Neural Network Based Approach for Optimization of Λ Signal/Background Ratio in the CBM Experiment at FAIR

Gianna Zischka

Institute of Computer Science
Johann Wolfgang Goethe-University
Frankfurt am Main, Germany

supervised by
Prof. Dr. Ivan Kisel

September 21, 2023

Bitte dieses Formular zusammen mit der Abschlussarbeit abgeben!

Erklärung zur Abschlussarbeit

gemäß § 25, Abs. 11 der Ordnung für den Bachelorstudiengang Informatik vom 06. Dezember 2010:

Hiermit erkläre ich

_____

(Nachname, Vorname)

Die vorliegende Arbeit habe ich selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel verfasst.

Zudem versichere ich, dass alle eingereichten schriftlichen gebundenen Versionen meiner vorliegenden Bachelorarbeit mit der digital eingereichten elektronischen Version meiner Bachelorarbeit übereinstimmen.

Frankfurt am Main, den

_____

Unterschrift der/des Studierenden

Abstract

In the field of heavy-ion physics, particle accelerators are used to examine different physical models. In collider experiments as well as fixed-target experiments, the colliding ions result in a burst of particles that can be measured through a detector setup. The measurements help physicists to understand the different states of matter.

The future Facility for Antiproton and Ion Research (FAIR) will provide scientists the ability to study the states of nuclear matter under extremely dense conditions. One of the experiments at the facility is the heavy-ion experiment Compressed Baryonic Matter (CBM). Since there is no simple criteria to find events of interest and CBM requires a collision rate of up to 10 MHz, a full event reconstruction is required to be performed online.

The First Level Event Selection package, including the Kalman Filter Particle Finder, is used to reconstruct the events with their particles. Some particles have such a short lifetime, that they decay almost immediately after the collision. The reconstruction of these particles is of high interest for physicists. The KF Particle Finder is used to find and reconstruct these particles.

Decaying particles are referred to as mother particles, whereas their decay products are referred to as daughter particles. One of these decaying particles is Λ, which will be investigated within this work. In the reconstruction process, when a decay is recognized, all possible mother particles are created, such that no signal of particles will be accidentally rejected, if the particle hypothesis is wrong. This approach creates noise, so called background.

In the present thesis, a neural network based approach is studied to improve the signal/background ratio, and thus, to reject the background produced by falsely reconstructed particles, without rejecting too many signal particles.

The presented approach, with a neural network implemented using the ANN4FLES package, has an accuracy of almost 99% on the validation set. This neural network is compared with the default approach of the KF Particle Finder and it is shown that the signal/background ratio was improved significantly by a factor of 10.97.

Zusammenfassung

In dem Forschungsgebiet der Schwerionenphysik werden Teilchenbeschleuniger verwendet, um verschiedene physikalische Modelle zu untersuchen. In Collider- sowie in Fixed-Target-Experimenten führen die kollidierenden Ionen zu einer Explosion von Teilchen, die durch Detektoren gemessen werden können. Die Messungen helfen Physikern, die verschiedenen Zustände der Materie zu untersuchen.

Die zukünftige Einrichtung Facility of Antiproton and Ion Research (FAIR) wird Wissenschaftlern die Möglichkeit bieten, die Zustände von Kernmaterie unter extremen Dichte-Bedingungen zu erforschen. Eines der Experimente der Einrichtung ist das Compressed Baryonic Matter Experiment (CBM). Da es keine einfachen Kriterien zur Identifizierung interessanter Kollisionen in dem CBM Experiment gibt und eine Kollisionsrate von bis zu 10 MHz erfordert wird, muss eine vollständige Rekonstruktion der Kollisionen online durchgeführt werden.

Das First Level Event Selection Paket, einschließlich des Kalman-Filter Particle Finders, wird verwendet, um die Kollisionen mitsamt ihren Teilchen zu rekonstruieren. Einige Teilchen haben eine so kurze Lebensdauer, dass sie fast unmittelbar nach der Kollision zerfallen. Die Rekonstruktion dieser Teilchen ist für Physiker von hohem Interesse. Der KF Particle Finder wird verwendet, um diese Teilchen zu finden und zu rekonstruieren.

Zerfallende Teilchen werden als Mutterteilchen bezeichnet, während ihre Zerfallsprodukte als Tochterteilchen bezeichnet werden. Eines dieser zerfallenden Teilchen ist Λ, das in der vorliegenden Arbeit untersucht wird. Im Rekonstruktionsprozess werden, sobald ein Zerfall erkannt wird, alle möglichen Mutterteilchen erzeugt. Dies ist nötig, damit kein Teilchensignal versehentlich abgelehnt wird, falls die Teilchenhypothese falsch ist. Dadurch erzeugt dieser Ansatz so genannten Background, bei dem es sich um falsch rekonstruierte Teilchen handelt.

In der vorliegenden Arbeit wird ein auf neuronalen Netzen basierender Ansatz untersucht, um das Signal/Background-Verhältnis zu verbessern und somit den durch falsch rekonstruierte Teilchen erzeugten Background zu unterdrücken. Dabei ist zu beachten, nicht zu viele korrekt rekonstruierte Parikel zu verwerfen.

Der vorgestellte Ansatz, der mit dem ANN4FLES-Paket implementiert wurde, hat mit dem Validierungsdatensatz eine Klassifizierungsgenauigkeit von fast 99%. Dieses neuronale Netz wird mit dem Standardansatz des KF Particle Finders verglichen, und es wird gezeigt, dass das Signal/Background-Verhältnis mit einem Faktor von 10,97 deutlich verbessert werden konnte.

Contents

Chapter 1

Introduction

Relativistic heavy-ion experiments represent a large research area in particle physics. This research area concentrates on the effects of collisions of two heavy ions with high temperature and density. This research aims to investigate the basic properties of matter. One of the primary objectives is to gain a deeper comprehension of the Quantum Chromo Dynamics (QCD). Additionally, another focal point is the exploration of the Quark-Gluon Plasma (QGP), which was first discovered at the Relativistic Heavy Ion Collider [1]. QGP is one of the objectives to be explored in the Compressed Baryonic Matter (CBM) experiment at the future Facility for Antiproton and Ion Research (FAIR), build in Darmstadt, Germany.

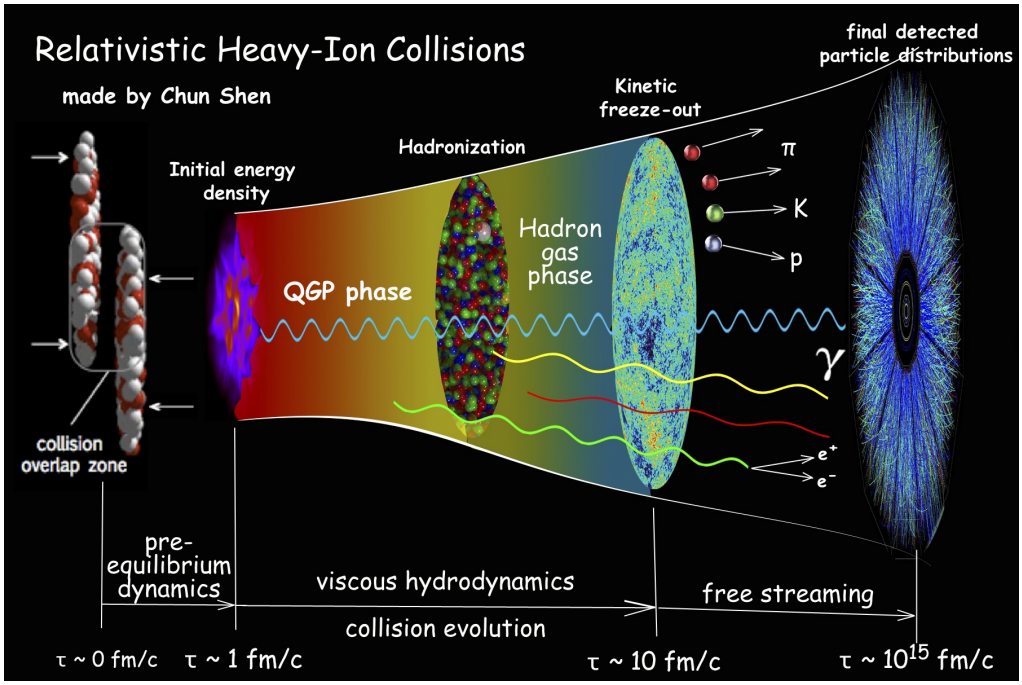


Figure 1.1: Illustration of the different states of a relativistic heavy-ion collision. [2]

In the process of the collision of two heavy ions with high temperature, particles are projected towards one another at relativistic speeds in order to examine their behavior in extreme conditions, such as high temperature or high density. These experiments can be separated in two fields, the *collider experiments* and the *fixed target experiments*. The former are two heavy-ion beams, which collide with each other.

In a fixed target experiment, heavy-ions are accelerated and fired against a fixed target.

When two heavy ions collide, an interaction region is formed in the collision overlap zone. This is visualized in Figure 1.1. In this region, nuclei from each ion interact with each other. If a large portion of the nuclei from both ions interact, it is called a central collision. If the nuclei just graze each other, it is termed a peripheral collision [3]. Nuclei that do not participate in the interaction are called *spectators* [4, p.22].

In the collision overlap zone, primary collision area are formed. This is followed by a pre-equilibrium phase of QGP, which is typically short-lived compared to the overall space-time evolution of the collision (event). After approximately 1 fm/c, if the energy-density is high enough, it is possible that the QGP is created. Within this, the normally bounded quarks and gluons can move freely. After this QGP phase, the energy-density is not high enough anymore and the quarks and gluons combine to form hadrons, which is also shown in Figure 1.1. Within the hadron gas phase, the energy-density continues to decrease and a chemical freeze-out occurs. This is the ending of inelastic collisions, where the particles are colliding and are still chemically changing. At around 10 fm/c the kinetic freeze-out also occurs, then even the elastic collisions end, which means, that there will be no more interactions between particles and the hadrons escape freely. [5] [6, p. 18,19]

For the investigation of QGP, the created particles of the collision have to be studied. Strangeness is expected to be an indicator for deconfined matter, like QGP [7]. The short-lived particle Λ consists of one up-, one down- and one strange-quark and therefore, Λ might be considered as an indicator for deconfined matter. Short-lived particles are particles, which have a short lifetime. Λ, for example, has a mean lifetime of $(2.632 \pm 0.020) \times 10^{-10} s$ [8]. These particles often decay quickly due to their instability. They are part of the most interesting physics, especially including those with a very small production probability [9]. As they can not be detected by the detectors, they have to be reconstructed by their decay products. For this purpose the KF Particle Finder package is used.

In this thesis, the signal/background ratio of Λ has to be optimized for the Compressed Baryonic Matter (CBM) experiment by using a neural network. The signal/background ratio indicates the ratio of correctly reconstructed Λ particles to those which are incorrectly reconstructed or classified. The neural network used in this thesis is a deep multilayer perceptron implemented by using the Artificial Neural Networks for First Level Event Selection (ANN4FLES) package. This network is integrated in the KF Particle Finder package after training using its trained weights.

Chapter 2

Compressed Baryonic Matter Experiment at FAIR

The Compressed Baryonic Matter (CBM) experiment [10] is part of the future Facility for Antiproton and Ion Research (FAIR) [11], build in Darmstadt, Germany and will be one of four major research pillars of FAIR. The other ones are Nuclear Structure, Astrophysics and Reactions (NUSTAR) [12], Antiproton Anihilation at Darmstadt (PANDA) [13] and Atomic, Plasma Physics and Applications (APPA) [14].

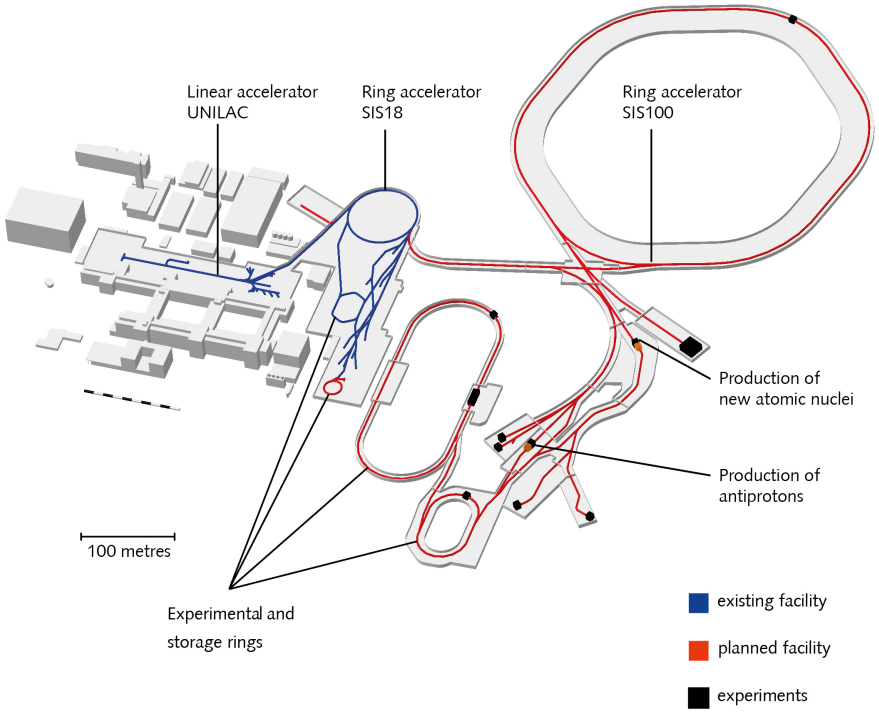


Figure 2.1: Layout of the Facility for Antiproton and Ion Research (FAIR) [15]. The blue marked parts represent existing facilities of the Gesellschaft für Schwerionenforschung (GSI). The red marked parts represent the planned facilities of FAIR.

FAIR (Figure 2.1) will extend the current existing accelerator facility of the GSI Helmholtz Centre for Heavy Ion Research in Darmstadt. The unique feature of FAIR is that it will be able to deliver particle beams of all chemical elements (or ions),

as well as antiprotons [16]. Currently FAIR is still under construction and CBM is expected to be ready for launch in 2028 [16]. FAIR will consist of the synchrotron SIS100, which will have 100 Tm bending power [17], as well as several storage rings and experiment areas. The SIS100 will be able to reach an energy up to 29 GeV [18, p.3]. The already existing particle accelerator SIS18 from GSI will be the injector for SIS100.

In general, researchers at FAIR create conditions in the laboratory that matter is normally exposed to in large planets, stars, and stellar explosions. These conditions include very high temperatures, pressures and densities. To produce these conditions, ions are beamed at small samples of matter using a particle accelerator. For a very short time, cosmic matter is then created at the collision points. [19]

The QCD phase diagram shows how strongly interacting matter behaves by variation of temperature T, baryon-chemical potential μ_B, and isospin-chemical potential μ_I. As it is shown in Figure 2.2, normal nuclei exist only in a small area with low temperature and small μ_B and μ_I. Increasing the temperature T and μ_B results hadronic matter, which consists of nucleons, baryonic resonances and mesons. CBM will concentrate on high baryon density (seen in the diagram with high baryon-chemical potential). This will be done at heavy-ion collisions at moderate temperature and high density. The experiments ALICE (Large Hadron Collider (LHC)) and STAR (Relativistic Heavy Ion Collider (RHIC)) on the other hand can explore the area with low baryon density, but high temperature. [20] [21]

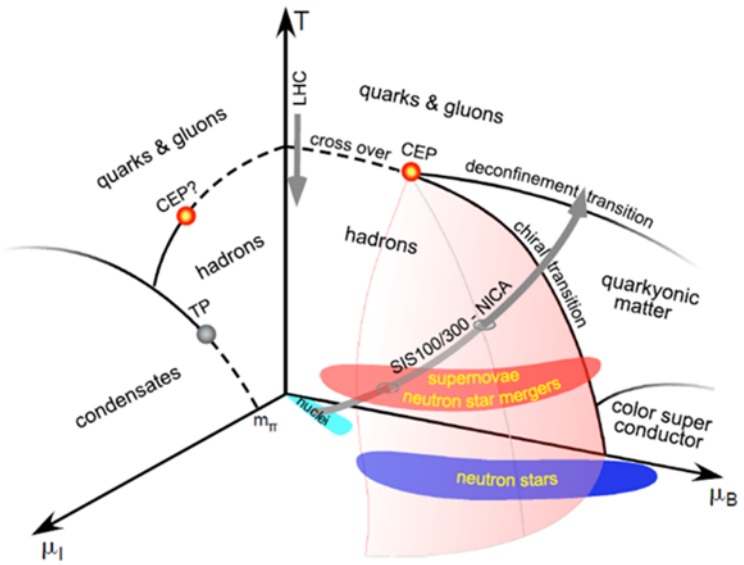


Figure 2.2: 3-dimensional QCD phase diagram. On the x-axis, the isospin-chemical potential μ_I is shown, the y-axis shows the baryon-chemical potential μ_B and the z-axis the temperature T. The grey arrow on the right shows the area of the CBM experiment with the synchrotron SIS100. The grey arrow in the middle indicates the research area of the ALICE experiment at the LHC. [22]

2.1 The CBM Detector Setup

The CBM experiment is a fixed target experiment. The target in CBM is typically 100μm thin [23]. CBM will contain 8 detectors:

- Micro-Vertex Detector (MVD)
- Silicon Tracking System (STS)
- Ring Imaging Cherenkov detector (RICH)
- Muon Chambers (MuCh)
- Transition Radiation Detector (TRD)
- Time Of Flight detector (TOF)
- Electromagnetic CALorimeter (ECAL)
- Projectile Spectator Detector (PSD)

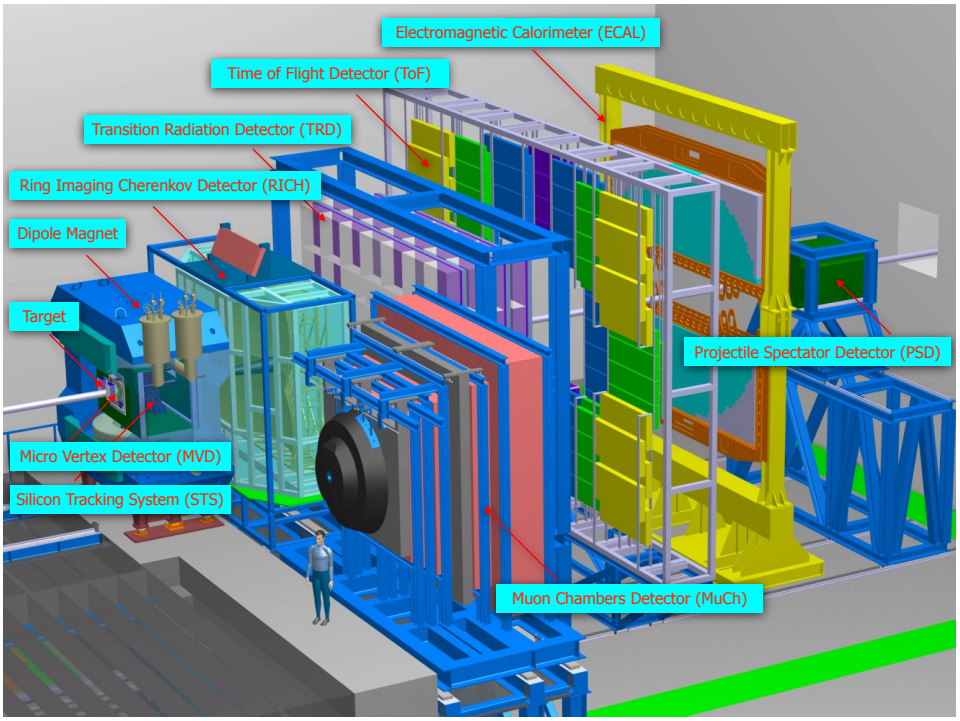


Figure 2.3: Detector setup of the CBM experiment. [24]

The *Silicon Tracking System (STS)* and the *Micro-Vertex Detector (MVD)* are located in the dipole magnet, as visualized in Figure 2.3. They form together a silicon tracking and vertex detection system. The STS consists of low-mass silicon micro-strip detectors and maybe also include one or two hybrid-pixel detector layers, which can provide unambiguous space point measurements[10, p. 877]. STS is used for track reconstruction and momentum determination of charged particles [10, p. 880]. Using STS, a track reconstruction in a wide momentum range from about 100 MeV up to more than 10 GeV is possible with a moment resolution of 1% [10, p.

