the contribution of the other polarization components, the accuracy of the discrimination is crucial. Therefore, the used NN is optimized using a hyperparameter optimization. A program that performs a hyperparameter optimization for the NN is OPTIMA (ref.[2]) which so far only supports Keras 2 (ref.[3]) models.
Keras has the advantage that many of the functions used in NN training are already implemented. However, the performance of training (and thus also the hyperparame-

ter optimization) differs between the different models that are used. How the different models perform depends, for example, on the NN size or the hardware which is being used. To improve the versatility of the OPTIMA framework, the compatibility with other models than Keras can be implemented. That is why in this thesis the implementation of Lightning (ref.[4]) into OPTIMA is presented.
Using the new implemented Lightning support, different possibilities to use NNs for the discrimination of the $W_L^{\pm}$ polarization component are explored. So far, for the discrimination of the polarization components of the $W^{\pm}$ bosons multiclass classifiers are used. Multiclass classifiers, as the name suggest, classify between different categories. In this application the categories are the different polarization configurations of the $W^{\pm}$ bosons. Since the upcoming version 3 of the event generator SHERPA (ref.[5]) is able to generate all polarization weights simultaneously for each event (ref.[6]), it is now possible to train the NN as a regression model. Different possibilities of training both types of NNs are tested and compared by using simulated data from SHERPA.
In chapter 2 the theoretical background for this thesis is explained. Chapter 3 presents the experimental methods which are used in this work. The implementation of Lightning into the OPTIMA framework is shown in chapter 4 and the exploration of different NN types to recognize the $W_L^{\pm}$ polarization component is discussed in chapter 5.

2 Theoretical Background

2.1 The Standard Model

The SM is an extensive theory and only some important aspects of it will be introduced in this chapter. A much more detailed and complete introduction can be found in ref.[7]. The SM is the model on which modern particle physics is based. It is able to describe the strong interaction, the weak interaction and the electromagnetic interaction. The last two can be combined to the electroweak (EW) interaction via the Higgs mechanism. The only fundamental interaction that the SM cannot describe is gravity.
In the SM elementary particles are used to describe matter and how interactions take place. All but one of these elementary particles are divided into fermions with spin 1/2 and gauge bosons with spin 1. Gauge bosons are therefore a subgroup of the so-called vector bosons (bosons with spin 1). The gauge bosons are responsible for transmitting the interaction between the fermions which make up the matter.
There are four types of gauge bosons: photons, gluons, $W^{\pm}$-bosons and Z-bosons. Photons are responsible for transmitting the electromagnetic interaction and can only interact with fermions that have an electric charge. The gluons mediate the transmission of the strong interaction and can only interact with particles that have a color charge. $W^{\pm}$- and Z-bosons are responsible for the mediation of the weak interaction and interact with particles having a non-zero weak isospin.
The fermions are divided into quarks and leptons. Quarks have an electric and a color charge and a non-zero weak isospin. Therefore, they can participate in all three fundamental interactions of the SM. Leptons on the other hand have no color charge and therefore are unable to participate in the strong interaction. The neutrinos, which are electrically uncharged leptons, can also not participate in the electromagnetic interaction. Both quarks and leptons have three (known) generations of each particle type. Particles of the same type but from different generations have the same charges and differ only in their mass, which increases with each generation.
The last discovered particle in the SM is the Higgs boson. It has spin 0 and is therefore a scalar boson. It arises from the Higgs mechanism which will be discussed in chapter 2.2. A complete overview of all elementary particles and their charges can be found in table 2.1.1.
The SM is based on symmetries and is therefore usually formulated using gauge theory.

The gauge theory of the SM can be written in a very compressed form as:

$$SU(3)_C \otimes SU(2)_L \otimes U(1)_Y \tag{2.1}$$

Here $SU(3)_C$ is a special unitary group of third degree and describes the strong interaction. The $SU(2)_L$ is also a special unitary group but of second degree and the $U(1)_Y$ is a first degree unitary group. The last two correspond to the EW interaction and the first one to the strong interaction.

Table 2.1.1: Elementary Particles of the SM and their charges and masses. The masses are taken from [8]

Particle	Electric Charge	Weak Isospin	Color Charge	Mass	Gen	Type
Up Quark (u)	$+\frac{2}{3}e$	$+\frac{1}{2}$	Yes	2.2 MeV	1st	Quark
Down Quark (d)	$-\frac{1}{3}e$	$-\frac{1}{2}$	Yes	4.7 MeV	1st	Quark
Charm Quark (c)	$+\frac{2}{3}e$	$+\frac{1}{2}$	Yes	1.28 GeV	2nd	Quark
Strange Quark (s)	$-\frac{1}{3}e$	$-\frac{1}{2}$	Yes	96 MeV	2nd	Quark
Top Quark (t)	$+\frac{2}{3}e$	$+\frac{1}{2}$	Yes	173.1 GeV	3rd	Quark
Bottom Quark (b)	$-\frac{1}{3}e$	$-\frac{1}{2}$	Yes	4.18 GeV	3rd	Quark
Electron (e)	$-e$	$-\frac{1}{2}$	No	0.511 MeV	1st	Lepton
Electron Neutrino (ν_e)	0	$+\frac{1}{2}$	No	< 1.1 eV	1st	Lepton
Muon (μ)	$-e$	$-\frac{1}{2}$	No	105.66 MeV	2nd	Lepton
Muon Neutrino (ν_μ)	0	$+\frac{1}{2}$	No	< 0.19 MeV	2nd	Lepton
Tau (τ)	$-e$	$-\frac{1}{2}$	No	1.7768 GeV	3rd	Lepton
Tau Neutrino (ν_τ)	0	$+\frac{1}{2}$	No	< 18.2 MeV	3rd	Lepton
Photon (γ)	0	0	No	0	-	Gauge Boson
Gluon (g)	0	0	Yes	0	-	Gauge Boson
Z Boson (Z)	0	0	No	91.2 GeV	-	Gauge Boson
W Boson ($W^\pm$)	$\pm e$	± 1	No	80.377 GeV	-	Gauge Boson
Higgs Boson (H)	0	$-\frac{1}{2}$	No	125.25 GeV	-	Scalar Boson

2.2 The Higgs Mechanism

Before the Higgs boson was introduced, the SM contained only the fermions and the gauge bosons. This form of the SM was able to satisfy the local gauge symmetries in eq. 2.1, but it would have required all particles to be massless. The problem was that the fermions as well as the $W^\pm$- and the Z-bosons were observed to be massive. This motivated the introduction of the Higgs mechanism.
For the Higgs mechanism, two complex fields forming a doublet are added to the SM field content. They form a doublet

$$\Phi(x) = \begin{pmatrix} \phi^+ \\ \phi^0 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} \phi_1 + i\phi_2 \\ \phi_3 + i\phi_4 \end{pmatrix} \tag{2.2}$$

with ϕ_1, ϕ_2 being real charged and ϕ_3, ϕ_4 real neutral scalar fields. Its Lagrangian is:

$$\mathcal{L}_{\oplus} = (D^{\mu}\Phi)^{\dagger}(D_{\mu}\Phi) - V(\Phi) \tag{2.3}$$

with the potential

$$V(\Phi) = \mu^2(\Phi^{\dagger}\Phi) + \lambda(\Phi^{\dagger}\Phi)^2 \tag{2.4}$$

where $\lambda > 0$ and $\mu^2 < 0$. Therefore, instead of having one minimum at the center, the potential now has an infinite number of minima outside the centre. The ground state (which is located inside a minimum) can be chosen to be:

$$\Phi_0 = \frac{1}{\sqrt{2}}\begin{pmatrix} 0 \\ v \end{pmatrix} \tag{2.5}$$

where $v = \sqrt{-\frac{\mu^2}{\lambda}}$ is the vacuum expectation value. Going into one of the minima, the ground state spontaneously breaks the symmetry of the Lagrangian. Choosing the specific minimum in equation (2.5) ensures that the $U(1)$ symmetry of quantum electrodynamics (QED) is not broken. Otherwise the photon would also obtain a mass, which would contradict experimental observations.
By expanding Φ around Φ_0 one gets:

$$\Phi(x) \approx \begin{pmatrix} \tilde{\phi_1}(x) + i\tilde{\phi_2}(x) \\ \frac{v+H(x)}{\sqrt{2}} + i\tilde{\phi_4}(x) \end{pmatrix} \tag{2.6}$$

where $H(x)$ is the Higgs field. Using a gauge transformation, eq. (2.6) can be simplified to:

$$\Phi(x) \approx \frac{1}{\sqrt{2}}\begin{pmatrix} 0 \\ v + H(x) \end{pmatrix} \tag{2.7}$$

By plugging eq. (2.7) into the kinetic part of eq. (2.3) one obtains:

$$(D^{\mu}\Phi)^{\dagger}(D_{\mu}\Phi) = \frac{1}{2}\partial_{\mu}H\partial^{\mu}H + \frac{1}{8}(v+H)^2(|g_W W_{\mu}^3 - g_Y B_{\mu}|^2 + g_W^2|W_{\mu}^1 + iW_{\mu}^2|^2) \tag{2.8}$$

Since the gauge bosons are physical particles, the mass matrix has to be diagonal. By diagonalizing the mass matrix, one obtains the following eigenstates representing the

physical particles:

$$W_\mu^\pm = \frac{1}{\sqrt{2}}(W_\mu^1 \mp W_\mu^2), \quad Z_\mu = \frac{g_W W_\mu^3 - g_Y B_\mu}{\sqrt{g_W^2 + g_Y^2}}, \quad A_\mu = \frac{g_W W_\mu^3 + g_Y B_\mu}{\sqrt{g_W^2 + g_Y^2}} \tag{2.9}$$

With that, the Lagrangian from eq. (2.3) can be written as:

$$\begin{aligned}
\mathcal{L}_\oplus = {} & m_W^2 W_\mu^+ W^{-\mu} + \frac{1}{2} m_Z^2 Z_\mu Z^\mu && \text{(W, Z mass terms)} \\
& + (vH + \frac{H^2}{2})(2\frac{m_w^2}{v^2} W_\mu^+ W^{-\mu} + \frac{m_Z^2}{v^2} Z_\mu Z^\mu) && \text{(W-, Z-Higgs interactions)} \\
& + \frac{1}{2}\partial_\mu H \partial^\mu H - \lambda v^2 H^2 && \text{(Higgs-Lagrangian)} \\
& - \lambda v H^3 - \frac{\lambda}{4} H^4 && \text{(Higgs-self-interactions)}
\end{aligned}$$

with the masses:

$$m_Z = \frac{1}{2} v \sqrt{g_W^2 + g_Y^2}, \quad m_W = \frac{g_W v}{2} \quad \text{and} \quad m_A = 0\,. \tag{2.10}$$

In conclusion, the ground state, breaks the electroweak $SU(2)_L \otimes U(1)_Y$ symmetry which leads to mass terms for the $W^\pm$ and Z bosons. This process is therefore called electroweak symmetry breaking (EWSB). At the same time, the EWSB predicts another massive boson. This extra boson is the Higgs boson, which was experimentally found in 2012 [1].