877]

The *Micro-Vertex Detector (MVD)* will be deployed specifically for precise measurements of open charm and also for determining secondary decay vertices of D mesons with a high precision. Especially the determination of the secondary vertices of D mesons is important to suppress the background of pions and kaons, since D mesons decay quickly in pions and kaons. It will also be used for electron measurements to filter out close pairs, thereby minimizing combinatorial background. To achieve this, the MVD consists of two layers of monolithic active silicon pixel sensors, located at 10 and 20 cm from the target. A possible third layer would be located only 5 cm away from the target. In addition, the MVD detector enhances hyperon identification.[10, p. 877,880, 881, 898]

The *Ring Imaging Cherenkov detector (RICH)* has the capability to measure electrons and suppress pions with momentum less than 8 GeV/c, while the *Transition Radiation Detector TRD* identifies electrons and positrons with momentum above 1.5 GeV/c. Three TRD units, each with 3-4 detector layers, are planned and are anticipated to exceed a pion suppression factor of 100, while maintaining a 90% electron efficiency. [10, p.881 - 882]

The Muon Tracking Chambers (MuCh), along with an active hadron absorber system featuring layers of iron, will be used for measuring muons. For this function, the MuCh will be located where the RICH currently resides [10, p.878] and will replace TRD and ECAL additionally [4, p. 14]. This low-momentum muons have to be identified among high particle densities.

The *Time Of Flight TOF* detector is a 120m^2 wall, made of Multigap Resistive Plate Chambers (MRPC) [10, p. 883] [25]. TOF is designed for identifying charged hadrons up to a momentum of approximately 4 GeV/c. For this, a system time resolution below 80ps and a 95% efficiency is required [25]. Additionally, they must be capable of managing high-density particle streams [25].

The *Electromagnetic Calorimeter ECAL* will be used for the measurement of direct photons and the neutral mesons like π^0, η, which decay into photons. It is a so-called ”shashlik” type calorimeter and will have 140 layers.[10, p. 884]

The Projectile Spectator Detector (PSD) will serve to discern if a collision is either central or peripheral, as well as to establish the orientation of the reaction plane. Additionally, it identifies the non-interacting nucleons in a nucleus-nucleus collision. [10, p. 884]

2.2 Basic Knowledge of Quarks

Each atom comprises an atomic nucleus and orbiting electrons, which belongs to the leptons. The nucleus of an atom is minuscule and comprises the majority of the matter. The hydrogen atom, for instance, has a size of approximately 10^{-10}m, while the nucleus is only 10^{-15}m in size, i.e. around 100,000 times smaller than the entire atom [26, p.148]. The nucleus is composed of nucleons: positively charged protons

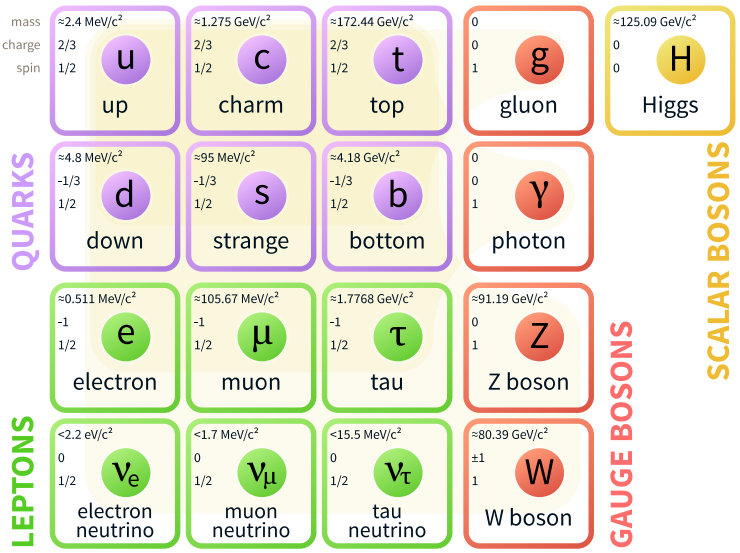


Figure 2.4: Standard model of particle physics [28]. Listed are all of the 6 quarks, the gauge bosons like gluons and the leptons like electron or muon and, additionally, the 2012 first discovered Higgs boson. All particles are listed together with their approximated mass, electric charge measured in e and spin, the intrinsic angular momentum

(p^+) and neutrally charged neutrons (n). Both of them are baryons, belonging to the hadrons. Except for hydrogen, all atomic nuclei contain multiple nucleons in the form of protons and neutrons. The nucleus of a hydrogen atom consists of only one proton. Protons and neutrons, in contrast to leptons, are each made up of three quarks. There are six different types (flavors) of quarks known: up (u), down (d), strange (s), charm (c), bottom (b) and top (t) [27, p. 8]. In Figure 2.4, all quarks, leptons and bosons are listed together with the approximated mass and electric charge. The electric charge is here, as usual, measured in units of the elementary charge (e), which is about $1.602 \cdot 10^{-19}$ coulomb [26, p.86, p.158]. Stable matter comprises solely of up and down quarks, Protons consist of two up quarks and one down quark: $p^+ = uud$, while neutrons comprise one up quark and two down quarks: $n = udd$. The other quark flavors can be found in short-lived hadrons, like the baryon lambda (Λ) which consists of one up quark, one down quark and one strange quark: $\Lambda = uds$.

According to quantum chromodynamics (QCD), a quark possesses not only electric charge but also strong charge, also known as color charge [27, p.8]. There are three types of color charge: red, green and blue [26, p.158]. For each quark is also one antiquark, which has the opposite charges, the complementary colors [29, p.54,55]. Under normal circumstances quarks never move freely, they are always in threes within baryons or twos in mesons [27, p.8]. The summed up color of the quarks in one particle is always white [29, p.55]. The reason for this are the gluons, which belongs to the gauge bosons. There are known exchange bosons for almost all of the four fundamental interactions, except for gravity, where the exchange boson has not yet been found [29, p.52]. Exchange bosons are seen as force carriers. While the photon is exchanged as an exchange boson in order to interact with each other in the case of electrically charged particles, it is the gluon in the case of particles that have a strong interaction and a color charge [27, p.8]. Gluons have eight possible types of color charge and are the reason why quarks are never isolated in normal

circumstances. However, there is a state that does exist in heavy-ion collisions, called quark gluon plasma (QGP) [1]. Within this plasma, quarks and gluons were bound only weakly and can move freely. This plasma is one of the research areas of the future CBM experiment.

2.3 CBM Research Area

The objectives of the CBM experiment is to investigate the quantum chromo dynamics (QCD) phase diagram at the region of high baryon densities [10, p. 37] similar to the inner core of neutron stars. Other goals include the investigation of the equation-of-state of nuclear matter at densities, also similar to the cores of neutron stars and the analysis of phase transitions, chiral symmetry restoration, and exotic forms of QCD matter, including strange QCD matter [30].

With collisions of higher energy and density levels, such as it can be produced at FAIR, it is possible for atomic nuclei to compress during the collision to the point where the nuclei touch and attempt to resist further compression. With increased density of nuclear matter, as it can be achieved in the CBM experiment, the nuclei begin to overlap and dissolve. This results in the formation of a Quark-Gluon Plasma (QGP). Within this plasma, quarks and gluons, which are normally confined within hadrons, are free to move and interact with each other. [31]
This QGP was first discovered at RHIC [1]. However, at lower beam energies of the CBM research program, the creation of QGP is unlikely.

After the QGP phase, the quarks and gluons combine each other again in new color-neutral hadrons. Generally, the produced particles can be categorized into long-lived and short-lived particles. The former are directly detectable by the detectors, including stable particles and those with an extended decay length $c\tau$ [4, p.63]. In contrast, short-lived particles are not detected, since they never reach any tracking system before their decay. These particles must be reconstructed using their decay products, commonly referred to as daughters, with the original particle termed the mother.

In Figure 2.5, an illustration of the decay of the particle $\overline{\Omega}^+$ into $\overline{\Lambda}$ and K^+ is shown, as part of a simulated central Au-Au collision at an energy level of 25 AGeV (GeV per nucleon). Consequently, two secondary vertices are observed: the initial decay vertex of $\overline{\Omega}^+$ and the subsequent decay vertex of $\overline{\Lambda}$, which further decays into π^+ and $\overline{p}$. One important challenge of the CBM experiment is to reconstruct the short-lived particles. Due to their rarity, a significant number of collisions are required to detect them online, including these decay chains. Regarding the CBM experiment, the collision rate will be up to 10 MHz [9], with measured data up to 1 TB/s [32]. This complexity requires a sophisticated reconstruction process that relies on both time and spatial parameters.

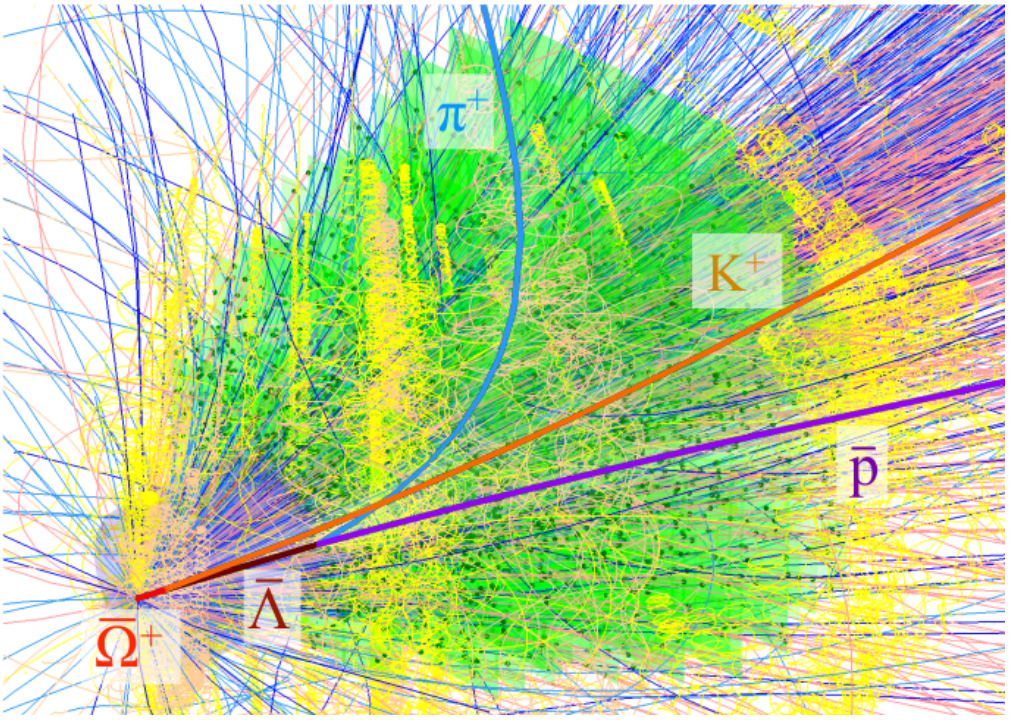


Figure 2.5: The decay of the particle $\overline{\Omega}^+$ in $\overline{\Lambda}$ and K^+ in a simulated central Au-Au collision with an energy of 25 AGeV (GeV per nucleon). Another secondary vertex is seen at $\overline{\Lambda}$ resulting in the daughter particles π^+ and antiproton $\overline{p}$. The simulation was done for the CBM experiment and resulted in about 1000 charged particles. [4, p. 8]

Chapter 3

Kalman Filter Particle Finder Package

The collision rate of the CBM experiment will be up to 10 MHz [9], with created data streams of up to 1 TB/s [32]. Due to this huge data rate, it is impossible to store all collision measurements. On the other hand, CBM requires this high interaction rate for its search for rare particles. Therefore, it is needed to select the most interesting events for the current research online. For this, the First Level Event Selection package (FLES package) [33] is used, which is shown in Figure 3.1. With this package, each event topology can be reconstructed [32] for a physics analysis and, thus, to find events of interest. The FLES package includes several modules and has to be very fast on CPUs, since only them are used for selecting the data [34].

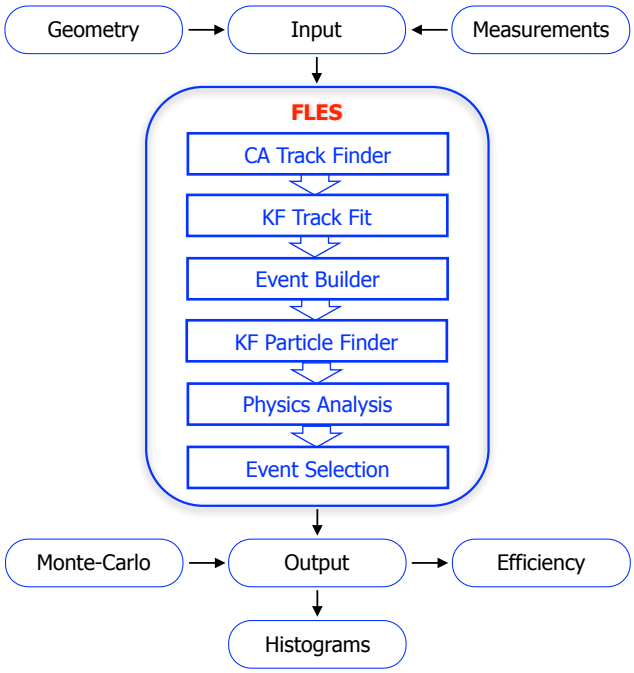


Figure 3.1: Illustration of the architecture of the FLES package [35]

Some of those modules are based on the Kalman filter. The Kalman filter is a method which was first introduced by Rudolf E. Kalman in 1960 [36] and updated 1961 [37]. It was originally intended for discrete-time linear systems and can estimate

the system states and parameters from noisy and partially redundant measurements [38, p. 3]. A major advantage over other stochastic estimation methods is its iterative structure, which makes it particularly suitable for real-time applications [38, p. 3], as it is also required for CBM. Nowadays, there are extended methods of the Kalman filter that also work for continuous and non-linear systems[39].

The first module in the FLES package is the 4-dimensional (space and time) Cellular Automaton Track Finder (CA Track Finder) [40], which is an efficient tracking algorithm based on cellular automaton. Using this, the tracks of the charged particles are reconstructed, since they can be detected by the detectors, due to their interaction with the detector materials. The second module is the Kalman Filter Track Fit (KF Track Fit) library [41]. With this Kalman filter based module, the track parameters and covariance matrices are estimated with high precision. This high precision is necessary for finding rare events of interest. After the KF Track Fit, the tracks that have been fitted are divided into events according to the fitted track time, with an event reconstruction efficiency of 83% [42]. Subsequently, short-lived decayed particles are reconstructed and selected using the KF Particle Finder package [4], with which they are reconstructed from their decay products, the so called daughter particles. After the KF Particle Finder package, an event is fully reconstructed. Based on this, a physics analysis can be applied and the events of interest can be selected afterwards.

The FLES package takes as an input the geometry of the detector, like the distances between detectors or the target, and, the measurements of the detectors. In the case of simulated events, the output of FLES package can be compared with the simulated data to allow a performance analysis of the reconstruction processes. Additionally, histograms are created and the efficiencies calculated.

For the present work, simulations are created using the Monte-Carlo (MC) simulation package Ultra-relativistic Quantum Molecular Dynamics (UrQMD) [43]. The simulated data is employed to enable an evaluation of the proposed approach and a thorough comparison with the default approach of the KF Particle Finder package. Additionally, this enables the application of supervised learning methods, as the true outcomes are known from the simulated data.

The investigated particle Λ consists of an up-, down- and a strange-quark. On the one hand, strange quarks are expected to indicate deconfined matter [7]. On the other hand, Λ is a particle that is created frequently in the energy ranges of the CBM experiment [44]. This makes Λ to an indicator that can be used to perform further research in this area.

The Λ-hyperon can decay in different types of particles. The branch ratios, which is the probability of the decay modes, of possible Λ-decays are given in Figure 3.2. The first decay, $\Lambda \rightarrow p\pi^-$, is discussed within this thesis, as it is the only 2-daughter decay with two charged daughter particles.

$\Lambda \rightarrow p\pi^-$	(BR: 63.9 ± 0.5)
$\Lambda \rightarrow n\pi^0$	(BR: 35.8 ± 0.5)
$\Lambda \rightarrow n\gamma$	(BR: $(1.75 \pm 0.15) \cdot 10^{-3}$)
$\Lambda \rightarrow p\pi^-\gamma$	(BR: $(8.4 \pm 1.4) \cdot 10^{-4}$)
$\lambda \rightarrow pe^-\bar{\nu}_e$	(BR: $(8.32 \pm 0.14) \cdot 10^{-4}$)
$\Lambda \rightarrow p\mu^-\bar{\nu}_\mu$	(BR: $(1.57 \pm 0.35) \cdot 10^{-4}$)

Figure 3.2: The branch ratios of the possible decays of Λ [45, p. 1127], after its lifetime of approximately $\tau = (2.632 \pm 0.020) \times 10^{-10} s$.

3.1 Functionality of KF Particle Finder Package

The KF Particle Finder package was developed in C++ for the online reconstruction and selection of short-lived particles. The package is fully vectorized and parallelized to provide the ability for a fast online reconstruction on a high performance computer cluster. Additionally, it has a robust linear scalability on multi-core architectures with at least as many as 80 cores. [4, p.36,p.112, p.117]

The package is based on the Kalman Filter method and is able to search more than 150 decay channels [46], which can be seen in Figure 3.3.

For the reconstruction, the particle tracks and particles created by FLES are divided into primary and secondary tracks. Primary tracks are the tracks that can be extrapolated to the primary collision point. Secondary tracks are those that belong to decay products of short-lived particles. Both, secondary and primary tracks are subdivided into positively and negatively charged tracks. This subdivision is necessary for the reconstruction of the respective particles. In order to be able to reconstruct the short-lived particles, the possible daughter tracks are extrapolated to find possible points of decay, so called secondary vertices. Then, all possible mother particle candidates for that specific decay are reconstructed in the package. While this approach allows to find all possible occurrences of particles, the multiple creation of mother particle candidates hinders the physics analysis, since it is not clear which particle really decayed. To solve this problem in the KF Particle Finder package, statistical cuts are applied to reject particles that have a high probability to be mistakenly reconstructed. Additionally, a particle competition can be enabled to find the best fitting mother particle candidates and to reject wrongly reconstructed ones, to improve the physics analysis quality.

Generally, with respect to the reconstructed particles, the KF Particle Finder package is designed in a recursive way. Starting from tracks and their respective particles, the package is also capable to reconstruct decay chains by handling reconstructed mother particles the same way. By that, reconstructed mother particle candidates and their extrapolated tracks can be used to find decays, where a mother particle was a decay product and is therefore also considered as a daughter particle at the same time.

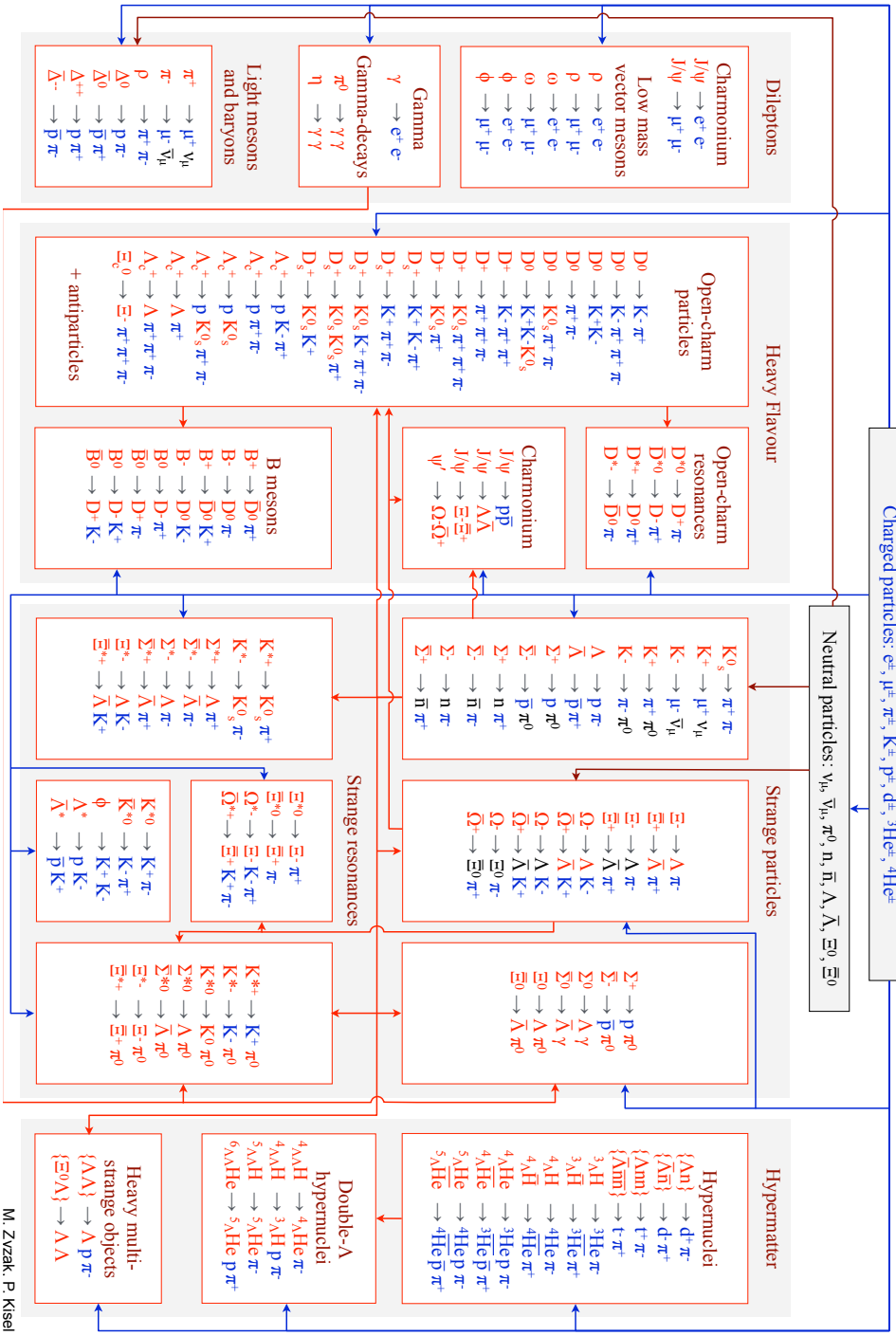


Figure 3.3: Block diagram of the reconstructable decays of the KF Particle Finder package [47, p. 22]

The KF Particle Finder package offers possibilities to measure its own performance. In the case of simulated data, as is currently the case for the CBM experiment, due to the unfinished particle accelerator, it is possible to compare this performance with the simulated data. Then, an efficiency table and histograms of all reconstructed events are created as an output. The four most important histograms for performance analysis within the discussions of this thesis are the histograms for all reconstructed Λ particles, the histogram of Λ combinatorial background, physical background and reconstructed signal. The last three are compared against the simulated data, and therefore, compared with the expected outputs of the package. The first histogram shows all reconstructed Λ particles and, therefore, shows the results that would be the output in a real experiment. The physical background histogram shows the particles that were reconstructed as Λ, but a comparison to MC information showed that they were actually a different particle. Combinatorial background is another type of noise, that is created by particles that were reconstructed but never existed. This is especially the case for the already mentioned mother particles, were all possible mother particles are created, but only a single one of them is the correct one. If cuts and the competition are not sufficient to reject these particles, they produce noise in the reconstructed Λ distribution. Signal, on the other hand, shows correctly reconstructed Λ particles that were confirmed by MC information.

3.2 Structure of the Package

The KF Particle Finder package is comprehensive, so this section provides a broad summary rather than a detailed breakdown of its architecture. To keep within the focus of this study, some classes are not discussed. The package is organized into two main directories: `KFParticle` and `KFParticlePerformance`. The `KFParticle` directory includes the main classes, which focuses on fundamental aspects like particle reconstruction. In the following, the focus will lay on the classes mentioned below:

- `KFParticle` and `KFParticleBase`
- `KFParticleSIMD` and `KFParticleBaseSIMD`
- `KFParticleTopoReconstructor`
- `KFParticleFinder`

The `KFParticlePerformance` folder contains classes that assess the performance of the KFParticle package. Regarding this directory, the classes `KFTopoPerformance` and `KFPartEfficiencies` will be shortly discussed. The whole package can be found in the CbmRoot [48].

The primary scalar class **`KFParticle`** in the KF Particle package serves to represent a particle object. A particle is specified by a state vector consisting of its spatial coordinates (X, Y, Z), momentum (Px, Py, Pz), energy level, and S value (defined as decay length divided by momentum). This is supplemented by an associated matrix to estimate covariance, which allows to calculate the χ^2-criterion, which is utilized to gauge the quality of the reconstructed output [4, p.65]. The class includes functionalities for forming particle objects from given tracks and for creating short-lived particles

using either other tracks or pre-existing particles. The mathematics used in this class is based on the principles of Kalman filter. In the context of this thesis, it is significant to point out that this class holds various parameters like the unique ID of the particle, daughter particle IDs, the χ^2-value, and the number of degrees of freedom (NDF).

KFParticleSIMD acts as the vectorized form of the original KFParticle class and is derived from KFParticleSIMDBase. It offers functionalities comparable to its scalar counterparts. By leveraging the Single Instruction, Multiple Data (SIMD) design, this class enables parallel processing, carrying out the same tasks on numerous data points simultaneously, which results in faster computations.

The Particle Data Group (PDG) has created a list in which all known particles have been assigned a so-called PDG code, which is also used in the KFPF to identify the particles [45]. Beside the PDG code that is a unique particle ID, the PDG mass is also assigned. The PDG mass refers to the mass a specific particle is expected to have. Each particle in the package is assigned a PDG hypothesis that states which particle it presumably is. For Λ the PDG code is 3122, for the daughters π^- and p^+ -211 and 2212 respectively. All possible PDG codes are listed in the **KFPartEfficiencies** class in the KFParticlePerformance package. Each particle has one hypothesis.

The **KFParticleTopoReconstructor** class receives the track data from the track finder with the PDG hypothesis as an input. It then reconstructs potential primary vertices, categorizes tracks as either primary or secondary, and further sorts them into positive and negative tracks based on the PDG hypothesis. After that, short-lived particles are constructed, and optionally, a comparison is performed among various particle hypotheses for the constructed candidates. The reconstruction of short-lived particles is done by calling the main interface method FindParticles of the KFParticleFinder class, which runs the reconstruction of short-lived particles. All reconstructed particles are saved in the array fParticles defined in the header file of KFParticleTopoReconstructor.

The **KFParticleFinder** class is for the reconstruction of short-lived particles by combining long-lived and previously reconstructed short-lived particles. It requires tracks and primary vertices as an input. It also uses hardcoded cuts, which are defined in this class, like χ^2_{geo}, χ^2_{prim}, $l/\Delta l$ and a few more for different approaches. These cuts are used as threshold parameters for particle selection. The defined cuts have default values, those values can be overwritten, depending on the experiment, as it is done for a few one for the CBM experiment. It is possible to get each reconstructed particle, since they are all stored together in one array in the KFParticleFinder. For the reconstruction of short-lived particles with one neutral daughter, the missing mass method [24] is implemented.

In the **KFTopoPerformance** class compiles various histograms for each particle identified by the KF Particle reconstruction scheme. These histograms cover parameter distribution, efficiencies, fit quality, and other metrics, including those related to primary vertex quality.

Chapter 4

Artificial Neural Networks

In August 1955, John McCarthy, Marvin L. Minsky, Nathaniel Rochester and Claude E. Shannon made ''A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence'', which took place in 1956 [49]. In this proposal they defined the term artificial intelligent and some aspects of the artificial intelligence problem. The study was ''[...] to proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it'' [49, p.2]. This workshop is considered as the start of the research field Artificial Intelligence (AI).

Machine Learning (ML) is a subfield of AI and describes the capability to extract patterns from raw data. This is necessary for AI systems to acquire their own knowledge [50, p.2]. To be able to extract patterns from raw data, the algorithm or the machine learning model has to learn those patterns. There are different types of learning, the best known of which will be described in more detail later.

One subfield of ML is the field of Artificial Neural Networks, for whose the single perceptron of Rosenblatt [51] was one of the origins. In the early years, research stopped due to discovered limitations of the model. While these problems were solved by more complex models (e.g. Multilayer Perceptron (MLP)s), the limitations in computational power hindered deep research in the area for several years. However, in the last decades, since multi-core CPUs and GPUs have become more powerful, research on neural networks has surged. Not only one is capable to chose models with a large complexity that outperformed existing approaches, such as convolutional neural networks [52], recurrent neural networks [53] or deep neural networks [54]. Additionally, it is possible to use data sets with millions of different input features [55] that help to create a neural network with unprecedented rates of accuracy for complex problems.

4.1 The Perceptron by Rosenblatt

The current understanding of a neural network is based on the single perceptron by Rosenblatt [51]. The single perceptron for solving a binary problem can be seen as one single neuron with n inputs $a_0, a_1, ..., a_{n-1}$. Each input a_i has one weighted connection to the neuron. The weight of input i to the neuron will be noticed in the following as w_i. The neuron can assume two states: active and inactive. If the neuron

is active, it will output the value 1, if it is inactive 0. The neuron is active when $\sum_{i=0}^{n-1} w_i a_i \geq \theta$, while θ is a threshold, otherwise the neuron is inactive. [56, p. 60, Definition 1]

One of the best known problems of a single perceptron is the XOR problem and with it the problem of linear separability, which is described in the following.

A problem or function is linear separable if values $w_0, ..., w_n$ exists with $w_i \in \mathbb{R}$ for subsets $B \subseteq \mathbb{R}^{n-1}$, $\vec{b} = (b_0, ..., b_{n-1}) \in B$ and $C \subseteq \mathbb{R}^{n-1}$, $\vec{c} = (c_0, ..., c_{n-1}) \in C$ such that $\sum_{i=0}^{n-1} w_i b_i \geq w_n > \sum_{j=0}^{n-1} w_j c_j$ applies [56, p. 63, Definition 2].

A visualization of a linear separable function is shown in Figure 4.1.

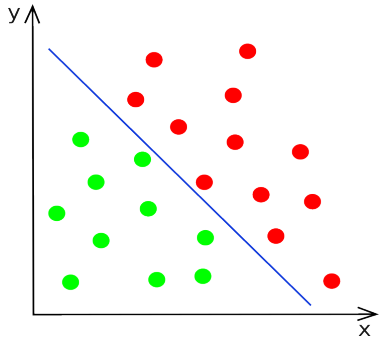


Figure 4.1: Graph to visualize a linear separable function. In this 2-dimensional case, it is possible to separate all green and red data points by a 1-dimensional hyperplane (line).

The OR-Problem with the input $\vec{a} = (a_0, a_1)$ with $a_0 \in \{0, 1\}$ and $a_1 \in \{0, 1\}$ can be solved by one single perceptron as this problem is linear separable. With the definition of a single perceptron we get the following equation:

$$Output(\vec{a}) = \left\{ \begin{array}{ll} 1 & for \ \sum_{i=0}^{1} a_i w_i \geq \theta \\ 0, & for \ \sum_{i=0}^{1} a_i w_i < \theta \end{array} \right\}$$

In the case of the OR-Problem $Output(\vec{a})$ should be 0 if $a_0 = 0$ and $a_1 = 0$ otherwise it should be 1. When considering each combination, one can see that it is possible to solve this problem with a simple perceptron:

$$\vec{a} = (0, 0); 0 \cdot w_0 + 0 \cdot w_1 = 0 < \theta \tag{4.1}$$
$$\vec{a} = (0, 1); 0 \cdot w_0 + 1 \cdot w_1 = w_1 \geq \theta \tag{4.2}$$
$$\vec{a} = (1, 0); 1 \cdot w_0 + 0 \cdot w_1 = w_0 \geq \theta \tag{4.3}$$
$$\vec{a} = (1, 1); 1 \cdot w_0 + 1 \cdot w_1 = w_0 + w_1 \geq \theta \tag{4.4}$$

If w_0 and w_1 are each greater or equal than the threshold θ to fulfill equation 4.2 and 4.3, than also $w_0 + w_1 \geq \theta$, which means that the definition of the OR-Problem

is fulfilled.

But for the XOR-Problem, which is not linear separable, the following equations are required:

$$\vec{a} = (0,0); 0 \cdot w_0 + 0 \cdot w_1 = 0 < \theta \tag{4.5}$$
$$\vec{a} = (0,1); 0 \cdot w_0 + 1 \cdot w_1 = w_1 \geq \theta \tag{4.6}$$
$$\vec{a} = (1,0); 1 \cdot w_0 + 0 \cdot w_1 = w_0 \geq \theta \tag{4.7}$$
$$\vec{a} = (1,1); 1 \cdot w_0 + 1 \cdot w_1 = w_0 + w_1 < \theta \tag{4.8}$$

However, since there is only one threshold, it is impossible to solve. If w_0 and w_1 are each greater or equal than the threshold θ to fulfill equation 4.6 and 4.7, then $w_0 + w_1 \geq \theta$ which is contrary to equation 4.8.

The concept of the MLP is a neural network model which solves this problem by adding more layers, neurons and non-linear activation functions.

4.2 Multilayer Perceptron

A Multilayer Perceptron (MLP) consists of three or more layers with neurons, one *input layer*, one *output layer* and one or more *hidden layer* between the input and output layer. Previously, the input was not considered as a separate layer. However, when describing multilayer perceptrons, it is often mentioned as a layer.

The input layer is the first layer of the MLP. The neurons of this layer hold the input data which consists of so-called input features on which the neural network is trained on. The number of neurons of the input layer is equal to the number of input features, e.g. the pixel values of an image to be classified or data points on which a regression should be performed. The layer directly after the input layer is the hidden layer. The neurons of this layer are calculated by the weighted sum of neuron outputs received from the previous layer, that are then activated by a non-linear activation function. Non-linear activation functions are crucial in multilayer perceptrons to provide the network with the capability to approximate complex, non-linear mappings from inputs to outputs. Using only linear activation functions would limit the network to linear transformations and render the use of multiple layers unnecessary, as the composition of linear functions is itself linear [50, p. 167]. Nonlinear activation functions introduce the necessary complexity and allow the network to learn high-level abstractions in the data. Therefore, nonlinear activation functions are indispensable to solve complex problems that are not linearly separable. The last layer is the output layer. The number of the output neurons depends on the output that is wanted to produce with the neural network. For example, a network which should solve a classification problem with three possible classifications a, b, c would have three neurons, one neuron per class. The output values would correspond to the predicted class, whereas the exact outcomes and interpretation is depending on the

activation function of the output layer.

In a fully connected neural network, each neuron of a layer of the MLP has weighted connections to each neuron of the next layer, which is visualized in Figure 4.2. The connections of the neurons of an MLP are not recurrent, that means that they only lead from the current neuron to the neuron of the next layer. Usually, weights are represented as floating point values, such that each weight can be positive, negative or neutral. A positive weight is an excitatory weight, whereas a negative weight is an inhibitory weight. A weight which has a value of zero is a neutral weight, which means that the neuron from which the weight emanates does not influence the following layer and thus has no impact on the final output.