2.3 Longitudinal VBS

Since vector bosons (like the $W^\pm$ and Z bosons) possess a non-zero spin, they can have different polarizations. Polarization is given by the alignment of the particle spin $\vec{S}$ along a given axis. Usually this axis is defined by the momentum $\vec{p}$ of the particle itself. The polarization can be expressed by using the helicity h, which is the projection of the spin onto the momentum axis:

$$h = \vec{S} \cdot \frac{\vec{p}}{|\vec{p}|} \tag{2.11}$$

The helicity is not Lorentz-invariant and therefore depends on the frame of reference. In this work, the reference frame is the centre of mass of the two $W^\pm$ bosons.
For massless spin $S = 1$ particles, the helicity h can have the eigenvalues $h = \pm 1$. I.e. the spin $\vec{S}$ is either parallel ($h = +1$) or anti-parallel ($h = -1$) to the momentum $\vec{p}$. Those two alignments are referred to as transversal polarization. Massive spin $S = 1$ particles, however, can additionally have the helicity eigenvalue $h = 0$. This corresponds to the

case where the spin $\vec{S}$ is orthogonal to the momentum $\vec{p}$ and is referred to as longitudinal polarization. Since through the Higgs mechanism the $W^{\pm}$ and Z bosons become massive, they gain an additional the polarization degree of freedom. This means that the existence of longitudinally polarized $W^{\pm}$ and Z bosons is a direct cause of the Higgs mechanism. By studying them, it is possible to learn more about the exact mechanism of EWSB.
One way to examine the longitudinal polarization of $W^{\pm}$ and Z bosons, is to study vector boson scattering (VBS). It includes triple and quartic self-interactions of, as well as the Higgs exchange between two VB as shown in figure 2.3.1. In this work the interaction between two same-charged $W^{\pm}$ bosons is studied. In order to extract the longitudinal polarization component of the $W^{\pm}$ bosons, one must first isolate the $W^{\pm}W^{\pm}$ events from other events that can also occur during the measurement. The possible interactions between two $W^{\pm}W^{\pm}$ bosons at leading order can be seen in figure 2.3.1.

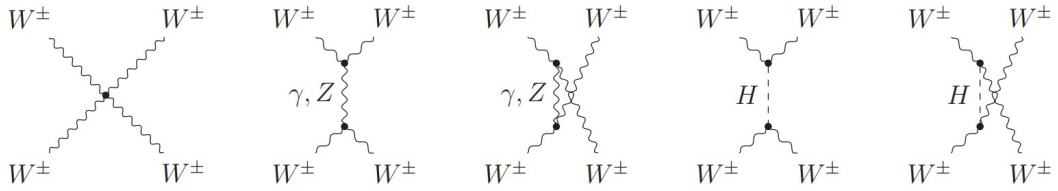


Figure 2.3.1: Feynman diagrams at leading order for possible interactions between two same-charged $W^{\pm}$ bosons. Picture taken from ref.[9].

Since $W^{\pm}$ bosons have a very short lifetime, it is impossible to build a $W^{\pm}W^{\pm}$-collider. For the same reason, it is also impossible to measure $W^{\pm}$ bosons directly in the detector. These $W^{\pm}$ bosons can only be created as intermediate particles and need to be reconstructed from their decay products. In the ATLAS experiment, the production of $W^{\pm}$ bosons is achieved by colliding two protons p. The considered decay process of the $W^{\pm}$ bosons is the leptonic $W^{\pm} \to l^{\pm}\nu$ process, where a $W^{\pm}$ boson decays into one charged lepton $l^{\pm}$ and the corresponding (anti-)neutrino ν[1]. In result, the observable process is not $W^{\pm}W^{\pm} \to W^{\pm}W^{\pm}$, but instead $pp \to l_1^{\pm}\nu_{l_1}l_2^{\pm}\nu_{l_2}jj$. The jets j result from the interacting partons (quarks) of the protons which produce more particles before entering the detector. That is why in the detector instead of single protons a group of particle is detected, which referred to as jet. The corresponding Feynman diagram containing VBS at leading order ($\sim \alpha_{EW}^6$) can be seen in figure 2.3.2, where the blob acts as a placeholder for the interactions in 2.3.1.

[1]The hadronic decay is the dominant decay process, but because hadrons consist of quarks they can not only participate in the weak but also in the strong interaction. This makes it more difficult to reconstruct the $W^{\pm}W^{\pm}$ interaction from the hadronic decay products. That is why for the analysis the leptonic decay is looked at.

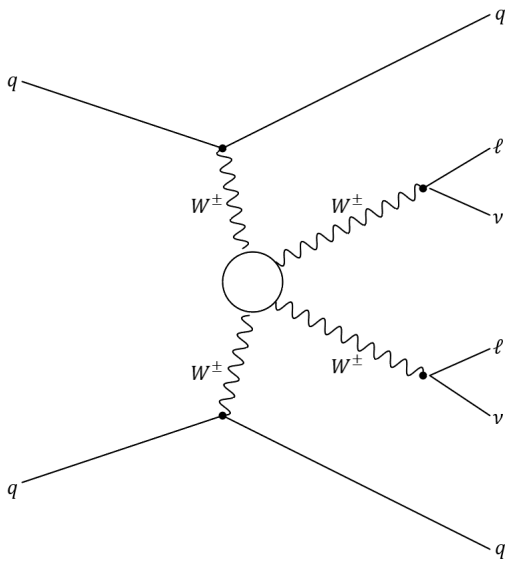


Figure 2.3.2: Leading order Feynman diagram for the scattering of two $W^{\pm}$ bosons (VBS) at the ATLAS experiment, with a placeholder for the interactions from figure 2.3.1.

In reality, the neutrinos are not measured, since they are almost impossible to detect (due to the fact that they can only interact via the weak interaction). This means the reconstruction of the process needs to be done with the information gained from the leptons and jets. This makes it even more difficult to determine whether VBS of two same-charged $W^{\pm}W^{\pm}$ bosons or another process with the same final state, took place. Some of the other processes with the same final state and same leading order ($\sim \alpha_{EW}^6$) as the VBS of two same-charged $W^{\pm}W^{\pm}$ bosons are shown in figure 2.3.3.

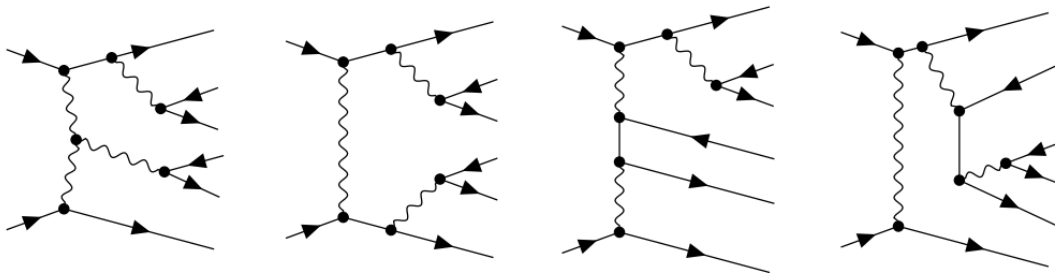


Figure 2.3.3: Examples of leading order Feynman diagrams without VBS topology which also contribute to the $pp \to l_1^{\pm}\nu_{l_1}l_2^{\pm}\nu_{l_2}jj$ process. Picture taken from ref.[10].

3 Experimental Methods

3.1 The Large Hadron Collider

In physics, it is very important to test the theoretical models with experimental data and to see if they fit or if the theory needs to be adapted or changed. The events that need to be observed in particle physics in order to obtain relevant information rarely occur in nature. Therefore, these events have to be artificially created in experiments. One of the biggest and most important experiments in particle physics is the Large Hadron Collider (LHC) at CERN near Geneva. The LHC is a 27 km long, circular, high-energy collider with a center of mass energy (CME) of currently 13.6 TeV and a peak luminosity of $L = 10^{34} \frac{1}{cm^2 s}$ [11] at the CMS and ATLAS detectors. It uses older accelerators as pre-accelerators as shown in figure 3.1.1. The ATLAS detector is shortly presented in chapter 3.1.1 and a more detailed description can be found in [12]. The ATLAS detector and the CMS detector are very similar but their work is independent from each other, so that the results from one experiment can be reproduced and checked by the other one.

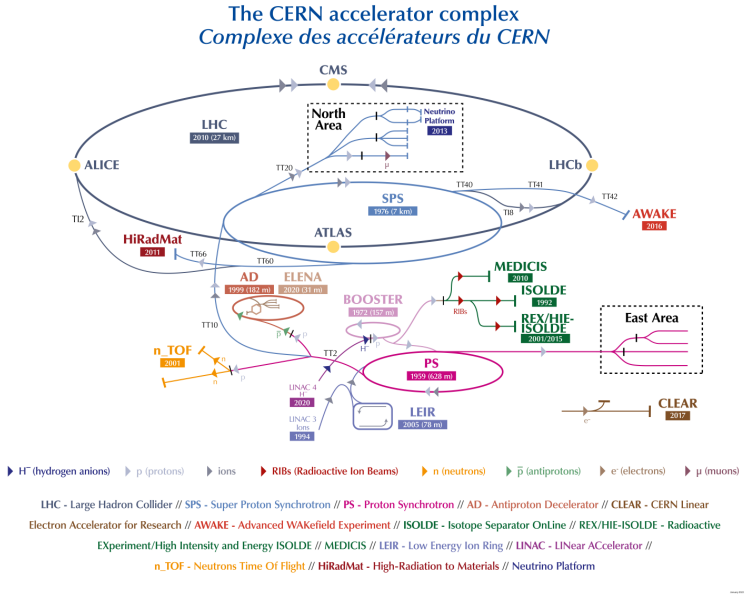


Figure 3.1.1: The CERN accelerator complex. Picture taken from [13].

3.1.1 The ATLAS Detector

ATLAS is the abbreviation for "**A** **T**oroidal **L**HC **A**pparatu**S**". The detector itself is 44 m long and 25 m tall with a mass of about 7 kilotons. It is shaped like a cylinder around the beam axis. The main components are the inner detector, the calorimeter system, and the muon spectrometer. The inner detector detects the trajectories of charged particles inside a magnetic field to determine their electrical charges and momenta. In the calorimeter system the particles are stopped in order to measure precisely their energy via bremsstrahlung (the energy lost inside the inner detector is neglectable). Since myons have a high mass they are not stopped by the calorimeter system. That is why behind the calorimeter system the myon spectrometer measures the momenta of the myons. A schematic representation of the ATLAS detector is shown in figure 3.1.2.

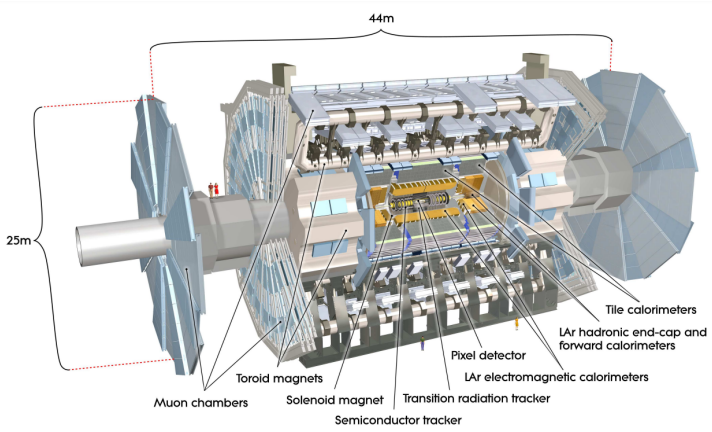


Figure 3.1.2: A schematic view of the ATLAS detector and its insides. Picture taken from ref.[12].

Because of its cylindrical shape the space inside the ATLAS detector usually is described with spherical coordinates. The axis of the beam is the z-axis. Normally for the other coordinates the azimuthal angle ϕ and the polar angle θ would be used. Since the particles move at relativistic speeds, usually the rapidity y is used instead of θ, because it has an additive behaviour regarding Lorentz-boosts along the z-axis. The rapidity y is defined as:

$$y = \frac{1}{2} \cdot \ln\left(\frac{E+p}{E-p}\right) \tag{3.1}$$

For ultra-relativistic particles the rapidity y can be approximated by the pseudo-rapidity η, which defined as:

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

3.2 Computational Foundation

Since this work is based on the framework presented in ref.[9], the approach to the topic is inspired by the more detailed approach in ref.[9].

In order to test the limits of the SM (or alternative theories) often very rare processes like the production of two $W_L^{\pm}$ bosons have to be studied. Since the amount of experimental data is very limited, it is important to reliably recognize those rare events in the available data. This is a task well suited for machine learning methods such as NNs.

3.2.1 Neural Networks

Neural Networks (NN) are a form of machine learning used to approximate functions that are difficult to approximate using classical mathematical methods. Many NNs are inspired by the function of biological brains (hence the name), and are built using small units called artificial neurons. These neurons can have several inputs, but only one output. A neuron takes the inputs and multiplies each input with a corresponding weight. These weighted inputs are summed and a bias term is added. The resulting value is then used as input for the activation function. There are many different functions that can be used as an activation function, the common property being that they are all non-linear. The output of the activation function is used as the output of the entire neuron.
To build a Multi-Layer Perceptron (MLP), which is the type of NNs used in this work, many independent neurons are combined into layers, which are arranged one after another. The outputs of the neurons in one layer serve as inputs for the neurons in the next layer. The first layer, called the input layer, consists of the inputs given to the NN and does therefore not have any neurons. In the last layer, called the output layer, the number of neurons has to be adapted to the task of the NN. In the case of this work, the MLP in chapter 4 for example, is supposed to perform a binary classification. Therefore, the output layer only has one neuron outputting a value between 0 and 1. The layers between the input and output layers are called hidden layers and can vary in number and size.
What happens in each layer of a NN can be elegantly expressed in terms of vectors and matrices:

$$a_i^{(k)}(x^{(k-1)}) = A\left(\sum w_{ij}^{(k)} x_j^{(k-1)} + b_i^{(k)}\right) \tag{3.2}$$

Hereby $a_i^{(k)}(x^{(k)})$ is the output vector of the k^{th} layer, A is the activation function, x is the input vector for that layer, w is the matrix containing the weights for this layer, and b is the bias vector for this layer.
Often, the data used by NNs needs to be normalized in order for the NN to work properly.

Therefore, a normalization layer is added after the input layer. An overview over the structure of a NN is shown in figure 3.2.1.
For a NN to give the correct output it first has to be trained. For supervised training, which is the training method used in this work, labeled data is needed. During supervised training, the data is given to the NN as input and the output of the NN is compared to the expected output and evaluated by a differentiable loss function. The output of the loss function is called loss and it is used as a measure of the NN's performance. Depending on the training objective, different loss functions must be used. For binary classifications, the most commonly used loss function is the binary cross entropy, which is defined as:

$$L(f(x), y) = -[y \cdot \ln(f(x)) + (1 - y) \cdot \ln(1 - f(x))] \tag{3.3}$$

Here, $f(x)$ is the output of the NN and y is the corresponding target label, i.e. the correct class. The assumption is made that y is either 0 or 1.
The goal of the training is to minimize the loss by finding the best parameters for the NN. Since this becomes difficult/impossible to solve analytically, it is solved numerically. To do that, a gradient descent approach is typically used. This approach takes advantage of the fact that each NN defines a differentiable function, allowing to calculate the gradient of the loss with respect to each parameter of the NN. The parameters are then changed in the opposite direction of the gradient according to the following formula:

$$w_{ij,t}^{k} = w_{ij,t-1}^{k} - \alpha \cdot \left[\frac{\partial L(f(x))}{\partial w_{ij}^{(k)}} \right]_{f=f_{t-1}} \tag{3.4}$$

where the parameter α is the learning rate, which regulates the step size. The higher the learning rate, the stronger the parameters are changed. To achieve optimal convergence, the value of the learning rate needs to be chosen carefully. A too high value might result in unstable convergence, unable to reach the minimum due to the large step size, or even diverge in extreme cases. On the other hand a too small learning rate will result in very slow convergence due to the small step size and is prone to get stuck in local, suboptimal minima.
This is done for all the data in the training dataset. If this has been done for the entire training dataset, an epoch has passed. Usually the training of the NN undergoes many epochs, meaning that the training data is used multiple times to adapt the parameters of the NN.
When choosing a sensible value for the learning rate, this algorithm always converges to a local minimum, but is often not close enough to optimal. The reason for that is, that the gradient is updated after every single input. Since the inputs can vary strongly from each other, each gradient can be very different to the previous one. This causes instability, which disturbs the training. To improve this, the training data can be divided into

small pieces, called batches. During training, the change of the parameters is then not calculated and executed for every input individually, but instead calculated as an average over the whole batch and then changed according to the average value. This algorithm is then called mini-batch gradient descent and but is also often referred to as mini-batch SGD or just SGD[1]. Similar to the learning rate, the size of the batches is also important for the performance of the training. If the batch size is too small, the effect of introducing the batches vanishes and the gradient becomes unstable. On the other hand, a certain fluctuation is needed, so that the loss can find its way out of very flat local minima. By setting the batch size too high, these fluctuations are suppressed due to the averaging, which makes the loss escaping bad minima unlikely.

Instead of SGD, in this work an algorithm called Adaptive Moment Estimation (ADAM) is used. ADAM enhances SGD by memorizing the previous gradients so that it jumps over small local minima. This works by introducing a momentum, which can be thought of as the momentum of a ball rolling down a hill. When the ball encounters a small, local minimum, its momentum will cause it to overcome this minimum and keep rolling down the hill to a lower minimum. This helps ADAM to find a better local minimum for the loss. ADAM is also capable of dynamically changing the learning rate. Therefore, a higher learning rate can be chosen at the beginning of training, which increases the speed of learning. When it starts approaching the minimum the learning rate is reduced to ensure the minimum is not missed. A detailed overview of how ADAM works can be found in ref.[14].

Another important part of training NNs is preventing underfitting and overfitting. Underfitting happens when the NN is unable to approximate its target function. It can happen when the NN did not undergo sufficient training epochs or when the structure of the NN is not expressive enough (for example a polynomial function of fifth degree can not be approximated by a NN with only two trainable parameters). Underfitting can be reduced by ensuring that sufficient training epochs are absolved and that the structure of the used NN is adequate to the problem. The second problem is overfitting. It occurs when the NN starts to fit the noise from the training dataset instead of approximating the function represented by the data, which is, after all, the function the NN is supposed to be trained for. If the NN performs significantly better on the training dataset than on an unknown dataset, then this is a strong sign of overfitting. In order to prevent overfitting, instead of using the available data only for training, it is split into a training and a validation dataset. After each epoch of training with the training dataset, the NN is tested with data from the validation dataset. If the calculated loss on the validation dataset keeps decreasing, then the training should continue. If it stops decreasing, the training should be stopped, because otherwise the NN will overfit. This stopping of the

[1]Technically SGD is the algorithm where the changes are updated after every single input. However, both algorithms are often referred to as SGD, since out of context it is clear which one is meant.

training to prevent overfitting is done by a so-called early stopper.

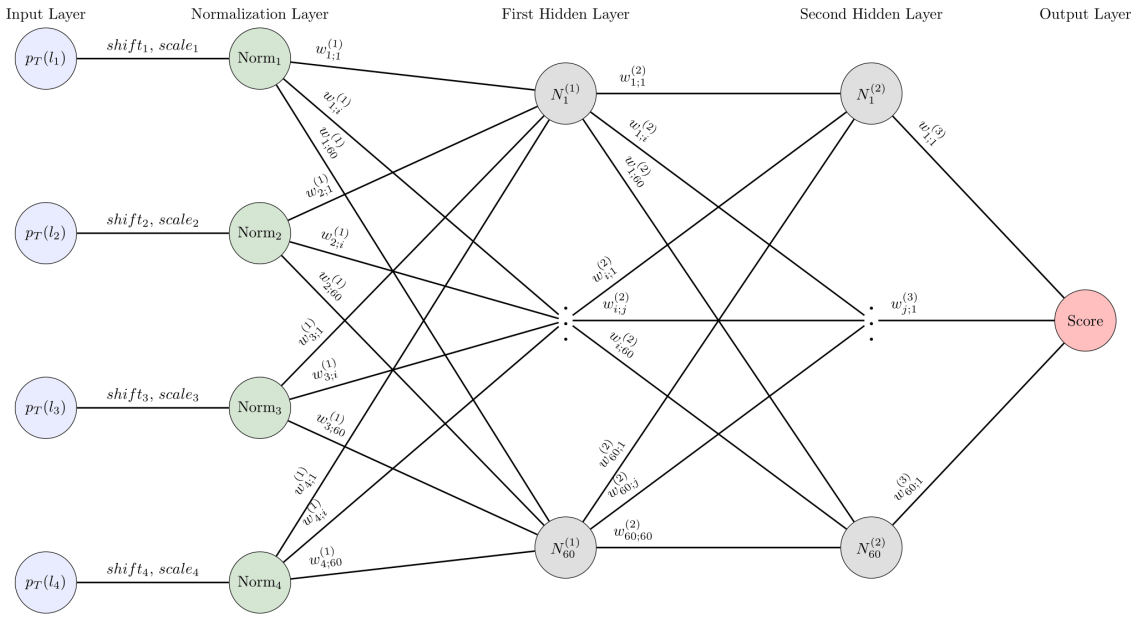


Figure 3.2.1: Structure of a simple MLP. Picture taken from [15].

3.2.2 Hyperparameter Optimization

The performance of NNs depends on their hyperparameters. Hyperparameters are the parameters of a NN which are not automatically optimized. Some examples of MLP hyperparameters are the number of hidden layers, the amount of neurons in each layer, or the learning rate. Typically, the hyperparameters are varied by hand until a set of acceptable hyperparameters is found. Since it is important to maximize the performance of the NN, it is necessary to find the best hyperparameters. To reduce the manual workload, the process of finding the best hyperparameters can be automated using hyperparameter optimization algorithms. Especially when the NN has many hyperparameters automation is very useful.

The most important part of the hyperparameter optimization is the strategy by which the hyperparameters are varied in order to find the best performing NN. The most intuitive approach would be grid search. In grid search each hyperparameter is changed individually in fixed steps, and then the NN is trained for each combination of values for each hyperparameter. This strategy has the advantage that it can be parallelized easily and therefore the computation can be done very efficiently. However, for a large amount of hyperparameters the number of different possible combinations becomes too large, making the computation time too long. There are many strategies that are better than grid search, but every strategy has some weaknesses when faced with certain problems. Therefore, the search algorithm should be chosen according to the problem which needs to be solved. Since OPTIMA is intended to be used in a variety of different situations,

the main search algorithm that was chosen is Bayesian optimization. The advantage of Bayesian optimization is that it has shown to work well with many different kinds of problems.

The idea behind Bayesian optimization is to use a surrogate function that is easy to compute, and use it to approximate the unknown function. In this case, the unknown function is the performance of the NN as a function of its hyperparameters, and the data points used for modeling are the measured performances of the trained NN for a set of hyperparameters. The goal is to find the optimum of this function, since finding the optimum means finding the best set of hyperparameters for the NN. The surrogate function is used to make predictions about promising regions of the search space. The most promising region is then tested next, and the results of this test are used to update the surrogate function. This process is then repeated a fixed amount of times or until a satisfactory result is obtained. In this work, Tree Parzen Estimators (TPEs) are used as surrogate function. A description of how they work can be found in ref.[16].