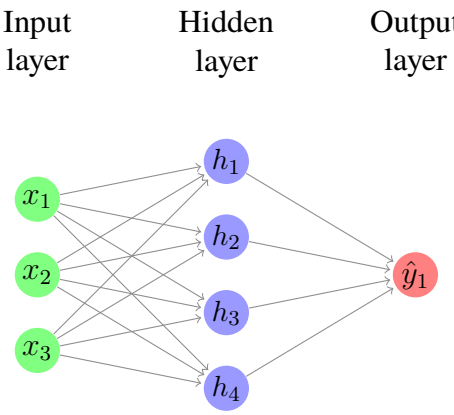


Figure 4.2: Representation of an MLP with an input layer of 3 neurons (green), one hidden layer of 4 neurons (blue) and an output layer of 1 neuron (red)

Using a MLP the input features have to be propagated forward through the network. For training, an optimization algorithm is often used together with backpropagation, to propagate the error back to the input layer and thus allow to adjust all weights of the network in a way that the network actually learns the patterns of given training data. These algorithms will be explained in the following sections.

4.3 Forward Propagation

As each layer consists of multiple neurons, one can represent each layer as a vector of neurons and the weights between two layers $k-1$ and k as a weight matrix $w^{(k)}$. As the input layer has no previous layer and also normally no activation function, the neurons of the input layer include only the input features, which are propagated directly to the next layer by multiplying the input feature vector with the weight matrix. For each neuron of other layers than the input layer, the following structure applies:

The input $a_i^{(k)}$ of neuron i of the current layer k is calculated by summarizing the bias $b_i^{(k)}$ with the sum over all neurons from the previous layer $k-1$ multiplied with the weight of each connection [50, p.205].

$$a_i^{(k)} = b_i^{(k)} + \sum_{j=0}^{n^{(k-1)}-1} h_j^{(k-1)} \cdot w_{ij}^{(k)}$$

with $a_i^{(k)}$:= input of neuron i of layer k
$n^{(k-1)}$:= number of neurons of the previous layer $k-1$;
$h_j^{(k-1)}$:= output of neuron j of the previous layer $k-1$;
$w_{ij}^{(k)}$:= weight between neuron i and j;
$b_i^{(k)}$:= bias for neuron i of layer k

The *bias* is a neuron which additionally influences the neuron $a_i^{(k)}$. It has no input and an activation level which is always 1. The bias has weighted connections between the neurons of one layer, whose weights can be positive or negative. [57, p. 29]

Since the impact of the bias neuron on the next layer's neurons is only depending on the weight, its weights allow shifting the values of the next layer with a set of trainable parameters. This procedure allows the MLP to represent functions of higher complexity and, thus, to improve its capability and performance.

Subsequently, neuron i of layer k will be assigned to an activation level by using an activation function f [50, p.205].

$$h_i^{(k)} = f(a_i^{(k)})$$

with $h_i^{(k)}$:= output of neuron i of layer k;
$f(a_i^{(k)})$:= activation level of neuron i of layer k

The activation functions have severe impact on the neural networks' performance. The features that have to be learned by the model are depend on the specific task, but the capability of the function the model represents is depending on the activation functions used. That means, whether the model can represent the data and learned features in a good way is depending on the chosen activation functions. Furthermore, the use of specific activation functions allow interpreting the results in a comfortable way. For instance in classification problems, Softmax activation is often preferred for the output layer, since the output can be interpreted as a probability distribution of the predicted classes.

However, not any non-linear activation function can be chosen for the activation of neurons. Since for a gradient descent based approaches the calculation of the gradients is mandatory, the function has to be differentiable in every point. In some cases, e.g. for Rectified Linear Unit or Leaky Rectified Linear Unit activation, which are not differentiable for $a_i^{(k)} = 0$, one can solve this problem by logical arguments. Here, $a_i^{(k)} = 0$ leads to a neuron output of 0 and, therefore, the neuron has no impact to following layers nor the final result. Thus, one can argue that a gradient of 0 for $a_i^{(k)} = 0$ is reasonable, such that the neuron has no impact but also no weight update. For any other point, these functions are differentiable.

4.4 Neural Network Training Paradigms and Key Terminologies

As mentioned before, machine learning is the capability to extract patterns from raw data and must acquire their own knowledge to achieve this. This acquisition of knowledge is called learning. There are different types of learning, where 3 types are the most known: *supervised learning*, *unsupervised learning* and *reinforcement learning*. Also, neural networks, as a part of machine learning, use these types. Before getting more in detail in the learning process of neural networks, the three types are shortly explained.

In *supervised learning*[58, p.9, et seq.], labeled examples are provided to the model or algorithm. These labeled examples consist of input data and corresponding desired output, also referred to as labels or target values, that the model should learn to predict. The goal of the model is to be able to make accurate predictions on new, unseen data. During the training phase, the model is presented with the labeled examples. The model adjusts its internal parameters to minimize the difference between its predictions and the truth provided by labels in the training data. This process involves optimization algorithms that fine-tune the model's parameters to reduce prediction errors.

Unsupervised learning [58, p.485, et seq.] focuses on finding patterns, structures, or relationships in data without using labeled examples. In other words, in unsupervised learning, the algorithm explores the inherent structure of the data without being guided by predetermined correct answers. Unsupervised learning comes into play, for example when one wants to uncover hidden patterns, group similar data points, or reduce the dimensionality of the data. One common task in unsupervised learning is clustering. The goal is to group similar data points together into clusters, where points within the same cluster are more similar to each other than to points in other clusters. This helps in discovering natural groupings or segments within the data. One well known neural network for solving this is the Self Organizing Map (SOM) [59].

Reinforcement learning [60] is about training agents to make a series of decisions in an environment to maximize cumulative rewards. This learning paradigm is inspired by behavioral psychology, where agents learn through trial and error how to interact with their environment to achieve desired outcomes. Thus, the agent performs an action, the environment responds, and the agent learns from the results to improve its decision strategy over time. In reinforcement learning, one has two main components: the agent and the environment. The agent is the learner that takes actions, whereas the environment is the context in which the agent operates. The environment responds to the agent's actions by rewards or penalties, and this interaction generates the agent's experiences.

To discuss the training more in detail, the terms epochs and batches should be introduced first. An epoch refers to a single pass through the entire data set during the training process. The network processes all available training data in a sequential or random order, depending on the selected training strategy. During training, usually

several epochs are used, so that the parameters keep adjusting to improve the performance of the model. Using basic weight optimization algorithms, often repeated iterations can result in overfitting. In this state, the neural network begins to learn the noise present in the training data rather than the underlying patterns. Therefore, the network has lost its ability for generalization and will perform worse on unseen data.

A batch, on the other hand, is a subset of the entire data set. Batches are used to update the weights already during an epoch and not only after a pass of the whole data set. This has several advantages. Firstly, some data sets are too large to fit into the computers' memory, leading to problems with training on the whole data set. Secondly, each batch element is independent of another, such that parallel processing is possible to perform faster training on modern multi-core CPUs or GPUs.

The whole learning process of a neural network can be separated mainly in three phases: the training phase, the validation phase and the test phase. The training phase is the phase in which the neural network adjusts its weights, which means that in this phase, the network learns the correct parameters for each connection based on the training dataset. The validation phase is used to tune the hyperparameters, i.e., the non-trainable parameters that define the model, such as the number of neurons, the network depth, or the learning rate. It consists of a subset of the full dataset that is not used for training. It is used to measure the network's performance on unseen data and can help to identify overfitting. During the test phase, the neural network is evaluated using again unknown input features (another unseen subset of the full data set) and thus it can be determined how well the neural network solves the problem. The difference between the validation and testing data set is that the final model is usually chosen by the validation phase results. However, this decision can be biased, since the model with the best results on the validation set is chosen. To avoid this biased decision, the testing set is used to see the final performance on unseen data again.

4.5 Training by Supervised Learning

If the neural network learns by means of supervised learning, it receives the correct output as feedback, the already mentioned true labels. This means, that for each data set of input features, the correct values for the output has to be available, so that these can then be compared with the values calculated by the network. For comparison, a so-called loss (also error or cost) is calculated by a loss function, which takes the output of the network and the true labels as an input. The loss function evaluates the difference of calculated output values and the target values. How the weights are updated is determined by the choice of the optimization algorithm and the calculated loss.

One popular loss function for classification problems is the cross-entropy, which is also employed in the presented MLP in this work and will be first discussed before delving deeper into the training of the MLP.

4.5.1 Cross-Entropy

The cross-entropy loss function combines a measure of uncertainty with the ability to evaluate the difference between two probability distributions. In the context of neural networks and supervised learning, the computed outputs of the neural network and the true labels each form a probability distribution. This makes the cross-entropy a suitable loss function to evaluate the network's performance in a supervised classification task as it is presented in this work.

The theory behind cross-entropy is that rare events are considered more informative than frequent ones. A function that fulfills this condition is the one that describes the self-information [50, p. 71]:

$$I(x) = -log(P(x))$$

with $x :=$ event, $P(x) :=$ probability of x

With this definition, a guaranteed event with probability of 1 results in a self-information of 0, whereas a rare event has a huge self-information, rapidly increasing with a lower probability. To apply this theory to multiple events, one can take the expected value of the information content, which corresponds to the Shannon entropy [50, p. 71]:

$$\begin{aligned} H(P) = H(x) &= E_{x\sim P}[I(x)] && (4.9)\\ &= \sum_x P(x)I(x) && (4.10)\\ &= \sum_x P(x) \cdot -log(P(x)) && (4.11)\\ &= -\sum_x P(x) \cdot log(P(x)) && (4.12) \end{aligned}$$

with $x :=$ event, $P(x) :=$ probability of x, $I(x) :=$ self-information of x

The Kullback-Leibler divergence (KL divergence) calculates how different two separate probability distributions $P(x)$ and $Q(x)$ are over the same event x [50, p. 72] [61]:

$$\begin{aligned} D_{KL}(P||Q) &= E_{x\sim P}[log(\frac{P(x)}{Q(x)})] && (4.13)\\ &= \sum_x P(x) \cdot log(\frac{P(x)}{Q(x)}) && (4.14) \end{aligned}$$

with $x :=$ event, $P(x) :=$ probability P of x, $Q(x) :=$ probability Q of x

Using supervised learning for neural networks, one has the calculated output values of the network and the target values which have to be compared. Cross-entropy uses the KL divergence and Shannon entropy together to calculate the dissimilarity

of the output values and the target values in consideration of the information content [50, p. 73]:

$$H(P,Q) = H(P) + D_{KL}(P||Q) \tag{4.15}$$
$$= -\sum_x P(x) \cdot log(P(x)) + \sum_x P(x) \cdot log(\frac{P(x)}{Q(x)}) \tag{4.16}$$
$$= \sum_x -P(x) \cdot log(P(x)) + P(x)log(P(x)) - P(x)log(Q(x)) \tag{4.17}$$
$$= -\sum_x P(x)log(Q(x)) \tag{4.18}$$

with $P := target\,probability\,distribution$, $Q := predicted\,probability\,distribution$, $x := event$

In the case of neural networks, $Q(x)$ is the activation level of the neuron at the time of the event x.

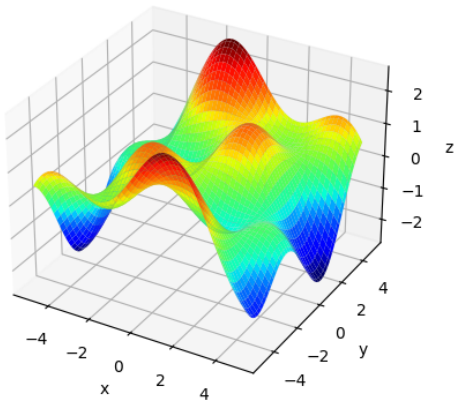
Related to a classification problem, where only a single class can be true at a time, $P(x)$ is only 1 if x is the correct class and for all other cases $P(x)$ is 0. Due to the multiplication with 0 for the neuron with target value 0, the loss can be only calculated for the neuron with target value 1. For that reason, a simplified implementation of the cross-entropy loss can be chosen: the binary cross-entropy. However, this does not change cross-entropy formula in general, but is rather a simplification.

Since Softmax returns a probability distribution of the predicted classes, cross-entropy is a good choice in combination with the Softmax activation function. With a gradient descent based weight optimization, the gradient of the loss function can then be used to adjust the weights in such a way that the loss is minimized over time. As a result, the network learns the patterns and classifies correctly more often.

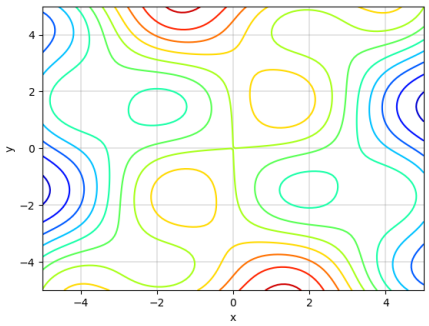
4.5.2 Backpropagation

Taking a look at a 3-dimensional graph of a simplified possible loss function, one can see, that it has higher regions and lower regions. The objective is to reach the minimum of the loss function to minimize the network's error. In Figure 4.3, one can see a graphical example of a possible loss function in $\mathbb{R}^3$ for two weights. One popular approach to train a MLP is using the so-called *backpropagation*. This is a method based on Rumelhart et al. [62] for calculating the gradient of the loss function with respect to the weights of the network. That way, weights can be adjusted to find the minimum of the loss. Since it is only possible to adjust the weights, the actual image of the loss function space that can be explored is a 2-dimensional contour plot as shown in Figure 4.3. Here, the colors visualize the depth in 3-dimensional space where a minimum should be found. The shortest distance between two levels show the steepest descent in 3-dimensional space.

Since during training it is only possible to adjust the weights, the gradient of the loss function is calculated with respect to each weight in the neural network. As each



(a) 3D graphic of a possible loss function



(b) 2D contour plot of the 3D loss function

Figure 4.3: Visualization of a simplified possible loss function in $\mathbb{R}^3$ and its contour plot with $x := weight_{00}$, $y := weight_{01}$, $z := loss$

neuron of layer k is connected to each of layer $k-1$, the gradients of each neuron of one layer are stored in a vector. The computation of the gradient is described below. To simplify the explanations, from here on, a neural network without bias neurons is assumed.

The gradient vector is

$$\nabla \vec{E} = \left[\frac{\partial E}{\partial w_{0j}^{(k)}}, \vec{\dots}, \frac{\partial E}{\partial w_{(n-1)j}^{(k)}} \right] \tag{4.19}$$

with $E := Loss$,
$n :=$ number of neurons in layer k,
$w_{ij}^{(k)} :=$ weight between neuron i of the current layer and neuron j of the previous layer.

Since $E = L(P, f(a^{(k)}))$, with L defined as the loss function, $f(a^{(k)})$ defined as the activation levels of all neurons of the output layer k
and $f(a_i^{(k)}) = f(\sum_{j=0}^{n^{(k-1)}-1} h_j^{(k-1)} \cdot w_{ij}^{(k)})$ as the activation level of neuron i of layer k, the chain rule can be applied twice to get the following term [62]:

$$\frac{\partial E}{\partial w_{ij}^{(k)}} = \frac{\partial E}{\partial f(a_i^{(k)})} \cdot \frac{\partial f(a_i^{(k)})}{\partial w_{ij}^{(k)}} \tag{4.20}$$

$$= \frac{\partial E}{\partial f(a_i^{(k)})} \cdot \frac{\partial f(a_i^{(k)})}{a_i^{(k)}} \cdot \frac{\partial a_i^{(k)}}{\partial w_{ij}^{(k)}} \tag{4.21}$$

Let $\delta_i^{(k)}$ be defined as $\delta_i^{(k)} := \frac{\partial E}{\partial f(a_i^{(k)})} \cdot \frac{\partial f(a_i^{(k)})}{a_i^{(k)}}$.

For calculating $\delta_i^{(k)}$ it is necessary to take into account that the MLP can have one or multiple hidden layers. In this case, neurons of the inner layers contribute to the final output indirectly because their activations pass through multiple intermediate layers before reaching the output layer. This indirect impact means that when calculating the gradient of the loss with respect to the connections (weights and biases) of these neurons, one must consider how changes in their parameters affect the output through all the intermediate steps. Therefore, the computation of these gradients has to be divided into two conditions: k is an output layer and k is a hidden layer. While the calculation of $\delta_i^{(k)}$ for the output layer is clearly evident, the computation for a hidden layer is more complicated due to the indirect impact and is shown below:

Let k be the hidden layer, then

$$\delta_i^k = \frac{\partial E}{\partial f(a_i^k)} \cdot \frac{\partial f(a_i^{(k)})}{a_i^{(k)}} \tag{4.22}$$

Since E is the loss and in a fully-connected MLP each neuron of layer k is connected with each neuron of the next layer $k+1$, the following function holds for E of the hidden layer k with the output layer k+1:

Let $L_j(P_j, f(a_j^{(k+1)}))$ be defined as the loss of neuron j of the output layer $k+1$.

$$E = \sum_{j=0}^{n^{(k+1)}-1} L(P_j, f(a_j^{(k+1)})) \tag{4.23}$$

Therefore applies:

$$\frac{\partial E}{\partial f(a_i^{(k)})} = \sum_{j=0}^{n^{(k+1)}-1} \frac{\partial L(P_j, f(a_j^{(k+1)}))}{\partial f(a_i^{(k)})} \tag{4.24}$$

It is known that

$$f(a_j^{(k+1)}) = \sum_{t=0}^{n^{(k)}-1} f(a_t^{(k)}) \cdot w_{jt}^{k+1} \tag{4.25}$$

with

$$f(a_t^{(k)}) = \sum_{l=0}^{n^{(k-1)}-1} f(a_l^{(k-1)}) \cdot w_{tl}^{(k)} \tag{4.26}$$

From this results:

$$\frac{\partial E}{\partial f(a_i^{(k)})} = \sum_{j=0}^{n^{(k+1)}-1} \frac{\partial L(P_j, f(a_j^{(k+1)}))}{\partial a_j^{(k+1)}} \cdot \frac{\partial a_j^{(k+1)}}{\partial f(a_i^{(k)})} \tag{4.27}$$

$$= \sum_{j=0}^{n^{(k+1)}-1} \frac{\partial L(P_j, f(a_j^{(k+1)}))}{\partial f(a_j^{(k+1)})} \cdot \frac{\partial f(a_j^{(k+1)})}{\partial a_j^{(k+1)}} \cdot \frac{\partial a_j^{(k+1)}}{\partial f(a_i^{(k)})} \tag{4.28}$$

from 4.25 follows:

$$\frac{\partial E}{\partial f(a_i^{(k)})} = \sum_{j=0}^{n^{(k+1)}-1} \underbrace{\frac{\partial L(P_j, f(a_j^{(k+1)}))}{\partial f(a_j^{(k+1)})} \cdot \frac{\partial f(a_j^{(k+1)})}{\partial a_j^{(k+1)}}}_{\delta_j^{(k+1)}} \cdot w_{ji}^{(k)} \tag{4.29}$$

With the definition of δ_i^k in (4.22), δ_i^k for the hidden layer k is defined as

$$\delta_i^k = \left(\sum_{j=0}^{n^{(k+1)}-1} w_{ji}^{(k)} \cdot \delta_j^{(k+1)} \right) \cdot \frac{\partial f(a_i^{(k)})}{a_i^{(k)}}$$

Therefore the following function is valid for δ:

$$\delta_i^{(k)} = \begin{cases} \frac{\partial L(P_i, f(a_i^{(k)}))}{\partial f(a_i^{(k)})} \cdot \frac{\partial f(a_i^{(k)})}{a_i^{(k)}} & \text{if } k \text{ is an output layer,} \\ (\sum_{j=0}^{n^{(k+1)}-1} w_{ji}^{(k)} \cdot \delta_j^{(k+1)}) \cdot \frac{\partial f(a_i^{(k)})}{a_i^{(k)}} & \text{if } k \text{ is an hidden layer} \end{cases} \tag{4.30}$$

And thus the gradient vector is:

$$\nabla E = \left[\delta_0^{(k)} \cdot \frac{\partial a_0^{(k)}}{\partial w_{0j}^{(k)}}, \ \vec{\dots}, \delta_{n-1}^{(k)} \cdot \frac{\partial a_{n-1}^{(k)}}{\partial w_{(n-1)j}^{(k)}} \right] \tag{4.31}$$

This also shows clearly where the name backpropagation comes from, namely from the back propagation of the error from the last to the first layer.