Another important hyperparameter optimization algorithm is Population Based Training (PBT). The idea behind PBT is to start training different NNs with different hyperparameters and, after a certain amount of training iterations, replacing the worst performing NNs with modified copies of better performing NNs. This process is repeated until all NNs are fully trained. A figure showing the PBT can bee seen in 3.2.2.

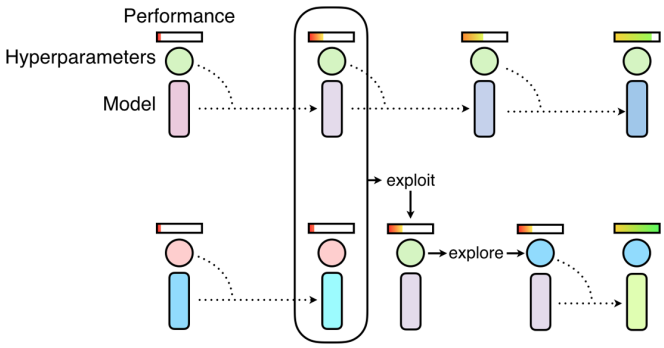


Figure 3.2.2: Population Based Training used for hyperparameter optimization. The figure is taken from ref.[17]. Exploit is the process of replacing bad performing NNs with copies of better performing NNs and exploration is the process of modifying the copies a bit.

In this work the hyperparameter optimization has three main steps. The first step is an input variable selection, where unimportant variables are determined and sorted out before the main optimization. This can be achieved, for example, by randomly shuffling one input variable and observing whether the output of the NN changes or not. If it does not, then the tested input variable is probably not important for the training and can be discarded. Doing this with every input variable sorts out unnecessary input variables.

This can benefit the training, because unnecessary input variables often introduce additional noise, which disturbs the training. Of course this will not work for inputs where the structure of the input data is important (like the pixels in an image). Furthermore, this method does not necessarily take into account correlations between different input variables, so one should be careful with this form of input variable selection. The second main step is the Bayesian optimization, which is used for the main part of the optimization. The third main step is PBT for a fine-tuning of the hyperparameters.
After each main step of the hyperparameter optimization a cross validation is performed. Cross validation takes advantage of the fact that the entire data is divided into different parts for training, validation (and testing). In order to obtain better estimations for the performance of a NN, different parts of the data can be used for training, validation or testing. For example if a dataset is split into five equal parts, then three of these parts can be used for training, one can be used for validation and one can be used for testing. These parts can then be permuted five times, so that each part is used three times for training, once for validation and once for testing. This means that by permuting the split data, it is possible to train the same NN five times with different training data. Therefore, by cross validating, one gets multiple results, which allow statistical estimates for the uncertainties of the NN's performance.

4 Implementation of the Lightning Support into OPTIMA

4.1 Changes to Framework to include Lightning

In this chapter only the changes for the embedding of Lightning to the OPTIMA framework are explained, since only those are part of this work. The full framework can be found under `https://gitlab.cern.ch/atlas-germany-dresden-vbs-group/optima` and is introduced in ref.[9].
Keras and Lightning have many similarities since they both are two different representations of the same mathematical model. But when it comes to implementation they differ in many points. Keras is based on TensorFlow (ref.[18]) and has many already implemented functions which help the user to quickly implement their NN. Lightning on the other hand is based on PyTorch (ref.[19]). Unlike Keras it does not facilitate the underlying library by creating a variety of already implemented functions. Instead it uses the fact that code in PyTorch is often repeated and uses classes to simplify the implementation. In the following the main classes which are changed/added for the implementation of the Lightning support are explained. Those classes do not only need to be defined, but also applied correctly in the trainable function. The trainable function is the heart of the OPTIMA framework and is described in ref.[9].

4.1.1 Trainer

One of the most important differences from Lightning compared to Keras is the use of the trainer class. The trainer is used in Lightning to perform all the important operations with the model (e.g. training the model, evaluating the model, etc.). How these operations are carried out is defined in the corresponding classes, like the model class or the early stopper class (which are presented in the following).

4.1.2 The Lightning Model Class

The heart of Lightning is the model class itself. There the architecture and most of the other hyperparameters of the model are defined. Since the model has to be built many

times with different hyperparameters, the Lightning model class creates the Lightning model based on the hyperparameters which are given in the model configuration. The functions which have to be defined in the Lightning model class are the following:

- **forward:** The forward function defines how the input vector passes through the MLP. This depends on the structure of the MLP.
- **configure_optimizers:** The used optimizer needs to be defined here. In case of this work the used optimizer is ADAM.
- **training_step / validation_step / test_step:** Here it is necessary to define what happens after every training/validation/testing step. For this framework, the training/validation/test loss is calculated here. If the data is weighted, then the weight is included into the loss calculation.
- **on_train_epoch_end / on_validation_epoch_end:** These functions define what happens after each training/validation epoch. Here, the metrics are calculated, logged and then reset.

4.1.3 Abstract Model Class

Integrating Lightning into the OPTIMA framework would be possible by just adding decisions which model is being used at every important point of the framework and then using the corresponding code. However, this method is very inefficient and susceptible to errors, since there are a lot of points where such a differentiation would be necessary and they can easily be overseen. To reduce this issue, the hyperparameter optimization now uses an abstract model with abstract functions for the MLP. These abstract functions have to be able to work with Keras models and with Lightning models. Some of the abstract functions are not dependent on the currently used model and therefore need no change. For the model-specific abstract functions only the Lightning compatibility needs to be implemented, since the Keras compatibility is already existing. In the following the abstract functions which need a Lightning compatibility are explained.

- **___init___:** The initializer is the constructor of the abstract model class. It gets the run_config file and the specific model which is used (i.e. a Keras or Lightning model) as argument and initializes them them as attributes. For Lightning the abstract model also receives the trainer as argument which then is also initialized as additional attribute of the abstract class.
- **predict:** The predict function gets as argument the inputs for the abstract model and is supposed to return the output of the abstract model. In the case of Lightning the inputs are given in form of a dataloader (i.e. an enumerated dataset). This

dataloader is given to the current Lightning model and returns the outputs of the MLP with the help of the trainer. These outputs are then concatenated and returned by the predict function as a numpy array.

- **loss:** This function returns the loss of the MLP for a given input. This task can easily be executed by the trainer, since the Lightning model class already has defined how the loss is calculated.

- **configure_environment:** This function sets the configurations for the trainer, which includes information about the hardware that the trainer can use, e.g. the amount of CPUs. After setting the configurations, the function returns the trainer.

4.1.4 Dataloader & Batching

Lightning operates with Dataloaders. Pre-processing those dataloaders improves the handling of the data. That is why a Dataloader class is typically implemented for Lightning. This class has the following functions:

- **___init___:** In the initialization the inputs, targets and weights are initialized for the train-, validation- and test data as attributes of the class[1].

- **setup:** Depending whether the current stage is ”fit” (training) or ”test”, this function calls the get_dataset function for either the training and validation data or for the test data. The get_dataset converts the given data to a dataset and returns it.

- **train_dataloader / val_dataloader / test_dataloader:** These functions return the train/validation/test dataloader using the get_dataloader function, which converts a dataset to a dataloader.

Batching the data is done for datasets. That is why this is performed in an additional class. The main functions of this class are:

- **___init___:** The constructor of this class gets inputs, targets, weights and optionally the batch size as inputs and initializes them as attributes of the class[2]. From the amount of inputs and the batch size the amount of batches is calculated and also initialized as class attribute.

- **___getitem___:** This function returns the batched inputs, targets and weights.

[1]The inputs, targets and weights for the test data are optional, since not in every hyperparameter optimization testing is performed.
[2]If no batch size is given, the default value is 1.

4.1.5 Early Stopper

The early stopping in Lightning is performed by an early stopper class. Lightning normally has a default early stopper class (similar to the model class) called Callbacks. However, for the hyperparameter optimization some functions have to work different and some additional functions need to be implemented. That is why, an abstract early stopper class is implemented. This class contains all the functions which are independent from the used model. For the Lightning application a third early stopper class is implemented, which inherits from the abstract early stopper class and the Callbacks class. In this class only the Lightning specific functions which need to be changed/added compared to Callbacks class are implemented. Those functions are:

- **___init___:** This function constructs the new early stopper class. It initializes the Callback class, the abstract early stopper class, the training and validation dataloaders, and additional given arguments as attributes.
- **on_train_end :** This function calls the finalize function after the training has been stopped. The finalize function prints the final status message, which includes e.g. the amount of training epochs the MLP did undergo.
- **on_validation_end:** This function is called at the end of every validation. Inside this function the at_epoch_end function is called. This function is defined inside the abstract early stopper class and performs the early stopping.
- **get_weights / set_weights:** These functions either return/set the weights of/for a given model. The weights are usually given in form of a numpy array.
- **save_model:** In this function, the current model together with its trainer are saved as a checkpoint.
- **stop_training:** If called, this function tells the trainer to stop the training due to early stopping.

4.2 The Hyperparameter Optimization for Keras and Lightning

In order to test if the implementation of Lightning into the OPTIMA framework was successful, a hyperparameter optimization for MLPs using Keras and Lightning is performed. For training, a MadGraph sample with 747 518 events is used. The MadGraph event generator is described in ref.[20]. The data is split up to 448 396 events for training, 149 535 for validation and 149 587 for testing. Since this is a large amount of data, the

computation was done on Barnard, which is one of the CPU clusters from the TU Dresden. For the optimization, 100 CPU cores from Barnard are used. Because OPTIMA still has some problems with the PBT when being used on CPU clusters (it runs very slow), this part of the optimization is deactivated. Since in this work the performances of the MLPs are only compared to each other, leaving out PBT does not affect the results. To ensure that the hyperparameter optimization works when PBT is activated, it first was successful tested locally using a very small dataset.

Collisions inside a detector cause particles to be emitted into the detector. Not all of these are detectable. The most common detectable scattering products that are of interest for the polarisation of the $W^{\pm}$ bosons are jets and leptons. Therefore, the properties of the detected leptons and jets are used as input variables for the MLP. The used variables are:

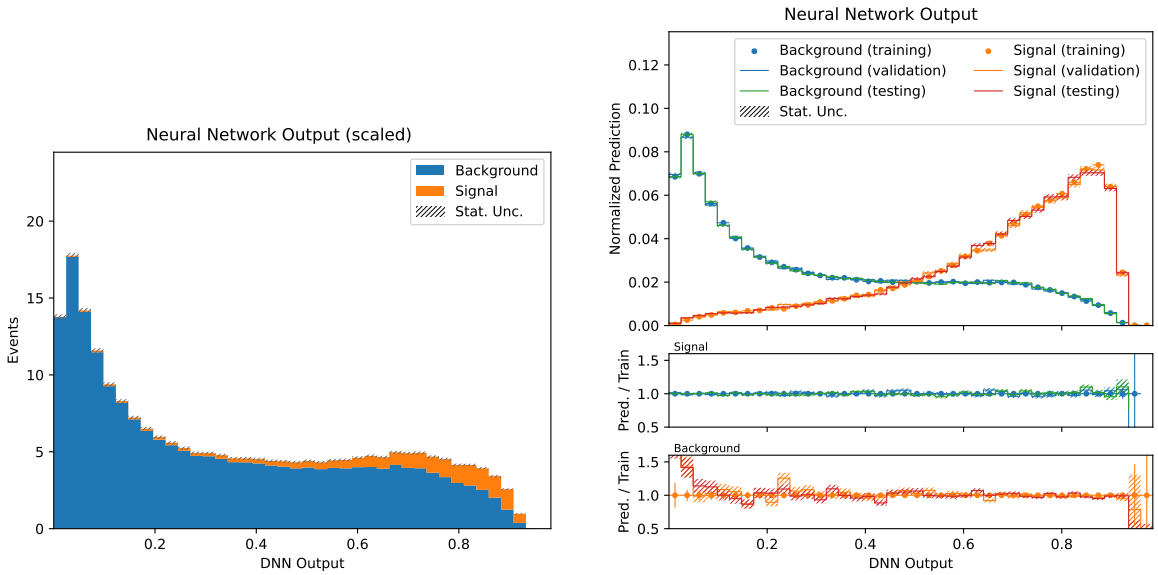
- p_T: The transversal momentum is the component of the momentum perpendicular to the z-axis.
- η: The pseudo-rapidity.
- y: The rapidity.
- Δy_{jj}: The rapidity difference between the two leading jets. Leading means, that they have the highest transversal momentum.
- ϕ: The azimuthal angle.
- $\Delta\phi_{ll}$: The difference in the azimuthal angle ϕ between the two leading leptons.
- E_T^{miss}: The missing transversal energy. It is not measured directly, but is obtained from the sum of the transversal momenta of all detected particles/jets. It arises because before the collision the net transverse momentum is zero, and due to the conservation of energy and momentum, it must also be zero after the collision. Since the total transverse momentum of all detected particles generally is not zero there is a missing transversal energy E_T^{miss}. This is caused, for example, by emitted neutrinos which carry a momentum but are not detected by the detector.
- m_{jj} : The invariant mass of the two leading jets.
- m_{ll} : The invariant mass of the two leading leptons.
- $P_{T_{ll}}$: The total transversal momentum of the two leading leptons.

The variables p_T, η, y, and ϕ are used from the two leading leptons and the two leading jets.

The optimizable hyperparameters of the MLPs in this work are: number of layers, neurons per hidden layer, activation function, batch size, parameters of ADAM optimizer (Adam_β1, Adam_β2, and Adam_ϵ), learning rate and dropout rate. The dropout rate is the hyperparameter of the dropout layer. The dropout layer is inserted after each hidden layer and randomly sets the outputs of neurons in the previous layer to 0. The percentage of outputs that are randomly set to 0 is determined by the dropout rate.

4.3 Results of Hyperparameter Optimization

One of the motivations to implement the Lightning support was, that different models compute faster/slower when faced with a certain type of problems. The question is whether for the problem of finding the LL polarization component in $W^{\pm}W^{\pm}$ boson scattering, Lightning is faster, slower, or equal compared to Keras. When testing with OPTIMA, the computation time of both Keras and Lightning do not differ significantly. This means that for this specific task, regarding the computation time, it makes no difference if Keras or Lightning is used.
The output of the MLP, found by the Lightning hyperparameter optimization, can be seen in figure 4.3.1.



(a) unnormalized (b) normalized

Figure 4.3.1: The output distribution of the resulting NN in cross validation 4.

In figure 4.3.1b it is visible that the MLP is sorting background events closer to the output 0 and signal events (so the LL events which are wanted) closer to an output of 1. It can also be seen, that both distributions have an overlap. This means that, depending on where the cut between background and signal is made, either more signal events are accepted by the cost of also falsely accepting more background events or more background

events are rejected by loosing more signal events at the same time. This behaviour can be represented by using receiver operating characteristic curves (ROC curves). A ROC curve shows the ratio of correctly classified signal events over the ratio of background events which falsely are classified as signal events. For a perfect classifier the curve would be rectangular with an area under the curve (AUC) of 1. A random classifier (the output of the NN is a random number between 0 and 1) would have a diagonal ROC curve with an AUC of 0.5. This means the better the classifier is, the closer the AUC value gets to 1. Therefore, the quality of a classifier can be evaluated by using the AUC value for the ROC curve. The ROC curve and the corresponding AUC value for the final NN can be seen in figure 4.3.2.

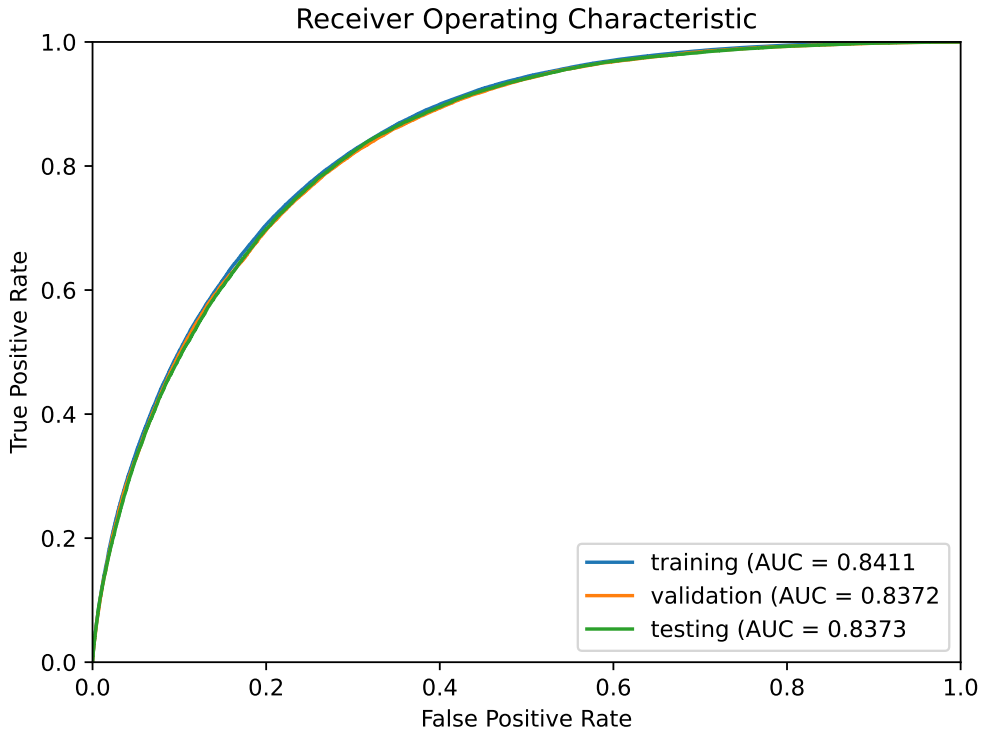


Figure 4.3.2: ROC curve with AUC for the NN in cross validation 4.

It is also visible that there is a difference in the performing between data used for the actual training and data used for validating/testing. This shows the different behavior of the NNs, when facing known and when facing unknown data, called the generalization gap. The AUC values, obtained on the testing data, for the five cross validations using Keras and Lightning can be seen in table 4.3.1.

Table 4.3.1: Testing AUC values for ROC curves for every cross validation using Keras and Lightning

Crossval	**Keras AUC**	**Lightning AUC**
0	0.8368	0.8377
1	0.8342	0.8345
2	0.8346	0.8350
3	0.8372	0.8376
4	0.8364	0.8373
Total	0.8358±0.0012	0.8364±0.0014

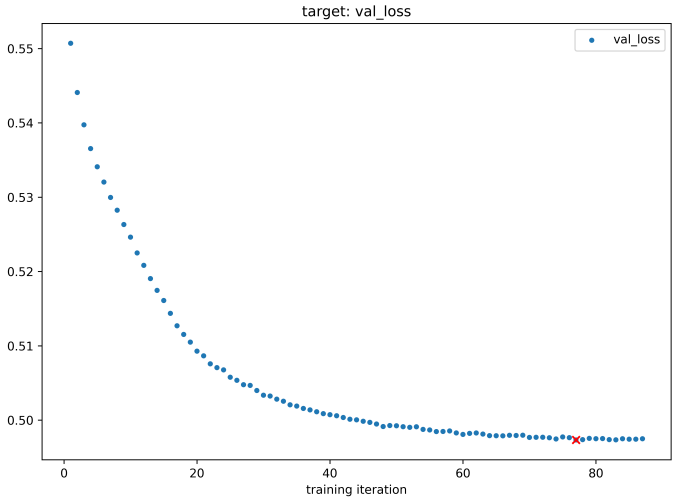
It is clearly visible, that the AUC performances of the MLPs picked using Keras and Lightning are very similar, especially when regarding the uncertainties. The same can be observed when looking at the minimal validation loss which is 0.4962 ± 0.0020 for Keras and 0.4954 ± 0.0021 for Lightning.
The question now arises whether Keras and Lightning found similar hyperparameters for the MLPs and therefore performed similarly or if they found very different MLPs which just have similar performances. This can be answered by looking at the found hyperparameters which are shown in table 4.3.2.

Table 4.3.2: The found hyperparameters of the best performing NNs when using Keras and when using Lightning.

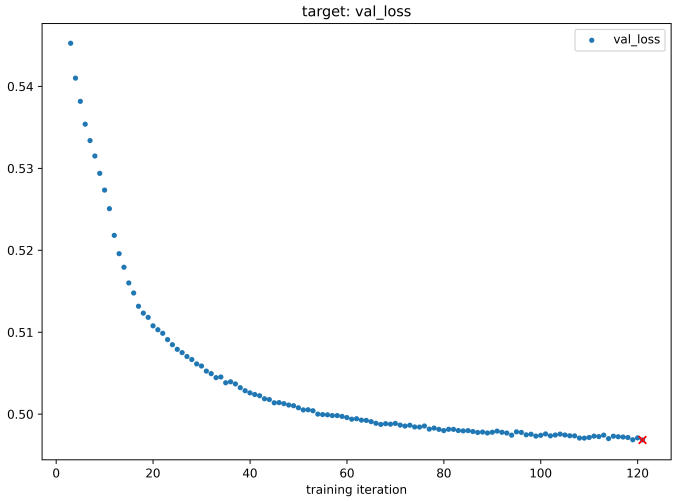
Hyperparameter	**Keras**	**Lightning**
number of hidden layers	3	6
neurons per hidden layer	168	135
batch size	188	198
learning rate	0.0002666842137584751	0.00015277265566554559
dropout rate	0.23042691547370447	0.2637079070416168
max epochs	120	120
epochs	77	121
seed	836318824	429870144
Adam beta 1	0.9840180944350672	0.4426642527956186
Adam beta 2	0.9970461167877457272	0.99979841380372269722
Adam epsilon	3.5400499810168906e-10	1.6448772625235155e-06

It can be seen, that the found hyperparameters differ a lot from each other, especially the most important ones like number of layers, neurons per layer or batch size. This means that the found hyperparameters for the MLPs are not similar and both found MLPs just happen to have similar performances. What is important to add is that the value for *max epochs* is fixed and therefore technically not part of the hyperparameter

optimization. The reason it is included in table 4.3.2 is that it shows, that the training of the best performing Lightning model did stop because of the max epochs limitation and not because the validation loss stopped decreasing. This leads to two hypothesis: The first one is that if the limit of max epochs would be increased, the Lightning model would enhance its performance. If the increment in performance significant, this would mean, that the hyperparameter optimization should be repeated. The second hypothesis is that the Keras model converges much faster to its final form during training than the Lightning model does. The second hypothesis can be checked by looking at the evolution of the validation loss during training. The evolutions can be seen in figure 4.3.3.



(a) Keras



(b) Lightning

Figure 4.3.3: The evolution of the validation loss during the training of the best performing NNs using Keras and Lightning models. The red crosses mark the best point of the training.