This gradient is used in several *optimization algorithms* to modify the weights.

4.5.3 Optimization Algorithm

Assuming the previous use of backpropagation, the optimization algorithm is used to change the weights based on the gradient to find the minimum of the loss function. The gradient indicates the direction for the steepest descent. One of the most popular optimization algorithm is the *gradient descent*. The algorithm itself is seen in Figure 4.4.

With gradient descent, each updated weight is calculated as follows:

$$w_{ij}^{(k,new)} = w_{ij}^{(k,old)} - \Delta w_{ij}^{(k)} \tag{4.32}$$

with

$$\Delta w_{ij}^{(k)} = \eta \cdot \delta_i^{(k)} \cdot \frac{\partial a_i^{(k)}}{\partial w_{ij}^{(k)}} \tag{4.33}$$

η is the learning rate; it is controlling how much the weight is adjusted with each update and has a significant impact on the learning process.

1) initialize the network with weights (usually randomized)

2a) forward propagation

2b) calculation of the loss

2c) backpropagation

2d) update the weights according to the optimization algorithm

3) repeat all steps from 2 until a termination criterion fulfills or the performance of the network is satisfactory

Figure 4.4: Algorithm of the neural network training, using gradient descent.

The algorithm shown in Figure 4.4 describes a weight update after each forward propagation. This type of training is called *incremental training* [57]. The gradient descent algorithm of this type of training is called *stochastic gradient descent (SGD)*. One advantage of it is, that it performs fast and frequent updates, and therefore, features a fast learning process. However, this also leads to the fact that the weights vary largely due to the large amount of updates and the loss function fluctuates. On the one hand, this ensures that a local minimum can partly be left again, but on the other hand, the SGD might overshoot the minima repeatedly, which makes it difficult to reach the exact minimum. [63]

Using *batch training* in general, each weight is updated after a batch [57]. There are two types of it in the case of gradient descent. The first one is the *batch gradient descent*, also referred to as *vanilla gradient descent*. Using this, the weights are updated after the whole training data set. For this, the gradients of the whole set are summed up for a single update. [63]

This also means that if the data sets are too large, the memory may not be sufficient and thus vanilla gradient descent is not feasible due to memory limitations. Another disadvantage is that this algorithm is slow, since the whole dataset has to be seen before the update [63]. On the other hand, is the learning behavior more stable than using SGD [63], since the weight update is based on all training data and not after seeing a single input.

The other type of *batch training* in the case of gradient descent is the *mini-batch gradient descent*. Instead of updating the weights after the whole epoch, *mini-batch gradient descent* updates the weights after a smaller batch of n training samples [63]. Therefore, it can be seen as a combination of SGD and vanilla gradient descent, with the best benefits of both. Due to the amount of information that can be processed at once, the learning behavior is more stable than using SGD. Also, the memory problem can be avoided, since a smaller batch size requires less memory at a time. Another factor is that the batch size is also an additional hyperparameter that can be

adjusted. The network can be trained fast and target-oriented. Furthermore, since each input is independent of each other, it can easily be parallelized for faster computations. Because of all these arguments, this type of gradient descent is considered as a popular optimization method.

There exists multiple other algorithms that are an optimization or extension of the gradient descent. One such algorithm is *Adaptive Moment Estimation (Adam)* [64], which is used in the implemented MLP for solving the problem of this thesis. Adam is a combination of a decaying learning rate that can help to avoid overfitting and momentum based weight updates. In Adam, each connection has a 1st and 2nd momentum that is calculated with each weight update. This is used to improve the learning process even further when neurons sometimes have an output value of 0, their weights are still trained. Additionally, if an input feature contains noise and the noise leads to the fact that the neuron fires, even if it usually should not, the momentum can hold against the wrong weight update. Thus, the momentum can help to stabilize the learning process.

Chapter 5

Optimization of Λ Signal/Background Ratio

The goal of the present work is to optimize the Λ signal/background ratio for CBM, using a neural network. The signal/background ratio indicates the ratio of correctly reconstructed particles to those particles which are incorrectly reconstructed or incorrectly classified. The incorrectly reconstructed particles as well as the incorrectly classified particles are background. The first one is the combinatorial background, reconstructed particles that do not exist. The second one are particles which exists but were incorrect classified.

To enhance the physics analysis quality of the CBM experiment, minimizing the combinatorial background, referred to as *ghosts*, is essential, as these reconstructed particles do not exist. Consequently, these particles offer no additional physical insights and only serve as confounding variables when assessing the event's outcomes. In addition, however, it is important to ensure that the number of reconstructed real Λ particles does not decrease significantly.

The deep neural network for this approach is created using the ANN4FLES package [16, p.161], which is a neural network package developed in C++ for applications in the FLES package. The ANN4FLES package is developed within the group of Prof. Kisel at Frankfurt Institute for Advanced Studies (FIAS). This group consists of a team of students and PhD students, supervised by Prof. Kisel, including Gianna Zischka. The package offers the flexibility to construct a wide variety of neural network architectures. It allows selecting from different types of networks, layer counts, neuron quantities, activation and optimization algorithms, as well as loss functions, among other options.

5.1 Adjusting Specific Cuts for the Network Approach

In its reconstruction process, the `KFParticleFinder` class employs various preset cut-off values, commonly referred to as "cuts." These statistical cuts serve as threshold parameters for particle selection, including values like χ^2_{prim}, which denotes the χ^2 relationship of the particle to the primary vertex.

In a χ^2-test, observed data points are evaluated against a predefined theoretical

framework or hypothesis. When the calculated statistic for a specific value exceeds a set threshold, it becomes improbable that the value aligns with the given model or hypothesis.

In this work, the focus is on three χ^2-cuts that are applied in CBM: χ^2_{geo}, χ^2_{prim} and χ^2_{topo}. The first one is used to create mother particle candidates. Whenever an intersection of particle tracks is found that could indicate a possible secondary vertex, χ^2_{geo} is used to test if it is likely that the given tracks build a decay vertex. In other words, it is the fit of a daughter track to another daughter track. The second cut, χ^2_{prim}, is used to determine if a track is pointing to the primary vertex, the primary collision point. This test helps to identify particles that come directly from the collision and are not a result of decayed particles that decayed after flying away from the vertex. The last cut, $\chi^2_{topo} = \chi^2_{geo} + \chi^2_{prim}$ is testing the whole particle topology. Thus, if the particle with its reconstruction is likely to be real. Additionally, the $l/\Delta l$ cut is focused in this work, since the normalized decay length can help to identify Λ.

For the presented approach, three cuts that are usually used to decide whether a particle is a Λ or not were loosened. The default cuts from the CBM experiment are shown in Table 5.1.

Cut value	Cut parameter	Cut description
$\chi^2_{prim} > 18.42$	`fCuts2D[0]`	χ^2 of the extrapolated track to the primary vertex
$\chi^2_{geo} < 3$	`fCuts2D[1]`	χ^2 of the track to the second daughter track
$l/\Delta l > 3$	`fCuts2D[2]`	decay length normalized on the error

Table 5.1: Default cut values of the KF Particle Finder in CBM

The looser cuts provide space for the neural network to operate more freely. If the cuts are too tight, it is possible that some Λ particles are already sorted out by the KF Particle Finder cuts before the neural network is able to classify the particles. The looser cuts can be found in Table 5.2. In case of the CBM experiment, the χ^2_{prim} cut threshold is overwritten using an external function. For this purpose, the function `SetPrimaryProbCut` is used, which is located in the class `CBMKFParticleFinder`. The function uses its own calculations to obtain a threshold for the χ^2_{prim} cut specifically for the CBM experiment. The parameter for this function was set from 0.0001 to 0.1 for this work, such that the cut was changed from 18.42 to 4.605.

Cut value	Cut parameter	Cut description
$\chi^2_{prim} > 4.605$	`fCuts2D[0]`	χ^2 of the extrapolated track to the primary vertex
$\chi^2_{geo} < 6$	`fCuts2D[1]`	χ^2 of the track to the second daughter track
$l/\Delta l > 2$	`fCuts2D[2]`	decay length normalized on the error

Table 5.2: Looser cut values of the KF Particle Finder in CBM

The parameters of the Λ particles which were compared with the cuts as well as other parameters of the particles were extracted and used as input parameters for the

network. Table 5.3 shows all the parameters used, which were then applied to train the network with varying configurations. The selected parameters are chosen, since they are usually used to identify the particles by rejecting falsely reconstructed ones. Furthermore, similar work [65] has already shown that machine learning approaches based on χ^2_{geo}, χ^2_{topo} and $l/\Delta l$, including the parameters of daughter particles, are promising. Since Λ is reconstructed based on its daughter particles, these contain cut information about the quality of the reconstructed Λ. Additionally, the mass and transverse momentum are also considered as important properties of a particle, why the decision was made to include these parameters in this work as well.

5.2 Extraction of the Training Data

A struct containing the necessary parameters was set up for data extraction in the class `KFParticle`. This allows for the immediate storage and access of values in the respective particles, all of which are kept in the `fParticle` array of the `KFParticleTopoReconstructor` class. In the case of this thesis, it was done for the particle Λ. However, it would be also possible to use the same way for the storage of parameters needed for other particles. Thus, this approach of extracting needed data for the neural network to improve the signal/background ratio would also be applicable to other particles. The parameters generated can be seen in Table 5.3 including the functions `GetMass()` and `GetPt()` which were utilized to obtain the mass and transverse momentum of Λ, due to the ease of information access.

Parameter	Parameter Description
`is_signal_ann`	Boolean if the neural network classified the reconstructed Λ as signal or background
`is_signal_mc`	Boolean if the reconstructed Λ is a real Λ coming from the MC simulation or not
`chi2geo`	χ^2_{geo} of the reconstructed Λ
`chi2prim`	χ^2_{prim} of the reconstructed Λ
`chi2topo`	χ^2_{topo} of the reconstructed Λ
`ldl`	$l/\Delta l$ of the reconstructed Λ
`daughterPtPos`	p_T of the positive daughter of the reconstructed Λ
`daughterPtNeg`	p_T of the negative daughter of the reconstructed Λ
`daughterChi2PrimPos`	χ^2_{prim} of the positive daughter of the reconstructed Λ
`daughterChi2PrimNeg`	χ^2_{prim} of the negative daughter of the reconstructed Λ
`GetMass()`	function of the `KFParticle` class which returns the mass. Used to get the mass of the reconstructed Λ.
`GetPt()`	function of the `KFParticle` class which returns the transverse momentum. Used to get the transverse momentum of the reconstructed Λ.

Table 5.3: Parameter which were stored for the neural network approach. The last 10 were used as an input for the neural network.

Most of the required values are stored within the `KFParticleFinder` class. Within this class, the `FindParticles` method is invoked to perform the reconstruction of short-lived particles. This method internally calls `Find2DaughterDecay` for reconstructing 2-daughter channels. This method is used in this approach to store the χ^2_{prim} values of the daughters. Within `Find2DaughterDecay` the method `ConstructV0` is called. It combines two particle candidates into a 2-daughter mother candidate, applying the necessary cuts and is used to store the remaining relevant parameters for this task.

To be able to store the χ^2_{prim} values of the daughters, two SIMD vectors are initialized in the `KFParticleSIMD` class: `cutPosDaughters`, for χ^2_{prim} of the positive daughter and `cutNegDaughters` for χ^2_{prim} of the negative daughter. Inside the method `Find2DaughterDecay` the χ^2_{prim} values are stored in the vectors, which can be seen in Listing 5.1. In this code section, the loop over the SIMD elements is shown, which recombines the `KFParticleSIMD` elements. The loop is used to reject created particles that do not fulfill several criteria for particle creation. Particles rejected are flagged as not active and will therefore be skipped (line 2). In general, information about non-rejected particles is stored in buffer SIMD vectors that will be further processed in the `ConstructV0` function if the buffer is full. Thus, further calculations on SIMD objects are possible, but some particles are already removed to reduce the number of particles that need further processing. In this loop, the own SIMD vectors are filled with the χ^2_{prim} values of the daughters (line 5-6), ensuring that they can be used when further processed in `ConstructV0` and later stages. Otherwise, this information would have been lost.

```
1     for (int iV = 0; iV < float_vLen; iV++) {
2       if (!(active[iPDGPos][iV])) continue;
3
4       // included code
5       mother.cutPosDaughters[nBufEntry] = chi2PrimPos[iV];
6       mother.cutNegDaughters[nBufEntry] = chi2PrimNeg[iV];
7       // end of the included code
8
9       idPosDaughters[nBufEntry] = iTrP + iV;
10      idNegDaughters[nBufEntry] = negInd[iV];
11
12      daughterPosPDG[nBufEntry] = trackPdgPos[iPDGPos][iV];
13      daughterNegPDG[nBufEntry] = trackPdgNeg[iV];
14
```

Listing 5.1: Storing of daughter χ^2_{prim} values in the `KFParticleSIMD` object of the mother particle

In `ConstructV0`, objects from the `KFParticleSIMD` class are utilized, each containing SIMD vectors for attributes like mass and transverse momentum. Therefore, the required information for Λ must be written into SIMD vectors, which are initially filled with zeros. These vectors are: `chi2geo_values`, `chi2prim_values`, `chi2topo_values`, `ldl_values`. This is carried out to enable the storage of values within the `KFParticle` object's struct at a later stage.

First, the χ^2_{geo} values are extracted. A unique aspect of the χ^2 values is that they are overwritten multiple times throughout the code. Therefore, it is important to ensure that they are extracted and stored at the correct location. The χ^2_{geo} value for the `KFParticleSIMD` object `mother` is calculated using the `Construct` method, which constructs a `KFParticleSIMD` object of short-lived particles from a set of daughters. This means that all parameters of `mother` are initialized using this method. Therefore, `chi2geo_values` is filled directly after this function call, which can be seen in Listing 5.2. Since the χ^2 value of an `KFParticleSIMD` object is stored normalized with the number of degrees of freedom (NDF), it must be divided by NDF.

```
1     // chi2 is calculated by this method
2     mother.Construct(vDaughtersPointer, 2, 0);
3
4     // Included code line: Saving the values of chi2geo for later storage in
    ↪  the structure of the KFParticle object.
5     chi2geo_values = mother.GetChi2() / simd_cast<float_v>(mother.GetNDF());
6
7     // code of the method ConstructV0 where the cut chi2geo is used
8     float_m saveParticle(simd_cast<float_m>(int_v::IndexesFromZero() < int(
    ↪ NTracks)));
9     float_v chi2Cut = cuts[1]; // chi2geo
10    float_v ldlCut = cuts[2];
11    if (!(simd_cast<float_m>(abs(mother.PDG()) == 421 || abs(mother.PDG())
    ↪ == 426 || abs(mother.PDG()) == 420)).isEmpty()) {
12      chi2Cut(simd_cast<float_m>(abs(mother.PDG()) == 421 || abs(mother.PDG()
    ↪ ) == 426 || abs(mother.PDG()) == 420)) = fCutsCharm[0];
13      ldlCut(simd_cast<float_m>(abs(mother.PDG()) == 421 || abs(mother.PDG())
    ↪  == 426 || abs(mother.PDG()) == 420)) = -1; //fCutsCharm[1];
14    }
15
16    // comparison between the values of chi2geo and the cut
17    saveParticle &= (mother.Chi2() / simd_cast<float_v>(mother.NDF()) <
    ↪ chi2Cut);
18    saveParticle &= KFPMath::Finite(mother.GetChi2());
19    saveParticle &= (mother.GetChi2() > 0.0f);
20    saveParticle &= (mother.GetChi2() == mother.GetChi2());
21
```

Listing 5.2: Code section of saving the χ^2_{geo} values in the method `ConstructV0` in the class `KFParticleFinder`

The code section where the values of `ldl_values`, `chi2topo_values` and `chi2prim_values` are filled can be seen in Listing 5.3 and is located further down in the `ConstructV0` method after `chi2geo_values` has been filled. In line 6, the χ^2 value is overwritten with the value of χ^2_{topo} using the method `SetProductionVertex`. In this function call, χ^2_{prim} is calculated and added to the χ^2 class member variable, so that $\chi^2_{topo} = \chi^2_{geo} + \chi^2_{prim}$ is finally stored as the χ^2 value of the class object. The decay length is given by using the function `GetDecayLength`, which determines the decay length of the particle and its error in the laboratory system and gets both as an output parameter. Since `ldlMin` is used for the comparison with the $l/\Delta l$ cut, `ldlMin` is used to fill the SIMD vector `ldl_values`. Since both values, χ^2_{topo} and χ^2_{prim}, are already stored, it is possible to calculate χ^2_{prim} by $\chi^2_{prim} = \chi^2_{topo} - \chi^2_{geo}$ which is done

in line 15.

```
1   #ifdef NonhomogeneousField
2     KFParticleSIMD motherTopo;
3     ldlMin = 1.e8f;
4     for (int iP = 0; iP < fNPV; iP++) {
5       motherTopo = mother;
6       motherTopo.SetProductionVertex(PrimVtx[iP]);
7       motherTopo.GetDecayLength(l[iP], dl[iP]);
8       float_v ldl = (l[iP] / dl[iP]);
9       ldlMin((ldl < ldlMin) && saveParticle) = ldl;
10    }
11
12    // included code
13    ldl_values = ldlMin; //set ldl_values
14    chi2topo_values = motherTopo.GetChi2() / simd_cast<float_v>(motherTopo.
      ↪ GetNDF()); // set chi2topo_values
15    chi2prim_values = chi2topo_values - chi2geo_values; // set
      ↪ chi2prim_values
16    // end of the included code
17  #endif
18
19
20    saveParticle &= ((float_m(!isPrimary) && ldlMin > ldlCut) || float_m(
      ↪ isPrimary)); // ldl cut is used here
21
```