It can now be seen, that in this case the Keras model seems to converge faster. For Keras the validation loss starts at a higher value and reaches the area of its final value after 50-60

training iterations (epochs) while for Lightning the validation loss starts lower and gets to the final area after around 80-90 epochs. This can be explained by the fact that the Keras model has a higher learning rate (almost twice as high) and a lower amount of parameters which need to be trained. It is unknown, whether the hyperparameter optimization with Keras generally tends to find smaller NNs or if this just happens randomly. Therefore, it is impossible to conclude a faster convergence for Keras from just this one example. In order to find that out, one would have to test the hyperparameter optimization for Keras and Lightning multiple times with different data.
In order to test the first hypothesis, the obtained MLP from Lightning is being trained again, but with a max epochs of 300. In figure 4.3.4 the full evolution of the validation loss can be seen.

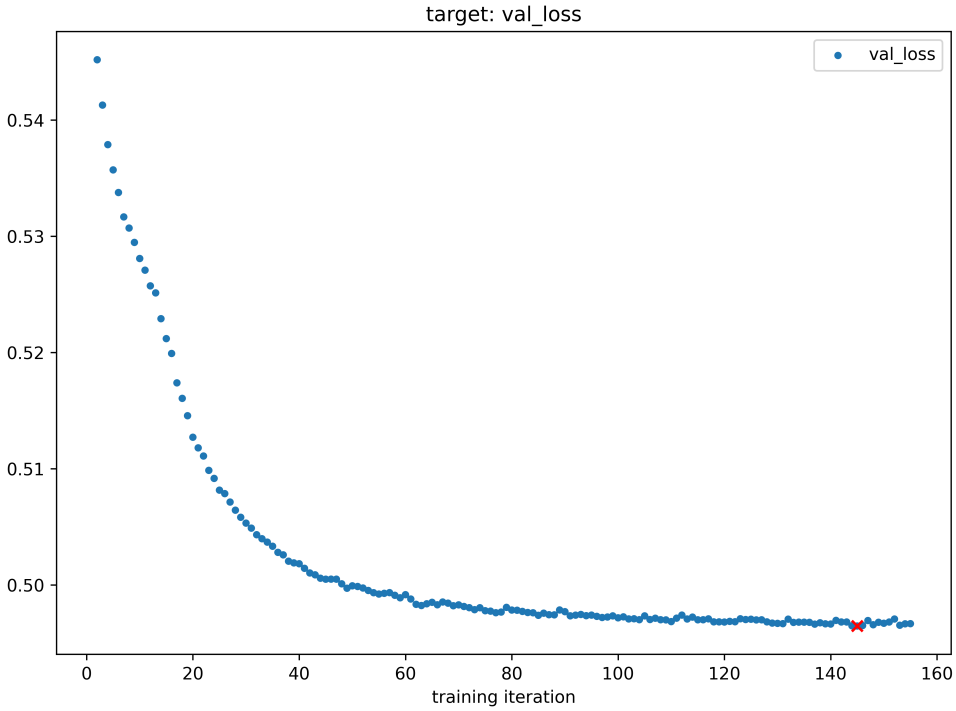


Figure 4.3.4: Evolution of the validation loss for the Lightning model with a max epochs of 300.

The MLP now is finished trained after 145 epochs of training. As seen in figure 4.3.4 and 4.3.3a the training continues afterwards. This is only because the early stopper has a patience and therefore checks whether after some epochs the validation loss starts to sink again or not. Turning to the question whether the longer training had a significant effect, we see that the validation loss for the Lightning model after 145 epochs of training is 0.4949 ± 0.0021 and the AUC value is 0.8368 ± 0.0012. This is an improvement in both validation loss ($-\Delta\, 0.0005$) and AUC value ($+\Delta\, 0.0004$) but neglectable regarding the uncertainties. Therefore, a repetition of the hyperparameter optimization with a higher limit for max epochs is not necessary.

5 Neural Network as Regression Model/Multiclass Classifier

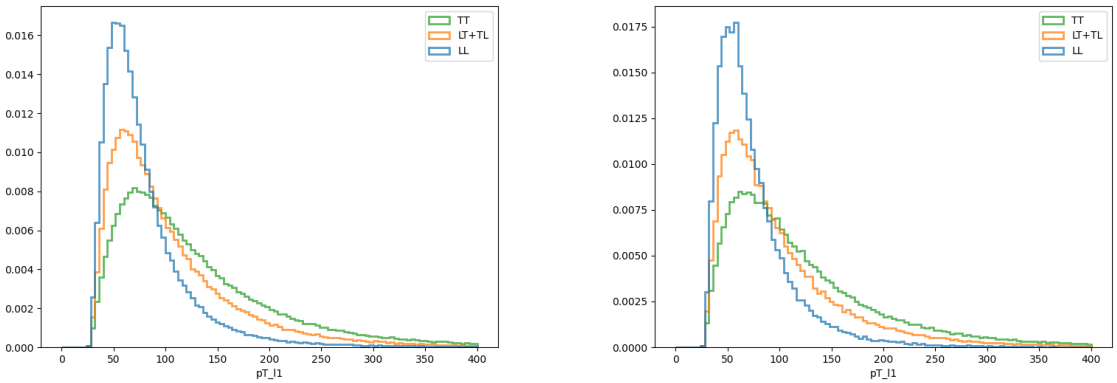
5.1 MadGraph and SHERPA

For the testing of the hyperparameter optimization in chapter 4 a dataset which was simulated with the event generator MadGraph is used. In this chapter, a sample with 1 498 059 events created with the event generator SHERPA is used. A introduction to the SHERPA generator can be found in ref.[6]. For this work only two main differences between both of them are important.
The first difference is that in MadGraph three separate samples are generated for each polarization configuration. They are then combined into one sample containing events for all three polarization configurations (LL, LT+TL and TT)[1]. Hereby the weights for the LL polarization configuration are the event weights of the first partial dataset, the weights for the LT+TL polarization configuration are the event weights of the second partial dataset and the weights for the TT polarization configuration are the event weights of the third partial dataset. This means, that when training a classifier with the combined dataset, the target label is determined by which one of the original three datasets the event is from. In SHERPA on the other hand, each generated event has already weights for LL, LT+TL, TT (and interference[2]), which also represent the ratio between the polarization configuration components. This opens the question, on how to select the target class for the training as classifier. This question is discussed in chapter 5.2.1. Furthermore, since in the SHERPA sample each event has weights for every polarization configuration, it is possible to train the MLP as a regression model. This possibility is discussed in chapter 5.2.2.

The second important difference is that for the used SHERPA sample the input values are taken directly from the event simulation meanwhile for the MadGraph sample from chapter 4 the input values are taken after an additional detector simulation. Since the

[1] In the sample in chapter 4 the polarization configurations TT and LT+TL are merged into the background.
[2] The additional weights for the interference terms are only used in certain cases. When they are used/not used is explained in the corresponding chapters.

detector is not perfect, the data from MadGraph contains detector effects. Therefore, the values in the MadGraph sample are further away from the "real" values, while being closer to the experimental data. This inclusion/exclusion of the detector simulation is not a general property of MadGraph and SHERPA, but a property of the specific datasets, which are used. Both event generators are capable of performing/leaving out the detector simulation.
In figure 5.1.1 the distribution of the transversal momentum of the leading lepton is shown for a MadGraph sample with detector simulation and the SHERPA sample without detector simulation. It can be seen that the distributions are the similar. In the figure a different MadGraph sample, where the LT+TL and TT events are not merged, is used. The purpose of this is to show, that both event generators produce similar distributions.



(a) MadGraph sample (b) SHERPA sample

Figure 5.1.1: The distributions for the transversal momentum of the leading lepton for a MadGraph and a SHERPA sample (SR cuts applied on both).

5.2 Different Possibilities of Training the NN with SHERPA samples

5.2.1 Multiclass Classifier

For the SHERPA sample the target labels are not assigned. However, in order to train a multiclass classifier which can recognize between the LL, LT+TL and TT polarization configuration target labels are needed. The first two options are to either divide the data into three groups and assign a different target label to each group or to take every event three times, but with different target labels assigned. In the first case the total data is significantly less, however, in the second case the MLP is given the same input multiple times, but is expected to give different outputs each time, which could eventually harm the training of the MLP. To see whether allowing the classes to overlap (second case) is

beneficial or not, both variants are tested.
It is also possible do an unweighting using the Monte Carlo (MC) method. Here, the weights for every polarization configuration are normalized using their mean value. The normalized weights are then used as probabilities. If a randomly picked number between 0 and 1 is bigger than the normalized weight, then the event is discarded and if the random number is smaller than the weight, the weight is set to 1 and the event is kept. The events which after the normalization have weights bigger than 1 are called overweight events. Since their weight can not be interpreted as probability they are instead automatically kept and they also keep their normalized weight. For the unweighted version, the target labels are assigned in the same way as for the not unweighted versions before applying the unweighting. Therefore, one gets again two different possibilities, by allowing/preventing overlap events. However, if unweighting is performed there will not be a full overlap, but only a partial overlap between the classes (if overlap is allowed). This is to due the fact, that an event which exists three times but with different target labels, because of the unweighting, will be discarded 0-3 times resulting in a smaller overlap. In the following the not unweighted versions will be referred to as weighted versions.
For all presented types of multiclass classifiers (weighted with full Overlap, weighted without Overlap, unweighted with Overlap and unweighted without Overlap), the interference terms are ignored. The reason for that is that the interference weights are often negative, which would cause problems in the training as a classifier.

5.2.2 Regression Model

A different and perhaps more natural approach to train the MLP with the SHERPA sample is to train it as a regression model. This means that instead of telling the MLP to categorise the events into clear categories using zeros and ones, the MLP is directly tasked to learn the polarization weights. This should represent the real physics better, since in reality the data which is measured for a single event does not come from just one single polarization configuration but is instead a superposition of different polarization configurations. Also for the regression, every event in the sample can be used and there is no need for splitting up the data or accounting for overlap events. In contrast to the classifier, the regression model which is directly trained on the weights, does not have any problems with the negative weights from the interference. Therefore, it can be tested whether including the interference terms improves the training or not.

5.2.3 Signal Region

The data in the SHERPA sample is, unlike the MadGraph sample from chapter 4, not focused solely onto the signal region (SR). The SR is a region in the phase space that has been optimized to provide the highest discovery significance for the $W^{\pm}W^{\pm}$ scattering

compared to all experimental backgrounds (ref.[21]). The data from SHERPA includes not only events from the SR, but also data from outside the SR. The SHERPA sample has only a loose event selection applied ($m_{jj} \geq 100\,\text{GeV}$ and $\Delta y > 2$). One can obtain the SR by additionally performing the event selection shown in table 5.2.1.

Table 5.2.1: SR event selection for the $W^{\pm}W^{\pm}$ scattering.

variable	m_{jj}	number of leptons	m_{ll}	E_T^{miss}	p_T_j1	p_T_j2	p_T_l1	p_T_l2
selection criteria	$\geq$500 GeV	2	$\geq$20 GeV	$\geq$30 GeV	>65 GeV	>35 GeV	>27 GeV	>27 GeV

By applying those event selections to the data before the training, one gets more events which can be measured in the analysis. Since training the MLP on a more general phase space might help the MLP to better understand general relations, it could improve the performance of the MLP inside the SR. To test if this has a significant effect on the performance of the MLP, each version of the multiclass classifier and regression model is trained once with the entire dataset and once with only SR data.