Listing 5.3: Code Section of saving the $l/\Delta l$, χ^2_{topo} and χ^2_{prim} values in the method `ConstructV0` in the class `KFParticleFinder`

Since `mother` is a `KFParticleSIMD` object, iteration is performed through the corresponding SIMD vector, using `NTracks` (the number of SIMD vector elements) as the loop limit. If the particle has not fulfilled the cut results, then `saveParticle[iv]` is set to false and the new iteration begins. The `saveParticle` serves as a mask that is logically ANDed with the cuts to determine whether the particle should be reconstructed. Using the `GetKFParticle` function, the element of the index `iv` will be copied to the KFParticle object mother_temp. The Array `Particles` in the line after the call of `GetKFPaticle` includes all reconstructed particles. The ID of each particle is the index of the corresponding particle inside the array `Particles`. Since the particles are written into the array `Particles` further down within the for-loop, the size of the array is also the ID of the particle. This can be seen in Listing 5.4.

```
1   for (int iv = 0; iv < NTracks; iv++) {
2     // NTracks = Number of SIMD vector elements
3     if (!saveParticle[iv])
4     continue;
5
6     mother.GetKFParticle(mother_temp, iv);
7     int motherId = Particles.size();
8     mother_temp.SetId(Particles.size());
9
```

Listing 5.4: Extraction of `KFParticle` object `mother_temp` from `KFParticleSIMD` object `mother`

Further down, before the array `Particles` is filled with the new particle, the information for the struct is set, which can be seen in Listing 5.5. First the PDG code is checked, if the current object is a particle object of Λ, as this is the interested particle for this task. Since `mother_temp` is a `KFParticle` object and no `KFParticleSIMD` object the element with the index `iv` of the SIMD vectors `chi2geo_vlaues`, `chi2prim_values`, `chi2topo_values` and `ldl_values` as well as of the arrays `cutPosDaughters` and `cutNegDaughters` of the `mother` object, is saved in the corresponding struct parameter. For the transverse momentum of the daughters, the daughter's `KFParticle` object has to be created first. Since the daughter IDs are saved inside each `KFParticle` object, the ID is extracted from `mother_temp`. With this, the daughters can be extracted from the `Particles` array using the ID and saved as an `KFParticle` object. This is done for the positive daughter as well as for the negative daughter, seen in line 13 and 16. Applying the function `GetPt` to the corresponding object the transverse momentum of the daughter is extracted and saved in the struct of `mother_temp`, seen in line 14 and 17.

```
1     // included code
2     // if reconstructed lambda
3     if (mother_temp.GetPDG() == 3122) {
4
5       mother_temp.data.chi2geo = chi2geo_values[iv];
6       mother_temp.data.chi2prim = chi2prim_values[iv]; // chi2topo - chi2geo
    ↪  = chi2prim
7       mother_temp.data.chi2topo = chi2topo_values[iv];
8       mother_temp.data.ldl = ldl_values[iv];
9       mother_temp.data.daughterChi2PrimPos = mother.cutPosDaughters[iv];
10      mother_temp.data.daughterChi2PrimNeg = mother.cutNegDaughters[iv];
11
12      // extract daughter information
13      KFParticle &daughParticle0 = Particles[mother_temp.DaughterIds()[0]];
14      mother_temp.data.daughterPtNeg = daughParticle0.GetPt();
15
16      KFParticle &daughParticle1 = Particles[mother_temp.DaughterIds()[1]];
17      mother_temp.data.daughterPtPos = daughParticle1.GetPt();
18
```

Listing 5.5: Included code inside the for-loop of Listing 5.4 where the information of the struct is set

After this, the parameters of the struct `chi2geo, chi2prim, chi2topo, ldl, daughterChi2PrimPos` and `daughterChi2PrimNeg` are limited to the interval $[-10,000, +10,000]$. Subsequently, they are normalized by 10,000, since large fluctuations can have a severe impact on the learning performance of the neural network, especially when Softmax activation function is applied in the output layer. Softmax activation takes the weighted sum as an input and uses the exponential function to proceed with these values. If the exponents are exploding due to the extreme values, the model is numerically instable. To avoid this problem, a limit of $\pm 10,000$ is set and values are then normalized between -1 and +1.

The input of the neural network in the KF Particle Finder package, which is also included in the method `ConstructV0` is set directly after the parameters have been

normalized. All parameters are stored in a float-vector `inputs_ann`. The mass and transverse momentum parameters are directly obtained from the particle through the `GetMass` and `GetPt` methods. These methods return the mass and transverse momentum, respectively, if both, the value and its error, are well-defined.

For training, the data is extracted at the end of the method `MatchParticles` of the class `KFTopoPerformance`. This method matches the reconstructed particles with the created particles of the MC simulation and stores this bidirectional connection by storing the IDs into arrays. Inside the function first, the matches are found. If a match is found, the type of the reconstructed particle and the matched MC particle is checked. Λ is treated as a normal decay within the `MatchParticles` method. Within this code section, the reconstructed Λ matches either with a Monte-Carlo generated Λ (signal) or with another MC generated particle (e.g. K_s) (physical background). In the first case, for the parameter `is_signal_mc` within the struct a 1 is assigned for true, in the second a 0 for false, which is seen in Listing 5.6. In the case that no match was found (ghosts/combinatorial background) the default value of the parameter remains, which is set to -1.

```
1       // rPart: reconstructed particle, mmPart: Monte-Carlo particle
2       // if PDG-Code and number of daughters of both is the same
3       if (mmPart.GetPDG() == rPart.GetPDG() && mmPart.NDaughters() == rPart.
    ↪ NDaughters()) {
4         MCtoRParticleId[mmId].ids.push_back(iRP);
5         RtoMCParticleId[iRP].ids.push_back(mmId);
6
7         // included code line
8         // Assignment of the particle to signal
9         const_cast<KFParticle&>(rPart).data.is_signal_mc = 1;
10      } else {
11        MCtoRParticleId[mmId].idsMI.push_back(iRP);
12        RtoMCParticleId[iRP].idsMI.push_back(mmId);
13
14        // included code line
15        // Assignment of the particle to background
16        const_cast<KFParticle&>(rPart).data.is_signal_mc = 0;
17      }
18
```

Listing 5.6: Assignment of the matched particles to signal or physical background. In the case of signal the struct `is_signal_mc` is set to 1, in the case of physical background to 0. If the particles weren't matched before (combinatorial background) `is_signal_mc` retains the default value which is set to -1.

Directly after the tagging of signal and physical background particles, the data set is extracted at the end of `MatchParticles` as it can be seen in Listing 5.7. The mass and transverse momentum parameters are also directly obtained from the reconstructed particle through the `GetMass` and `GetPt` methods.

After the data set was extracted, a Python script was used to create a balanced dataset that consists of 50% signal and 50% background particles.

```
1       // included code
2       std::ofstream file_input_ANN4FLES;
3       file_input_ANN4FLES.open("ANN4FLES_input.txt", std::ios::app);
4
5
6       for (const KFParticle part : fTopoReconstructor->GetParticles()) {
7         if (part.GetPDG() =/= 3122) continue; // incl. ghosts (is_signal_mc =
    ↪  -1)
8
9         if (part.data.is_signal_mc == -1)
10          const_cast<KFParticle&>(part).data.is_signal_mc = 0; // for the
    ↪ neural network, ghosts are just concidered as background)
11
12          file_input_ANN4FLES << part.data.is_signal_mc
13          << ";" << part.GetMass()
14          << ";" << part.GetPt()
15          << ";" << part.data.chi2geo
16          << ";" << part.data.chi2prim
17          << ";" << part.data.chi2topo
18          << ";" << part.data.ldl
19          << ";" << part.data.daughterPtPos
20          << ";" << part.data.daughterPtNeg
21          << ";" << part.data.daughterChi2PrimPos
22          << ";" << part.data.daughterChi2PrimNeg
23          << std::endl;
24      }
25
26      file_input_ANN4FLES.close();
27
```

Listing 5.7: Extracting the data for training and validation of the network in the method `MatchParticles` of the class `KFTopoPerformance`

5.3 Neural Network Model

The architecture created with ANN4FLES features an input layer, four hidden layers, and an output layer. Since a neural network with more than two hidden layers is usually considered deep, the presented approach can be considered as deep learning. The input layer has 10 neurons, corresponding to the number of parameters. Each of the four hidden layers contains 64 neurons and utilizes Leaky Rectified Linear Unit (Leaky ReLU) as the activation function. The output layer comprises 2 neurons and employs Softmax for activation. As the loss function, binary cross-entropy loss is used and Adam was chosen as weight optimizer. The learning rate α is set to 0.001, β_1 is set to 0.9, β_2 to 0.999, and the ϵ value to 10^{-8}.

The definition for Leaky ReLU is as follows:

$$LeakyReLU := f(a_i^{(k)}) = \left\{ \begin{array}{ll} s \cdot a_i^{(k)}, & if\ a_i^{(k)} \leq 0 \\ a_i^{(k)}, & if\ a_i^{(k)} > 0 \end{array} \right\} \tag{5.1}$$

with $a_i^{(k)} := input\ of\ neuron\ i\ of\ layer\ k$ and negative slope $s \in \mathbb{R}$.

The parameter s is a constant that is typically set to a small value. In this case, s was set to 0.1. The problem of differentiability exists at $a_i^{(k)} = 0$, thus the function is not differentiable at all points. Since in the case of $a_i^{(k)} = 0$, the neuron has no influence on the subsequent layers, one solution would be to set the gradient manually to 0 and consequently not to carry out a weight update for this neuron. It is therefore possible to find a suitable gradient for this special case despite the lack of differentiability at $a_i^{(k)} = 0$ by setting it to 0. In Figure 5.1, Leaky ReLU is seen with s set to 0.1.

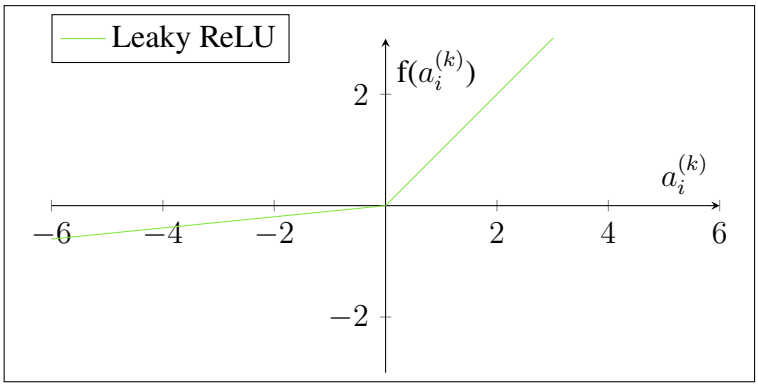


Figure 5.1: Leaky ReLU function with $s = 0.1$ and $a_i^{(k)}$ as the input of neuron i of layer k and $f(a_i^{(k)})$ the output of Leaky ReLU

In the output layer Softmax is used, since Softmax can be interpreted as a probability distribution of the predicted classes, in this case *signal* and *background*. Softmax is applied to the whole layer, since all input values of each neuron are needed. Softmax is defined as follows:

$$Softmax := f(a_i^{(k)}) = \frac{\exp(a_i^{(k)})}{\sum_{j=0}^{n-1} \exp(a_j^{(k)})} \tag{5.2}$$

with $a_i^{(k)} := input\ of\ neuron\ i\ of\ the\ current\ layer\ k$ and
$\sum_{j=0}^{n-1} \exp(a_j^{(k)})$ as the sum from $j = 0$ to $n - 1$ of the exponential of neuron inputs $a_j^{(k)}$ of layer k. n is the number of the neurons of layer k.

In the neural network being used, n is equal to 2, as two output neurons are used. The corresponding graph of Softmax can be seen in Figure 5.2.

For the training and validation of the neural network, a dataset of 787,000 Λ particles was used. The dataset consists of 50:50 signal and background (physical and combinatorial) Λ particles. It was then split by 80:20 ratio for training and validation respectively. Thus, the network has been trained with 630,000 Λ particles and validated by using 157,000 particles. The training process was repeated over 30 epochs, using a batch size of 100. With the given settings, the network achieved a raw classification performance of almost 99% accuracy on the validation set, which can be seen in Figure 5.3.

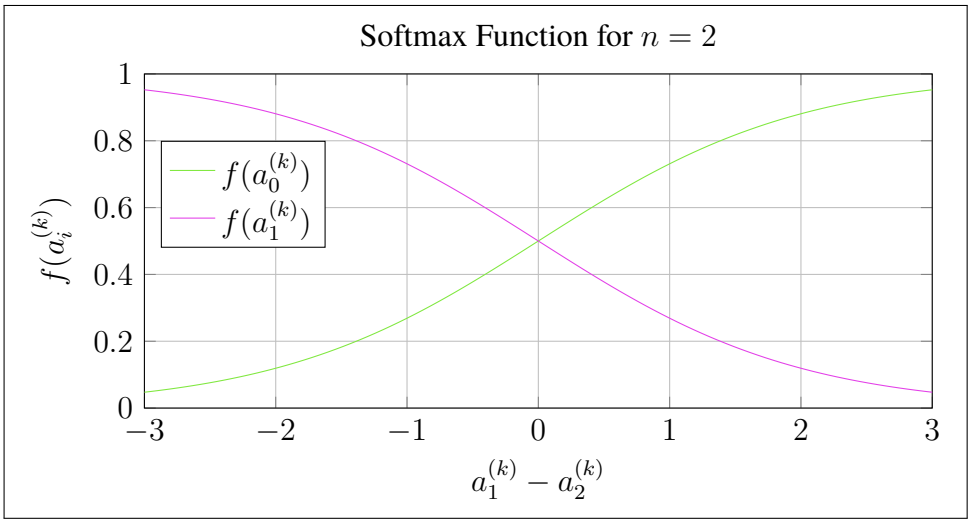


Figure 5.2: Softmax function for $n = 2$, $a_i^{(k)}$, $i \in \{0, 1\}$ as the input of neuron i of layer k and $f(a_i^{(k)})$ as the output of Softmax

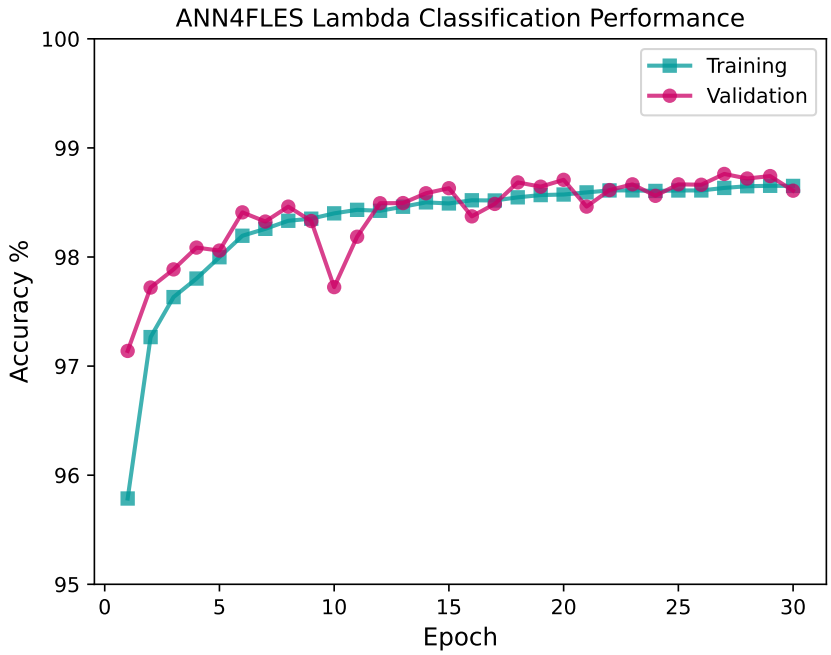


Figure 5.3: Accuracy values of training and validation, using ANN4FLES for Λ signal/background classification. The accuracy values reach up to almost 99% and the training curve is stable.

5.4 Implementation of ANN4FLES in the KF Particle Finder Package

Within the constructor of the `KFParticleFinder` class, the neural network is initialized. The network topology of 10 input neurons, 4 hidden layers with 64 neurons each and 2 output neurons is set and the weight file with the pre-trained network weights is loaded. This can be seen in Listing 5.8.

```
1       // vector with the layer sizes
2       std::vector<int> topology = {10, 64, 64, 64, 64, 2};
3       net = new Network();
4       net->InitializeANN(topology, true);
5       // pre-trained weights
6       std::string weight_path = "weights_ann4fles_sb.nnw";
7       net->ImportNeuronWeights(weight_path);
8
```

Listing 5.8: Initialization of ANN4FLES in the constructor of `KFParticleFinder`

The neural network is included in the `ConstructV0` method within the `KFParticleFinder` class. It is located directly after the input parameters for the neural network has been set (see also Listing 5.9). The `InferOutput` function of ANN4FLES takes the current input vector as an input parameter and then applies the feed-forward through the network. Since Softmax is used in the output layer, a probability distribution is the result. However, `InferOutput` does not return this probability, but the Arguments of the Maxima (Arg Max) of the neuron outputs and, thus, it returns the predicted class. In this case, it returns 1 if the particle is classified as signal, 0 otherwise. For later analysis, this result is stored in the struct parameter `is_signal_ann` of the particle.

```
1         // included code
2         // set input values for ANN
3         std::vector<float> inputs_ann = { mother_temp.GetMass(),
4           mother_temp.GetPt(),
5           mother_temp.data.chi2geo,
6           mother_temp.data.chi2prim
7           mother_temp.data.chi2topo,
8           mother_temp.data.ldl,
9           mother_temp.data.daughterPtPos,
10          mother_temp.data.daughterPtNeg,
11          mother_temp.data.daughterChi2PrimPos,
12          mother_temp.data.daughterChi2PrimNeg };
13
14        // infer output from neural network and set result into the struct
15        int out = net->InferOutput(inputs_ann);
16        mother_temp.data.is_signal_ann = bool(out);
17      }
18      // end of included code
19
20      Particles.push_back(mother_temp);
21
```

Listing 5.9: The insertion of the neural network in the `ConstructV0` method to classify the Λ particles.

5.5 Results

For the test phase of the deep neural network, ANN4FLES is implemented in the KF Particle Finder package and compared to the results of the default approach of the CBM experiment. Here, the performance measurement tools of the KF Particle Finder are used to evaluate the results. The number of simulated events used for testing is 46,000. The same data set is used for the default approach and the neural network based approach to compare both results.

Figure 5.4 displays the entire set of reconstructed Λ particles within the mass from 1.06 GeV/c^2 to 1.2 GeV/c^2. The distribution generated by using the KF Particle Finder package with the looser cuts in combination with ANN4FLES is shown in red. In blue, the distribution with the default settings is displayed. The peak of Λ is around the expected mass of 1.115 GeV/c^2, which corresponds to the PDG mass of Λ. Entries distant from this value are highly likely to be physical or combinatorial background. The total amount of particles shown in the histogram are 818,055 for the default approach and 207,215 for the network approach. So, in general, the amount of reconstructed Λ particles has been reduced by a factor of 3.94. It can be seen that the neural network approach has a clearer peak in the region of the PDG mass of Λ, which indicates that the background has been well reduced. On the other hand it is also shown, that the peak has fewer entries, therefore the number of correctly reconstructed Λ particles must be studied more closely.

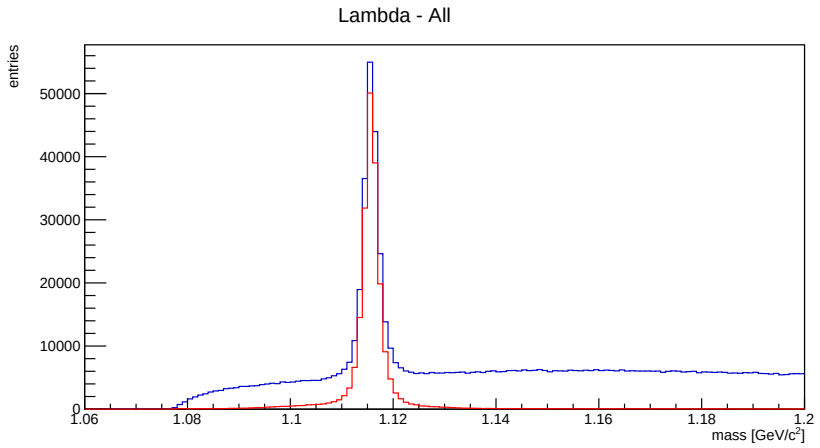


Figure 5.4: Histogram of all reconstructed Λ particles consisting of signal, physical background and ghosts in the mass range from 1.06 GeV/c^2 to 1.2 GeV/c^2. It presents a comparison between Λ particles reconstructed using the default settings of the KF Particle Finder package (blue) and those reconstructed using the looser cuts within the KF Particle Finder package in combination with ANN4FLES (red).

In Figure 5.5, however, it can be seen that in the region of the peak, the number of correctly reconstructed Λ particles did not decrease significantly. The total amount of true positives are 182,364 correct reconstructed Λ particles for the default approach and 179,891 for the neural network approach in the mass range from 1.1 GeV/c^2 to 1.135 GeV/c^2. Hence, the approach with looser cuts in combination with ANN4FLES reconstructs a negligible factor of 1.36% less true Λ particles.

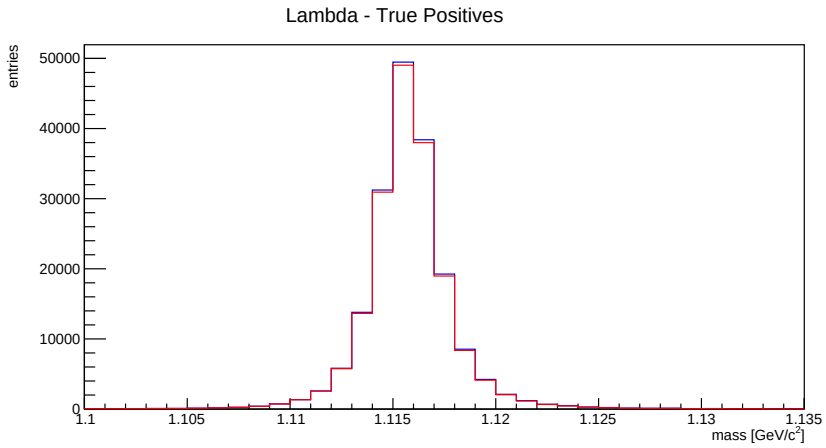


Figure 5.5: Histogram of the true positives of all reconstructed Λ particles in the mass range from 1.1 GeV/c^2 to 1.135 GeV/c^2 using the default settings of KF Particle Finder package (blue) in comparison to those reconstructed using the looser cuts within the KF Particle Finder package in combination with ANN4FLES (red).

The previous assumption that the background has been reduced by using the deep neural network in combination with loose cuts is confirmed by a look at Figure 5.6. Here, a reduction of the physical background is clearly visible. The peak around 1.08 GeV/c^2 is reduced significantly. Moreover, the background in the region from 1.14 GeV/c^2 upwards was almost completely removed. Around the PDG mass of Λ, the neural network approach also had difficulties in reducing the physical background, but a reduction compared to the default approach is also visible. In total, in the range from 1.06 GeV/c^2 to 1.2 GeV/c^2, the physical background of the default approach consists of 34,746 falsely classified Λ particles. In case of the approach with ANN4FLES the absolute number of falsely classified Λ particles is 8,314, a factor of 4.18 less physical background.

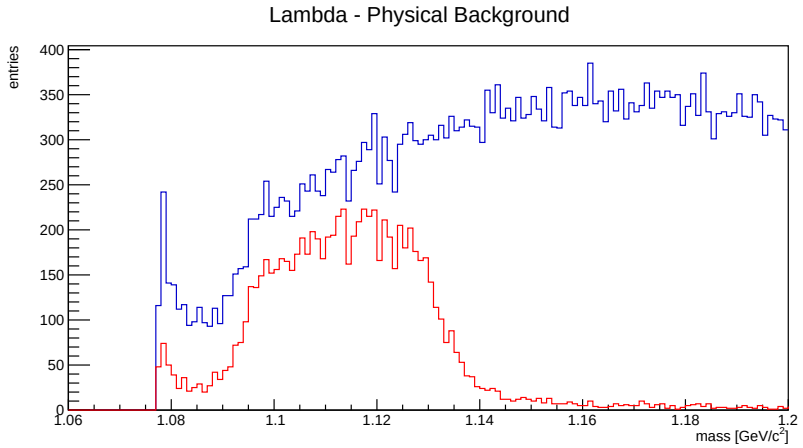


Figure 5.6: Histogram of the physical background of all reconstructed Λ particles in the mass range from 1.06 GeV/c^2 to 1.2 GeV/c^2 using the default settings of KF Particle Finder package (blue) in comparison to those reconstructed using the looser cuts within the KF Particle Finder package in combination with ANN4FLES (red).

A highly significant reduction of the background can be seen in the combinatorial background (ghosts) (Figure 5.7). A reduction of this is of particular interest. Unlike physical background, which comprises genuine particles, ghosts are irrelevant for physical analysis as they represent non-existent, reconstructed particles. Around the PDG mass of Λ a small increase is still seen, but in general the amount of ghosts is reduced drastically with a factor of 34.33. Overall, the histogram shows 599,025 ghost Λ particles for the default approach and only 17,447 for the neural network based approach.

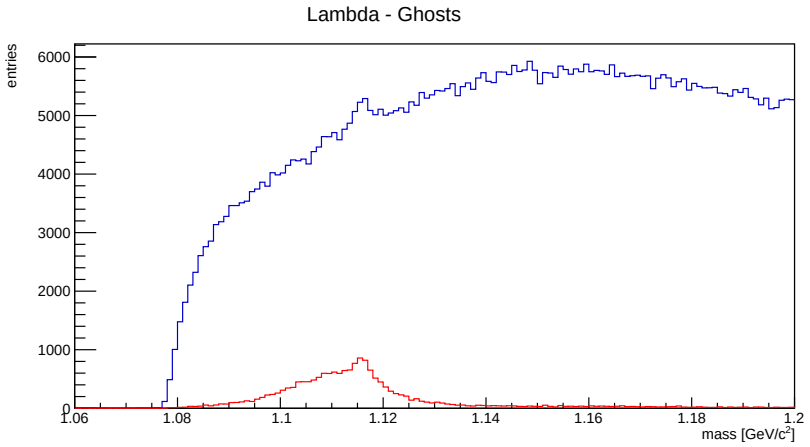


Figure 5.7: Histogram of the combinatorial background (ghost) of all reconstructed Λ particles in the mass range from 1.06 GeV/c^2 to 1.2 GeV/c^2. Comparison between the reconstructed Λ particles using the default settings of KF Particle Finder package (blue) and the reconstructed ones using the ANN4FLES approach (red).

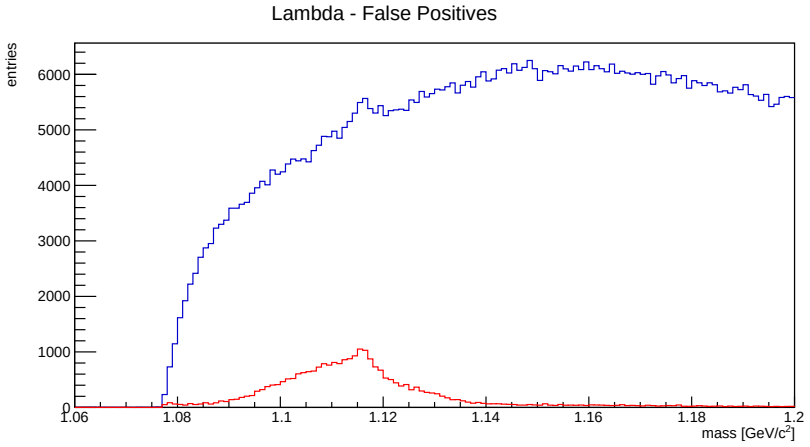


Figure 5.8: Histogram of the false positives of the reconstructed Λ particles in the mass range of 1.06 GeV/c^2 to 1.2 GeV/c^2 using the default settings of the KF Particle Finder package (blue) in comparison to the ANN4FLES approach (red).

In summary, the background in general is reduced by a huge amount, which is seen in Figure 5.8. This histogram shows the total amount of falsely reconstructed Λ particles (combinatorial + physical background) in the mass range of 1.06 GeV/c^2 to 1.2 GeV/c^2. Here, the default approach reconstructs 633,771 Λ particles incorrectly,

the neural network approach to differ in that only reconstructs 25,761 Λ particles that are physical or combinatorial background. In comparison to the default cuts, the neural network rejects a factor of 24.6 more background. Hence, the approach with looser cuts in combination with ANN4FLES produces 95.94% less total background compared to the default approach.

With regard to the neural network approach, it can be seen that the number of false negative Λ particles, which means all Λ particles which has been produced by the MC simulation but have been classified incorrectly as background by the neural network, is with a total amount of 2,758 particles not that high (Figure 5.9). In comparison, the number of true negative Λ particles is significantly greater with a total amount of 2,006,910 particles (Figure 5.10).

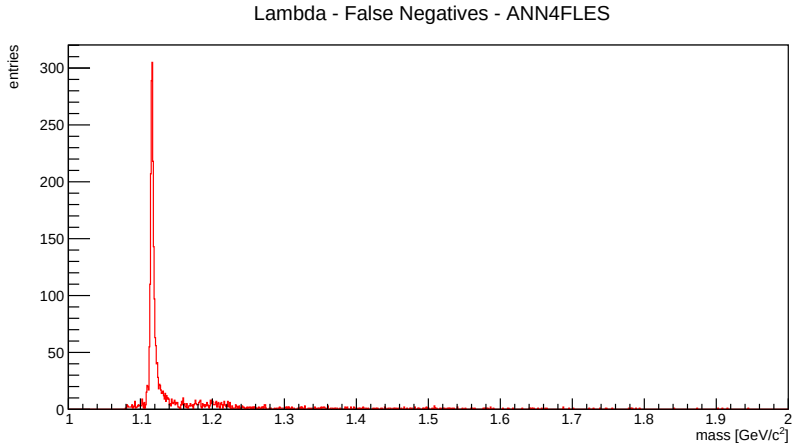


Figure 5.9: Histogram of false negatives of all reconstructed Λ particles in the range of 1 GeV/c^2 to 2 GeV/c^2, using the looser cuts within the KF Particle Finder package in combination with ANN4FLES.

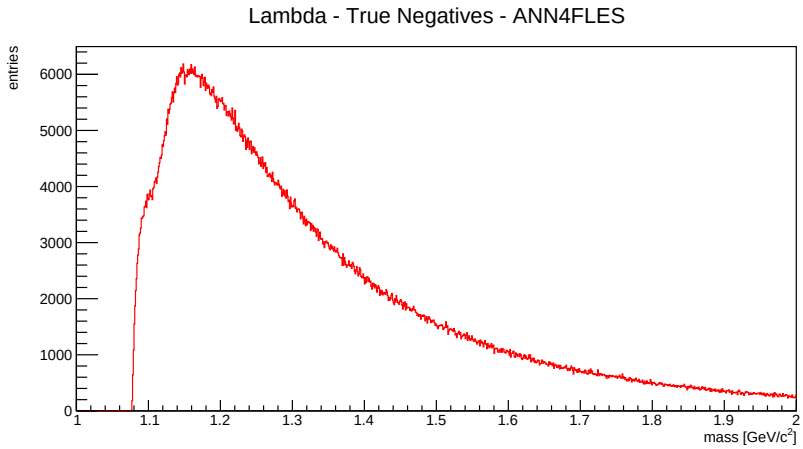


Figure 5.10: Histogram of the true negatives of all reconstructed Λ particles in the range of 1 GeV/c^2 to 2 GeV/c^2 using the looser cuts within the KF Particle Finder package in combination with ANN4FLES.

The performance in comparison to MC information shows that the neural network based approach is not rejecting too much more signal than stronger cuts, and furthermore, has a huge background rejection potential. While this, in theory, is a good result from a raw performance perspective, there are no MC information in a real

experiment and the particles reconstructed can not be compared. Thus, statistical and mathematical methods are used that are only based on the reconstructed Λ particles. This includes the signal/background ratio of the reconstructed particles as well as the respective significance.

The signal/background ratio S/B is a metric that describes the proportion of signal to background. The significance $S/\sqrt{S+B}$ is a metric that measures the clarity of a signal peak in comparison to fluctuations of the background. Here, a threshold of 5 is considered as a significance value that shows a peak that is unlikely to be a background fluctuation. Both metrics are calculated to evaluate the results from physics perspective.

In Figure 5.11, the total amount of reconstructed Λ particles in the mass range of 1.06 GeV/c^2 to 1.2 GeV/c^2 is fitted by the sum of a Gaussian function (signal) and a 2nd order polynomial (combinatorial and physical background). The fitted function represents approximately the mass distribution of Λ. Based on the fit, σ is extracted from the Gaussian fit. Afterwards, the signal can be calculated by the integral in the range of $\pm 3\sigma$ of the Gaussian function that represents the signal, whereas the background is calculated using the integral of the 2nd order polynomial in the range of $\pm 6\sigma$ around the mean of 1.115 GeV/c^2. The resulting signal/background ratio calculated by these integrals is 3.49. Thus, for the approximation based on the function fits, there is almost 3.5 times more signal than background represented in the data. The significance for this case is 11.38.

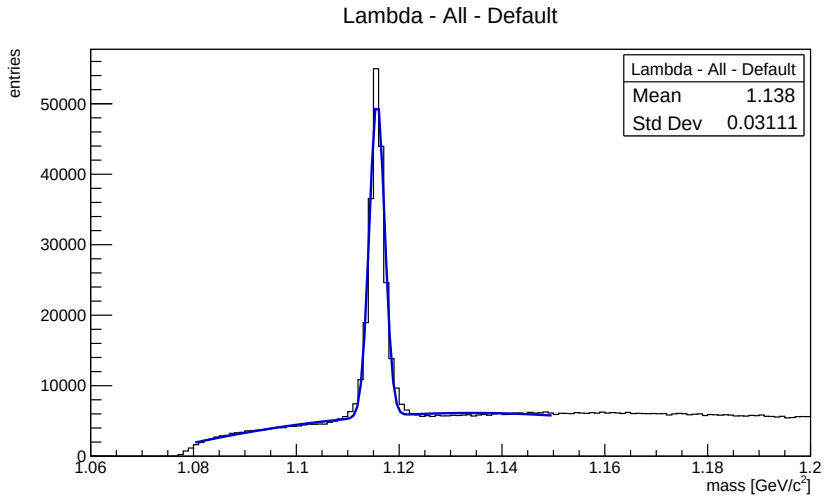


Figure 5.11: Fitted histogram of all reconstructed Λ particles using the default approach in the mass range of 1.06 GeV/c^2 to 1.2 GeV/c^2. The black function shows the unfitted distribution of the default approach, the blue one the fitted distribution.

In Figure 5.12, the same functions for fitting are applied to the Λ particles classified as signal. Thus, to the Λ particles that count as correctly reconstructed if the neural network makes the decision without MC information. Again, the σ is extracted to calculate the $\pm 3\sigma$ and $\pm 6\sigma$ interval boundaries. Using the integrals of these regions, the resulting signal/background ratio is 38.28. The significance for

this approach has also increased to 12.95.

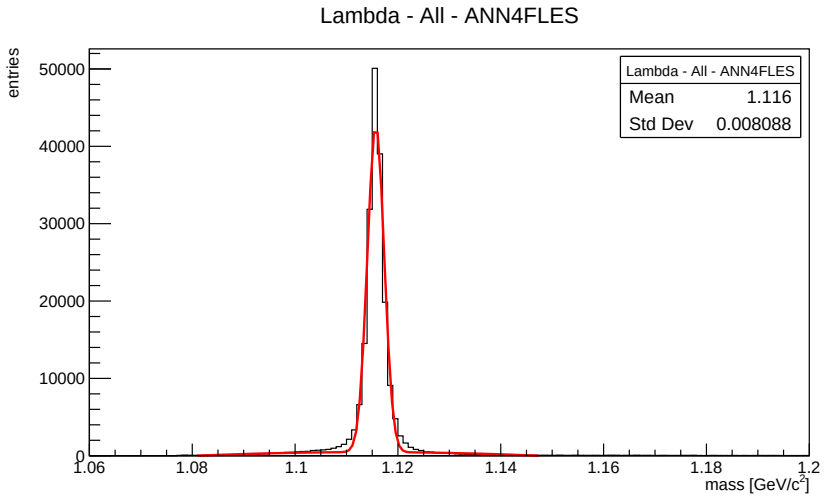


Figure 5.12: Fitted histogram of of all reconstructed Λ particles using the ANN4FLES approach in the mass range of 1.06 GeV/c^2 to 1.2 GeV/c^2. The black function shows the unfitted distribution of the neural network approach, the red one the fitted distribution.