5.3 Comparison

5.3.1 Procedure

To find out which method is the most successful, all of the different possible combinations are tested, with the same sample. To make it comparable, the NNs which are trained on the full data and not only the SR are evaluated only in the SR. Otherwise the NNs trained on the SR would have an advantage, because they are tested in a phase space region where the polarizations are easier to separate.
For this test the OPTIMA framework with Lightning is used. As loss function for the multiclass classifier the Categorical-Cross-Entropy-Loss (CCE) and for the regression model the Mean-Squared-Error (MSE) is used as loss function. They are defined as:

$$L_{CCE} = -\frac{1}{n}\sum_{i=1}^{n} t_i \log(y_i)$$

$$L_{MSE} = \frac{1}{m}\sum_{j=1}^{m}(x_j - \hat{x_j})^2$$

where n is the number of classes, t_i is the true value of the i^{th} category, y_i is the multiclass classifier output for the i^{th} category, m is the number of data points, $\hat{x_j}$ is the correct output value of the regression model for the j^{th} data point and x_j is the output value which the regression model gave for the j^{th} data point. Since different types of NNs need different loss functions the validation loss is not a good figure of merit to compare NNs

of different types.
The ROC curve for the AUC value only tests how well the separation between two distributions is. Since the MLPs now have multiple outputs, the ROC curve is not well defined anymore as it can only evaluate a binary separation. To get around this problem, one ROC curve per output is defined in a one-vs-rest manner. here, the signal category corresponds to the true class for a given output and the background category is defined as the union of the remaining classes. So in total every MLP has three AUC values which can be used for evaluation.

5.3.2 Results

Performance

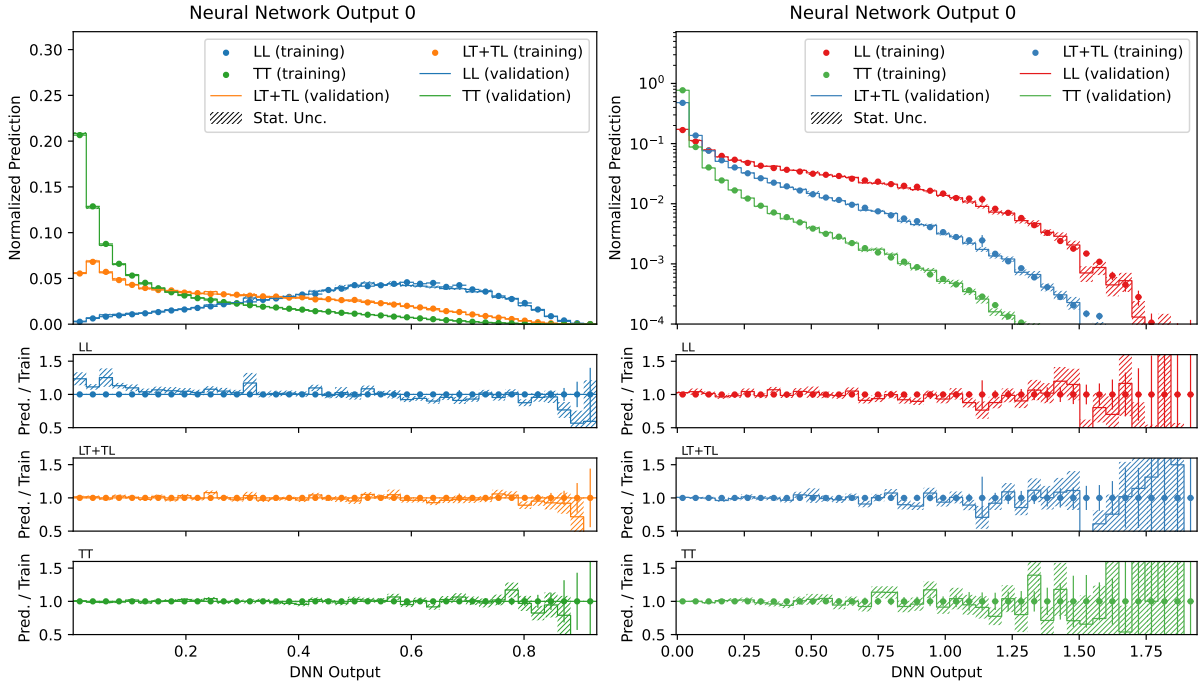
The results for the validation losses and all the AUC values for every different MLP can be seen in table 5.3.1. The data shows, that for all different versions, training the MLP

Table 5.3.1: Performances of the tested NNs.

	Multiclass Classifier							
	weighted				unweighted			
	full overlap		no Overlap		with Overlap		no Overlap	
	all	SR	all	SR	all	SR	all	SR
val. loss	0.8727 ±0.0016	0.8773 ±0.0015	0.890 ±0.006	0.9025 ±0.0032	0.8750 ±0.0019	0.8820 ±0.0029	0.8877 ±0.0013	0.8932 ±0.0014
AUC (LL)	0.8402 ±0.0013	0.8379 ±0.0014	0.8314 ±0.0020	0.8231 ±0.0031	0.8392 ±0.0015	0.8354 ±0.0007	0.8317 ±0.0016	0.8285 ±0.0017
AUC (LT+TL)	0.662 ±0.004	0.6592 ±0.0014	0.6470 ±0.0026	0.640 ±0.005	0.6658 ±0.0034	0.6538 ±0.0022	0.656 ±0.005	0.642 ±0.008
AUC (TT)	0.7771 ±0.0009	0.7748 ±0.0010	0.7672 ±0.0022	0.760 ±0.004	0.7757 ±0.0009	0.7718 ±0.0017	0.7689 ±0.0019	0.7653 ±0.0018
	Regression							
	with Interference				no Interference			
	all		SR		all		SR	
val. loss	0.6594 ±0.0025		0.6623 ±0.0028		0.6105 ±0.0014		0.6148 ±0.0017	
AUC (LL)	0.8288 ±0.0018		0.8258 ±0.0011		0.8268 ±0.0016		0.820 ±0.006	
AUC (LT+TL)	0.6988 ±0.0016		0.6966 ±0.0018		0.6984 ±0.0010		0.6953 ±0.0018	
AUC (TT)	0.7450 ±0.0011		0.7432 ±0.0013		0.7455 ±0.0019		0.7417 ±0.0032	

on the whole dataset and not only on the data from the SR, results in a better performance. This means, that it can in fact be useful to perform the training with a looser event selection than during the application.
That in general all NNs have the worst AUC value for the LT+TL events is not sur-

prising, since it is very similar to both of the other polarizations, meanwhile the other polarizations have only one other similar polarization. This becomes clearer, when looking again at figure 5.1.1, where it is visible, that the LT+TL polarization is in between the other two polarizations. Furthermore, it is visible that, compared to each other, the multiclass classifiers perform better when it comes to the ”pure” polarizations (LL and TT) and the regression models perform significantly better at recognizing the mixed polarization (LT+TL). The normalized outputs (cross validation 0) of the best performing multiclass classifier (weighted, full overlap, inclusive) and the best performing regression model (with interference, inclusive) can be seen in figure 5.3.1. Their ROC curves are shown in figure 5.3.2.

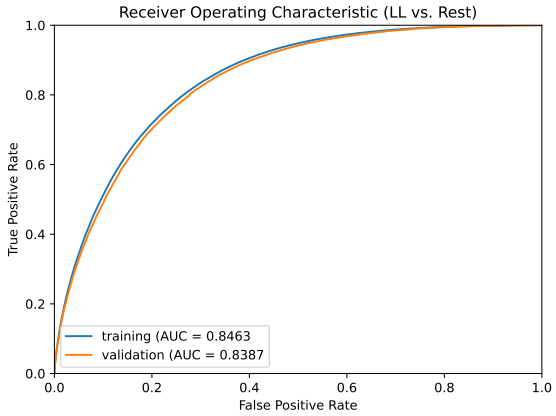


(a) Multiclass classifier (b) Regression model

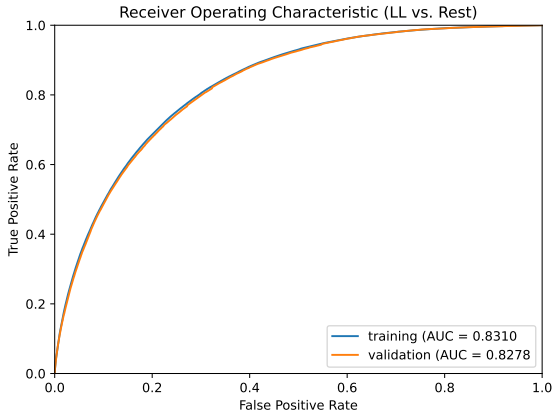
Figure 5.3.1: Normalized outputs in cross validation 0 of the multiclass classifier (weighted, full overlap, inclusive) and of the regression model (with interference, inclusive).

The reason why the regression models have this more averaged performance on the AUC values than the classifiers, might be due to the fact, that the outputs of the classifier are normalized and therefore not as independent from each other as the outputs from the regression model[3].

[3]The normalization is performed, because otherwise the CCE loss function would cause the classifier to give the output vector (1,1,1).

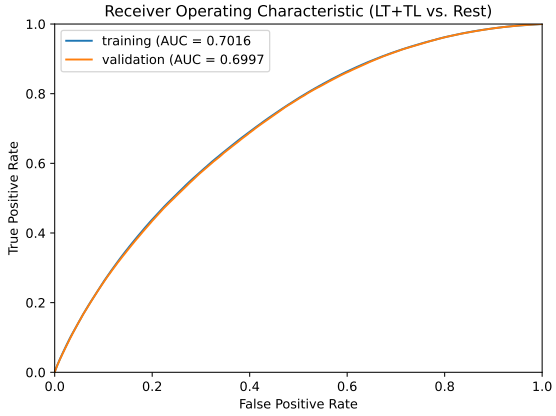


(a) Classifier (LL vs. rest)

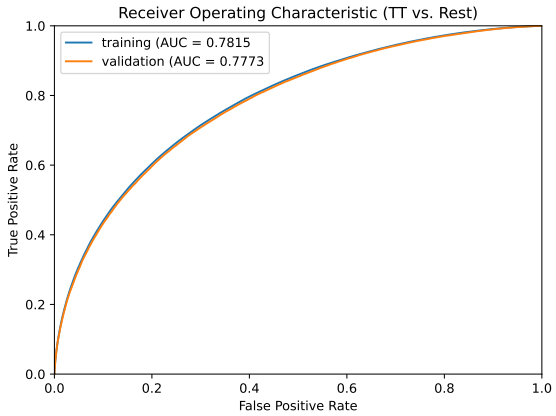


(b) Regression (LL vs. rest)

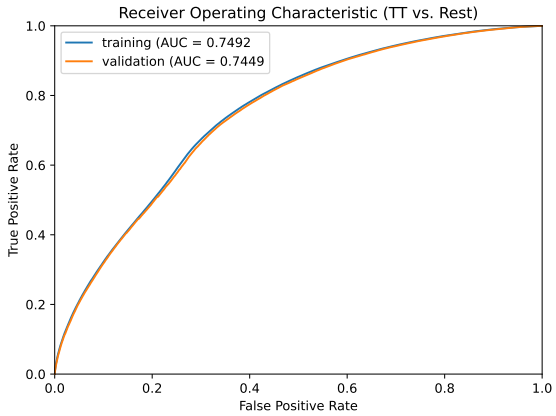
(c) Classifier (LT+TL vs. rest)



(d) Regression (LT+TL vs. rest)



(e) Classifier (TT vs. rest)



(f) Regression (TT vs. rest)

Figure 5.3.2: ROC curves for the multiclass classifier (weighted, full overlap, inclusive) and for the regression model (with interference, inclusive).

To test this theory, it would be necessary to normalize the output vector from the regression model. In order to do so, one would have to solve the problem of the negative interference weights, which make a normalization very difficult. One way would be, to just ignore the interference weights. However, the mean absolute interference weight is around 0.3 and therefore not easily neglectable. For this reason, another way of normalization, which does not destroy the underlying distributions, should be found and

implemented.
Another reason why the regression models are outperformed by the multiclass classifiers, is that the criteria to measure the performance is the AUC value for the ROC curve. Since this value measures separation, the classifier has an advantage, since its fundamental task is to separate, meanwhile the fundamental task of the regression model is to approximate the actual polarization weight. Therefore, the regression model might be better at its task, but is worse when it comes to separating the polarizations. An idea that should be investigated in the future, is combining regression model and classifier, by training a classifier with the outputs of the regression model.

When comparing the multiclass classifiers amongst themselves, it becomes visible, that the most influential factor is overlapping. In both, weighted and unweighted, the performance of the NN increases significantly when overlapping is allowed. This effect is stronger when using weighted data because there the overlap is significantly larger. The enhanced performance of the overlap versions is due to the fact, that the training dataset is significantly larger, when overlap is allowed. Therefore, it is to expect, that this effect becomes neglectable for very large training datasets, because at some point the performance of the NN does not improve when using a larger training dataset. For the same reason, the allowance of overlap events should become more important, for smaller training datasets.
Between the NNs with weighted and with unweighted data, the unweighted version performs slightly better than the weighted version for the case of no overlap. If overlap is allowed, then the weighted version performs slightly better than the unweighted one. This is explained, by the fact, that the overlap is larger in the case of weighted data and therefore this version benefits stronger from the overlap effect. Another effect which explains why the weighted data benefits more from overlapping is caused by the fact that in reality polarizations are not pure but superpositioned. This means that giving an event to the MLP only once, with one of the classes at target labels, does not help the MLP to adapt to the fact, that the polarizations are not pure. However, if the same event is given three times to the MLP with weights for each component of the possible polarization configurations, the MLP can better adapt to the superpositioned nature of the polarizations. This is more difficult if to learn for the MLP if it is presented with the same event multiple times, but without the weights. However, it can be seen that in general the difference between using weighted and unweighted data, seems to not be of mayor importance, since the differences in performance are small compared to the uncertainties.

Consequently when training a NN for the recognition of either LL or TT, a multiclass classifier should be picked. The best one between all the tested multiclass classifiers is the

one trained on the complete, weighted and fully overlapped training data. This variant has the lowest validation loss of all multiclass classifiers and the highest overall AUC values for LL and TT. Since this work is motivated by the search for the LL polarization, this is the NN that should be used. However, the observed effects of overlapping, training the NN on the complete dataset etc. all depend on the size of the training dataset. If training NNs with significantly larger datasets, these dependencies should be reevaluated. If, in some future, one would be interested in the mixed polarizations, then one of the regression models should be used. Between the regression models, the ones where the interference weights are included, seem to have a slightly better performance at AUC values. However, regarding the uncertainties, the improvement is almost neglectable. When looking at the validation losses, it is visible, that the ones without interference have a much lower validation loss. This difference is caused by the interference weights itself. The NN significantly worse at predicting these interference weights and therefore when the MSE loss function calculates the average loss, this average loss gets worse when including the interference. In result, even if the performance on the polarizations stays the same, the validation loss gets worse because of the interference. Therefore, it is impossible to conclude from the obtained data, which of the tested regression models is the best one.

Training epochs

Now the number of training epochs and the corresponding learning rate for each NN are examined. The values for every NN can be seen in table 5.3.2.

Table 5.3.2: Number of training epochs and the learning rates.

	Multiclass Classifier							
	weighted				unweighted			
	full Overlap		no Overlap		with Overlap		no Overlap	
	all	SR	all	SR	all	SR	all	SR
training epochs	112	113	252	86	89	160	45	147
learning rate	4.050e-4	1.859e-4	1.0124e-4	18.288e-4	4.281e-4	7.654e-4	6.863e-4	6.287e-4
	Regression							
	with Interference				no Interference			
	all		SR		all		SR	
training epochs	279		104		78		80	
learning rate	1.0124e-4		4.970e-4		6.049e-4		10.398e-4	

The data shows, that the number of epochs needed for the training, varies strongly between the different types of NN. A pattern is not visible, which suggests that those deviations do not depend on the NN-type. When looking at the number of training epochs and the learning rate, it can be seen, that for lower learning rates (around 1e-4)

the number of epochs is significantly larger. For higher learning rates the number of training epochs is smaller, but is independent on how much higher it is. This behaviour is caused by the ADAM optimizer, because it dynamically adapts the step size during the training. If the learning rate is low, then the convergence will start slowly, which slows down the entire training. A higher learning rate, causes the training to start fast and slow down, when approaching the minimum. However, after a certain learning rate, increasing the learning rate does not accelerate the training anymore, because the ADAM optimizer will just start to decrease the step size faster.

NN size

To see if there is are tendencies towards larger or smaller Nets for any of the NNs, the number of layers and the number of neurons per layer are compared. The corresponding values can be seen in table 5.3.3. Regarding the size of the NNs, no general tendencies can be observed. Furthermore, when comparing table 5.3.2 and 5.3.3, no correlations between the number of training epochs and the NN size can be observed.

Table 5.3.3: Size comparison between the tested NNs.

	Multiclass Classifier							
	weighted				unweighted			
	full overlap		no Overlap		with Overlap		no Overlap	
	all	SR	all	SR	all	SR	all	SR
Number of layers	5	5	5	6	6	6	5	4
Neurons per layer	166	167	178	74	226	58	196	216
	Regression							
	with Interference				no Interference			
	all		SR		all		SR	
Number of layers	5		5		5		4	
Neurons per layer	178		208		178		219	

6 Summary and Outlook

In this thesis the implementation of Lightning in the OPTIMA framework is presented. The implementation of Lightning is tested and compared to the already existing Keras implementation, by performing a binary classification using polarized MadGraph samples of the scattering of two same-charged $W^{\pm}$ bosons. When tasked to discriminate events with doubly-longitudinally polarized $W^{\pm}$ bosons from the longitudinal-transverse and transverse-transverse polarization components, no significant differences in performance are detected between the two implementations.
Using Lightning, the framework is then used to examine different possibilities to train MLPs to perform the same task again, but for every polarization possibility. Therefore, the different fundamental types of NNs, which are tested, are multiclass classifiers and regression models. The classifiers show to have a better performance, when it comes to separate LL and TT events from the rest, meanwhile the regression models separate the LT+TL events better.
For regression models, the possibility of including/excluding the interference terms is tested, however, no significant difference is observed.
For classifiers, the effect of weighting/unweighting the data and of allowing/forbidding the existence of overlapping events, is tested. It can be seen, that there is a significant improvement in performance, if overlapping is allowed. This is due to the fact that through overlapping, the MLP has significantly more data and is also able to better adapt to the superpositioned nature of the polarization components. Meanwhile the effects from using weighted/unweighted data shows to not be so influential.
For both types of NNs and every possibility of using the training data, performing the training on a more general phase space with looser event selection showed improved performance when applied in the SR compared to training only on SR-events.
However, the observed effects on using the training data differently are dependent on the size of the used training dataset. Therefore, when using training datasets of significantly different sizes to what was investigated in this thesis, the performances should be reevaluated.
In general, the analysis done in this thesis could/should be repeated with datasets of different sizes, in order to see how exactly the described effects depend on the size of the dataset.
A possibility which should be investigated in the future, is the combination of a regression

model and a classifier. This can be achieved by, e.g. training a classifier with the outputs of a regression model. By doing so, the regression model might be able to account for the superpositioned nature of the polarizations and the classifier then might be able to better discriminate the different polarization components, resulting in better performances.

References

[1] ATLAS Collaboration. "Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC". In: Physics Letters B 716.1 (Sept. 2012), pp. 1–29. DOI: 10.1016/j.physletb.2012.08.020. URL: https://www.sciencedirect.com/science/article/pii/S037026931200857X?%20via3Dihub.

[2] The OPTIMA framework. https://gitlab.cern.ch/atlas-germany-dresden-vbs-group/optima.

[3] F. Chollet et al. Keras. https://keras.io/.

[4] Lightning. https://lightning.ai/.

[5] The SHERPA event generator. https://sherpa-team.gitlab.io/.

[6] M. Hoppe. "Implementation and phenomenology of polarized cross sections in the simulation of vector boson prouduction with the SHERPA event generator". 2023. URL: https://cds.cern.ch/record/2872971.

[7] M. Thomson. "Modern Particle Physics". Cambridge University Press, 2013. ISBN: 978-1-107-03426-6.

[8] R.L. Workman et al. (Particle Data Group). "Review of Particle Physics". In: Progress of Theoretical and Experimental Physics 2022.8 (Aug. 2022). 083C01. ISSN: 2050-3911. DOI: 10.1093/ptep/ptac097. eprint: https://academic.oup.com/ptep/article-pdf/2022/8/083C01/49175539/ptac097.pdf. URL: https://doi.org/10.1093/ptep/ptac097.

[9] E. Bachmann. "Polarization studies in same-sign W-boson pair production at the LHC with the ATLAS detector". Presented 12 April 2023. 2023. URL: https://cds.cern.ch/record/2872930.

[10] S. Todt. "Theoretical Studies, Observation, Cross-section Measurement, and EFT Reinterpretation at $\sqrt{s} = 13\,\text{TeV}$ with the ATLAS Detector". PhD thesis. 2022.

[11] Lyndon Evans and Philip Bryant. "LHC Machine". In: Journal of Instrumentation 3.08 (Aug. 2008), S08001. DOI: 10.1088/1748-0221/3/08/S08001. URL: https://dx.doi.org/10.1088/1748-0221/3/08/S08001.

[12] ATLAS Collaboration. "The ATLAS Experiment at the CERN Large Hadron Collider". In: Journal of Instrumentation 3.08 (Aug. 2008), S08003. DOI: 10.1088/1748-0221/3/08/S08003. URL: https://dx.doi.org/10.1088/1748-0221/3/08/S08003.

[13] The CERN accelerator complex, layout in 2022. URL: https://cds.cern.ch/record/2800984.

[14] D. P. Kingma and J. Ba. "Adam: A Method for Stochastic Optimization". In: (2014). URL: https://doi.org/10.48550/arXiv.1412.6980.

[15] M. V. Stange. "Teachings-higgs-dnn". (unpublished).

[16] J. Bergstra et al. "Algorithms for Hyper-Parameter Optimization". 2011. URL: https://dl.acm.org/doi/10.5555/2986459.2986743.

[17] M. Jadeberg. Population Based Training of Neural Networks. 2017. URL: https://arxiv.org/abs/1711.09846.

[18] TensorFlow. https://www.tensorflow.org/.

[19] PyTorch. https://pytorch.org/.

[20] J. Alwall et al. "MadGraph 5: Going Beyond". In: Journal of High-Energy Physics, 2011. URL: https://arxiv.org/abs/1106.0522.

[21] ATLAS Collaboration. "Measurement and interpretation of same-sign W boson pair production in association with two jets in pp collisions at $\sqrt{s} = 13$ TeV with the ATLAS detector". In: Journal of High Energy Physics (2023).

Erklärung

Hiermit erkläre ich, dass ich diese Arbeit im Rahmen der Betreuung am Institut für Kern- und Teilchenphysik ohne unzulässige Hilfe Dritter verfasst und alle Quellen als solche gekennzeichnet habe.

José Antolín Neumann
Dresden, Dezember 2023