One can see that the signal/background ratio has been increased by a factor of 10.97, using the neural network based classification approach to reject mistakenly reconstructed Λ particles. This confirms the previously shown results that were compared to MC information, that the amount of background was reduced by a factor that is much larger than the amount of falsely rejected signal. Furthermore, the significance has improved by almost 14%. While both approaches have a significance that is clearly above the threshold, and therefore both approaches show clear peaks as expected, this improvement in the significance suggests that the peak produced by the signal has become more clear compared to the default approach, using the neural network.

	S/B Ratio	Significance
Default	3.49	11.38
ANN4FLES	38.28	12.95
Improvement factor	**10.97**	**1.14**

Table 5.4: Summary of the results of signal/background ratio and significance, including the factor of improvement.

Chapter 6

Conclusion

The future Facility for Antiproton and Ion Research (FAIR), build in Darmstadt, Germany, has four major research pillars. One of them is the Compressed Baryonic Matter (CBM) experiment, which has as the object to investigate the quantum chromo dynamics (QCD) phase diagram at the region of high baryon densities. One interest is the investigation of Quark Gluon Plasma (QGP), which is expected to be indicated by enhanced strangeness. Since Λ is a strange particle that frequently occurs in the energy range of the CBM experiment, it is chosen to be investigated within this thesis.

To improve the quality of the physics analysis, the objective of this work is to increase the signal/background ratio of Λ. For this, a deep neural network was deployed, using ANN4FLES in the KF Particle Finder package.

The signal/background ratio can be increased by either increasing the signal or reducing the background. Since the reconstruction of the signal depends not only on the KF Particle Finder, but also on the quality of the reconstructed tracks, it is difficult to improve the signal with the presented approach. Reducing the combinatorial and physical background generated by the KF Particle Finder package, on the other hand, is possible with the approach of employing a neural network. However, it is important that the signal is not additionally reduced by the new approach.

The ANN4LES package was used to create a neural network that could then be inserted into the KF Particle Finder. The generated neural network for classifying Λ signal/background, consisting of 10 input neurons, 4 hidden layers consisting of 64 neurons each, and an output layer with 2 neurons, achieved a pure classification accuracy of nearly 99% on the validation dataset.

The pre-trained neural network was implemented in the KF Particle Finder to classify the signal and background of the Λ particles on a test data set of 46,000 events. The generated histograms showed that the background could be significantly reduced with the help of the neural network. In particular, the reduction of the combinatorial background is of particular importance, since this background has no added physical value, consisting purely of falsely generated particles. It could be shown that the combinatorial background could be drastically reduced.

To evaluate the results from a physical point of view, the signal/background ratio

was approximated using a function fitting. The Gaussian fit (signal) and the 2nd order polynomial fit (background) were each calculated over the mass distribution of Λ. The resulting signal/background ratio showed that the signal/background ratio could be improved by a factor of 10.97. Thus, the goal of this work, to optimize the signal/background ratio, was achieved. In addition, significance was determined to confirm the quality of the results. It was shown that the significance was improved by 14%. Even though the neural network based approach reconstructed slightly less signal particles, the reduction of signal was only a negligible factor of 1.36% in comparison to the default approach.

The obtained results show that the use of neural networks in the reconstruction of generated particles from heavy ion collisions can help to improve the physical analysis. The approach shown to classify Λ particles is a first step in cleaning up the results. Although Λ is already well reconstructed by the KF Particle Finder package, it could be shown that improvements, especially in reducing combinatorial background, are possible.

A next step would be to apply the approach shown to rarer particles to investigate whether signal/background classification could help reduce background in this region. In particular, combinatorial background interferes with physical analysis and should be almost completely removed.

The results generated in this thesis are based on data simulated with the UrQMD model. For a final evaluation, it should be investigated how the use of a neural network for background reduction behaves with real data. Especially since it is not obvious which patterns the neural network learns, a detailed analysis is important. For this purpose, for example, Shapley Additive exPlanations (SHAP) values can be determined, by means of which it is possible to analyze on which of the input parameters the neural network focuses in order to compare these results with physical theories. This would help to estimate the behavior of the neural network and to verify the results from a physical point of view.

In summary, the presented neural network was able to reject the background of Λ efficiently and thus, helped to improve the physics analysis quality of the CBM experiment by increasing the signal/background ratio significantly.

List of Figures

List of Tables

Listings

Abbreviation list

CBM Compressed Baryonic Matter

FAIR Facility for Antiproton and Ion Research

GSI Gesellschaft für Schwerionenforschung

NUSTAR Nuclear Structure, Astrophysics and Reactions

PANDA Antiproton Anihilation at Darmstadt

APPA Atomic, Plasma Physics and Applications

QCD Quantum Chromo Dynamics

LHC Large Hadron Collider

RHIC Relativistic Heavy Ion Collider

MVD Micro-Vertex Detector

STS Silicon Tracking System

RICH Ring Imaging Cherenkov detector

MuCh Muon Chambers

TRD Transition Radiation Detector

TOF Time Of Flight detector

ECAL Electromagnetic CALorimeter

PSD Projectile Spectator Detector

MRPC Multigap Resistive Plate Chambers

QGP Quark-Gluon Plasma

KF Particle Finder Kalman Filter Particle Finder

FLES package First Level Event Selection package

CA Track Finder Cellular Automaton Track Finder

KF Track Fit Kalman Filter Track Fit

SIMD Single Instruction, Multiple Data

PDG Particle Data Group

UrQMD Ultra-relativistic Quantum Molecular Dynamics

MC Monte-Carlo

AI Artificial Intelligence

ML Machine Learning

MLP Multilayer Perceptron

KL divergence Kullback-Leibler divergence

SGD stochastic gradient descent

Adam Adaptive Moment Estimation

SOM Self Organizing Map

NDF number of degrees of freedom

ANN4FLES Artificial Neural Networks for First Level Event Selection

Leaky ReLU Leaky Rectified Linear Unit

FIAS Frankfurt Institute for Advanced Studies

Arg Max Arguments of the Maxima

SHAP Shapley Additive exPlanations

References

[1] R. S. Bhalerao, ‘‘Relativistic heavy-ion collisions,’’ 2014.

[2] D. C. Shen. `https://u.osu.edu/vishnu/`. [Online; accessed 2023-06-22].

[3] U. Alberica Toia, ‘‘Participants and spectators at the heavy-ion fireball.’’ https://cerncourier.com/a/participants-and-spectators-at-the-heavy-ion-fireball/, April 2013. [Online; accessed 2023-09-07].

[4] M. Zyzak, *Online selection of short-lived particles on many-core computer architectures in the CBM experiment at FAIR*. doctoral thesis, 2016.

[5] S. Bass, ‘‘Microscopic models for ultrarelativistic heavy ion collisions,’’ *Progress in Particle and Nuclear Physics*, vol. 41, 1998.

[6] S. Glässel, ‘‘Simulationsstudien zur Identifikation von leichten Kernen und Hyperkernen mit dem CBM-TRD,’’ Master’s thesis, Goethe-Universität Frankfurt am Main, Institut für Kernphysik FB 13, 2019.

[7] J. Rafelski and B. Müller, ‘‘Strangeness production in the quark-gluon plasma,’’ *Phys. Rev. Lett.*, vol. 48, Apr 1982.

[8] P. Zyla and P. D. Group, ‘‘Prog. theor. exp. phys.,’’ vol. 2020, 2020.

[9] M. Zyzak, I. Kisel, I. Kulakov, and I. Vassiliev, *The KF Particle Finder package for short-lived particles reconstruction for CBM*, vol. 2013-1 of *GSI Report*. Darmstadt: GSI Helmholtzzentrum für Schwerionenforschung, 2013.

[10] B. Friman, C. Höhne, J. Knoll, S. Leupold, J. Randrup, R. Rapp, and P. Senger, eds., *The CBM Physics Book: Compressed Baryonic Matter in Laboratory Experiments*. Lecture Notes in Physics, Springer Berlin, Heidelberg, 1 ed., 2011.

[11] H. Gutbrod, *FAIR baseline technical report - Volume 1 Executive Summary*. Darmstadt: GSI, 2006.

[12] N. Kalantar-Nayestanaki and A. Bruce, ‘‘NUSTAR: NUclear Structure Astrophysics and Reactions at FAIR,’’ *Nuclear Physics News*, vol. 28, no. 3, 2018.

[13] PANDA Collaboration, W. Erni, *et al.*, ‘‘Physics Performance Report for PANDA: Strong Interaction Studies with Antiprotons,’’ 2009.

[14] H. Gutbrod, ed., *FAIR Baseline Technical Report Volume 5 - Experiment Proposals on Atomic, Plasma and Applied Physics (APPA)*. Darmstadt: GSI, March 2006.

[15] FAIR/GSI, ‘‘FAIR-the machine.’’ https://www.gsi.de/en/researchaccelerators/fair/the_machine. [Online; accessed 2023-06-30].

[16] P. Senger and V. Friese, ‘‘CBM Progress Report 2022,’’ Tech. Rep. CBM PR 2022, Darmstadt, 2022.

[17] V. Friese, ‘‘Computational Challenges for the CBM Experiment,’’ in *Mathematical Modeling and Computational Science* (G. Adam, J. Buša, and M. Hnatič, eds.), (Berlin, Heidelberg), Springer Berlin Heidelberg, 2012.

[18] N. Hermann, ‘‘Status of CBM and expected FAIR day-1/full results,’’ in *Workshop on Highly Baryonic Matter at RHIC-BES and Future Facilities - beyond the Critical Point towards Neutron Stars*, (University of Tsukuba, Japan), University of Heidelberg, Germany, 2023.

[19] FAIR/GSI, ‘‘Fair research.’’ https://www.gsi.de/en/researchaccelerators/fair/research. [Online; accessed 2023-07-05].

[20] FAIR/GSI/CBM, ‘‘Physics.’’ https://www.cbm.gsi.de/physics. [Online; accessed 2023-07-05].

[21] FAIR, ‘‘Nuclear matter physics.’’ https://fair-center.de/user/experiments/nuclear-matter-physics. [Online; accessed 2023-07-05].

[22] NuPECC, N. P. E. C. Committee, and A. Bracco, eds., *NuPECC long range plan 2017 : perspectives in nuclear physics*. Strasbourg: European Science Foundation, 2017.

[23] V. Friese and for the CBM Collaboration, ‘‘The high-rate data challenge: computing for the CBM experiment,’’ *Journal of Physics: Conference Series*, vol. 898, oct 2017.

[24] P. Kisel, *KF particle finder package: missing mass method for reconstruction of strange particles in CBM (FAIR) and STAR (BNL) experiments*. doctoral thesis, 2023.

[25] I. Deppner and N. Herrmann, ‘‘The CBM Time-of-Flight system,’’ *Journal of Instrumentation*, vol. 14, sep 2019.

[26] H. Bannwarth, B. P. Kremer, and A. Schulz, *Basiswissen Physik, Chemie und Biochemie: Vom Atom bis zur Atmung – für Biologen, Mediziner und Pharmazeuten*. Berlin: Springer-Verlag GmbH Deutschland, 3 ed., 2021. Erweiterte und aktualisierte Auflage.

[27] I. Kisel, ‘‘Superphysics and supercomputers, introduction to experimental physics.’’ Presentation, August 2021.

[28] MissMJ, ‘‘Standard model of elementary particles.’’ https://en.wikipedia.org/wiki/File:Standard_Model_of_Elementary_Particles.svg, 2013. Source: PBS NOVA [1], Fermilab, Office of Science, United States Department of Energy, Particle Data Group. Copyright MissMJ, licensed under Creative Commons Attribution 3.0 Unported license.

[29] I. Volker Hertel and Claus-Peter Schulz, *Atome, Moleküle und optische Physik 1*. Heidelberger Platz 3, 14197 Berlin, Germany: Springer-Verlag GmbH Deutschland, 2 ed., 2017. Atome und Grundlagen ihrer Spektroskopie.

[30] GSI Helmholtzzentrum für Schwerionenforschung, ''Compressed Baryonic Matter (CBM) at FAIR.'' https://www.gsi.de/work/forschung/cbmnqm/cbm, 2023.

[31] ''The quest for the Origin of Matter: Future research with high-energy heavy-ion beams at FAIR in Darmstadt: The Compressed Baryonic Matter Experiment.'' https://www.gsi.de/work/forschung/cbmnqm/cbm; Windows Media Video. [Online; accessed 2023-09-07].

[32] V. Akishina, I. Kisel, I. Kulakov, and M. Zyzak, ''FLES–First Level Event Selection Package for the CBM Experiment,'' *GPUHEP2014 Proceedings*, 2014. DOI: http://dx.doi.org/10.3204/DESY-PROC-2014-05/4.

[33] I. Kisel, I. Kulakov, and M. Zyzak, ''Standalone First Level Event Selection Package for the CBM Experiment,'' *IEEE Transactions on Nuclear Science*, vol. 60, no. 5, 2013.

[34] CBM Collaboration, V. Friese, *et al.*, ''The high-rate data challenge: computing for the CBM experiment,'' *Journal of Physics: Conference Series*, vol. 898, 2017. 54.02.03; LK 01.

[35] V. Akishina, I. Kisel, P. Kisel, P. Senger, I. Vassiliev, and M. Zyzak, ''Reconstruction of Particles Produced at Different Stages of Heavy Ion Collision in the CBM Experiment at FAIR.'' Poster, February 5-11 2017.

[36] R. E. Kalman, ''A New Approach to Linear Filtering and Prediction Problems,'' *Journal of Basic Engineering*, vol. 82, 03 1960.

[37] R. E. Kalman and R. S. Bucy, ''New Results in Linear Filtering and Prediction Theory,'' *Journal of Basic Engineering*, vol. 83, 03 1961.

[38] R. Marchthaler and S. Dingler, *Kalman-Filter: Einführung in die Zustandsschätzung und ihre Anwendung für eingebettete Systeme*. Springer Vieweg Wiesbaden, 1 ed., 2017. Softcover ISBN: 978-3-658-16727-1.

[39] G. Einicke and L. White, ''Robust extended Kalman filtering,'' *IEEE Transactions on Signal Processing*, vol. 47, no. 9, 1999.

[40] V. Akishina and I. Kisel, ''Parallel 4-Dimensional Cellular Automaton Track Finder for the CBM Experiment,'' *Journal of Physics: Conference Series*, vol. 762, oct 2016.

[41] S. Gorbunov, U. Kebschull, I. Kisel, V. Lindenstruth, and W. Müller, ''Fast SIMDized Kalman filter based track fit,'' *Computer Physics Communications*, vol. 178, no. 5, 2008.

[42] V. Ashikina, ''Time-based particle reconstruction and event selection in the cbm experiment.'' presentation.

[43] M. Bleicher, E. Zabrodin, C. Spieles, S. A. Bass, C. Ernst, S. Soff, L. Bravina, M. Belkacem, H. Weber, H. Stöcker, and W. Greiner, ''Relativistic hadron-hadron collisions in the ultra-relativistic quantum molecular dynamics model,'' *Journal of Physics G: Nuclear and Particle Physics*, vol. 25, sep 1999.

[44] A. Banerjee, I. Kisel, and M. Zyzak, ''Artificial neural network for identification of short-lived particles in the CBM experiment,'' *Int. J. Mod. Phys. A*, vol. 35, no. 33, 2020.

[45] C. Amsler *et al.*, ''Particle Data Group,'' *Physics Letters B*, vol. 667, 2008.

[46] Y. Fisyak, V. Ivanov, H. Ke, I. Kisel, P. Kisel, G. Kozlov, S. Margetis, A. Tang, and I. Vassiliev, ''Missing Mass Method for Short-Lived Particle Reconstruction in the CBM and STAR Experiments,'' in *9th International Conference on Distributed Computing and Grid Technologies in Science and Education*, 2021.

[47] I. Kisel, ''AI Techniques for Event Reconstruction.'' Presentation.

[48] GSI, ''cbmroot.'' `https://git.cbm.gsi.de/computing/cbmroot`, 2023. GitLab Community Edition.

[49] J. McCarthy, M. L. Minsky, N. Rochester, and C. E. Shannon, ''A proposal for the darthmouth summer research project of artificial intelligence,'' 1955.

[50] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Adaptive Computation and Machine Learning, The MIT Press, Massachusetts Institute of Technology, 2016.

[51] F. Rosenblatt, ''The perceptron: a probabilistic model for information storage and organization in the brain,'' *Psychological review*, vol. 65 6, 1958.

[52] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, ''Gradient-based learning applied to document recognition,'' *Proceedings of the IEEE*, vol. 86, no. 11, 1998.

[53] S. Hochreiter and J. Schmidhuber, ''Long short-term memory,'' *Neural Computation*, vol. 9, no. 8, 1997.

[54] I. Sarker, ''Deep learning: A comprehensive overview on techniques, taxonomy,'' *Springer*, 2021.

[55] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, ''Imagenet: A large-scale hierarchical image database,'' 2009.

[56] R. Rojas, *Neural Networks, A Systematic Introduction*. Springer Berlin, Heidelberg, 1 ed., 1996.

[57] G. D. Rey and K. F. Wender, *Neuronale Netze - Eine Einführung in die Grundlagen, Anwendungen und Datenauswertung*. Hans Huber, Hogrefe AG, Bern, 1 ed., 01 2008.

[58] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*. Springer Series in Statistics, New York, NY: Springer New York, NY, 2 ed., 2009. Published: 26 August 2009.

[59] T. Kohonen, *Self-Organizing Maps*. Springer Series in Information Sciences, Berlin, Heidelberg: Springer Berlin, Heidelberg, 3 ed., 2001. Published: 06 December 2012.

[60] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction, Second Edition*. The MIT Press, 2 ed., 2018. Published: November 13, 2018.

[61] J. Joyce, *Kullback-Leibler Divergence*. Springer, Berlin, Heidelberg, 2011. Published: 02 December 2014.

[62] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, ‘‘Learning representations by back-propagating errors,’’ *Nature*, vol. 323, Oct. 1986.

[63] S. Ruder, ‘‘An overview of gradient descent optimization algorithms,’’ 2017.

[64] D. P. Kingma and J. Ba, ‘‘Adam: A method for stochastic optimization,’’ 2017.

[65] J. Kubát, ‘‘Reconstruction of strange hadrons in collisions of nuclei at RHIC,’’ Master’s thesis, Czech Technical University in Prague, August 2020.