Improving the $t\bar{t}t\bar{t}$ event selection with Graph Neural Networks in multilepton final states at the ATLAS detector

MASTERARBEIT IN PHYSIK

VON

VAKHTANG ANANIASHVILI

ANGEFERTIGT IM

PHYSIKALISCHEN INSTITUT

VORGELEGT DER

MATHEMATISCH-NATURWISSENSCHAFTLICHEN FAKULTÄT DER

RHEINISCHEN FRIEDRICH-WILHELMS-UNIVERSITÄT BONN

NOVEMBER 2021

ATLAS EXPERIMENT

UNIVERSITÄT BONN

1. Gutachter: Prof. Dr. Markus Cristinziani

2. Gutachter: Prof. Dr. Florian Bernlochner

Abstract

A study on the use of Graph Neural Networks for the selection of $t\bar{t}t\bar{t}$ events in the ATLAS detector is presented. Data used is the Monte-Carlo simulated proton-proton collision events at $\sqrt{s} = 13\,\text{TeV}$. The analysis is only concerned with the same-sign multilepton channel. After optimization, GNNs achieved a performance of AUC $= 0.87435 \pm 0.00166$ which is an improvement over the previous studies conducted on the same data using Boosted Decision Trees and Feedforward Neural Networks.

Acknowledgements

First and foremost, I would like to thank Prof. Dr. Markus Cristinziani for the opportunity of conducting my research as a member of the ATLAS Collaboration. Prof. Cristinziani was extremely supportive throughout my studies, providing valuable feedback, and sharing knowledge and experience. I am also grateful to Prof. Dr. Florian Bernlochner, for being the second examiner of my master's thesis, and taking time to evaluate my work.

I would like to express my immense gratitude towards PD Dr. Akaki Rusetsky, for granting me the possibility of pursuing my master's degree studies at the University of Bonn, for being a great mentor and always showing support.

I am also very thankful to fellow members of Prof. Cristinziani's group. Dr. Ogul Öncel, was instrumental at the early stages of my research. He was always available for help and was eager to share his vast experience. Niklas Schwan was kind enough to walk me through the research he conducted on a similar topic a year earlier. He shared his code with me, which greatly assisted me in understanding the inner-workings of the Neural Networks. Katharina Voß and Gabriel Gomes helped me with the preparation of my master's thesis colloquium, by giving me valuable feedback.

I am very grateful to Dr. Peter Falke, for introducing me to the Graph Neural Networks and for convincing me to use this Neural Network architecture for my analysis. He guided me throughout my research by providing useful and practical knowledge, and was always available for quick questions as well as insightful discussions.

Lastly, I would like to thank my fellow students, Lado Razmadze and George Chanturia who were by my side throughout my studies, and were always willing to lend a helping hand.

Contents

1

Introduction

Understanding the world of elementary particles holds the keys to uncovering the inner workings of the universe. Decades of research have culminated into the most precise description of particle physics, the Standard Model. Despite its remarkable success, there still remain physical phenomena that are yet to be explained.

Construction of the Large Hadron Collider, the largest particle accelerator in the world, allows for producing and detecting extremely rare and highly energetic processes, such as the simultaneous production of four top quarks. The precise measurement of this process can give insight into Beyond Standard Model physics. The ATLAS detector is the largest general purpose particle detector, located around the Large Hadron Collider. It produces an inordinate amount of data, which needs to be rigorously processed to detect the events we are looking for.

Recent advances in the field of Machine Learning have aided the efforts of particle physicists with this task. This thesis will explore the use of Graph Neural Networks for the signal-background separation of the $t\bar{t}t\bar{t}$ events, in multilepton final states. Graph Neural Networks are a novel machine learning architecture that operate on mathematical graph structures. Representing particle physics data as graphs, allows for capturing complicated relationships between different particles.

This thesis will start by introducing the Standard Model and the top quark. The next two chapters will concern the ATLAS experiment and the data used for this analysis. The following two chapters will explain the basic concepts behind machine learning algorithms, as well as Graph Neural Networks. Chapter 7 will summarize the results of previous studies conducted on the same data, using different machine learning algorithms. Lastly, Chapter 8 will document the findings of this analysis, and compare the achieved performance to the results presented in Chapter 7.

2

Theoretical Background

This chapter will provide theoretical background necessary for understanding elementary particle physics processes that are the focus of this research. Firstly, the Standard Model (SM) of particle physics will be introduced, which represents the most precise description of elementary particles to date.

Section 2.2 presents the top quark, which is of particular interest for our analysis, and how this particle stands out among other SM particles. Further, this section provides a brief review of the possible production and decay channels of the top quark.

Lastly, Section 2.3 will describe the simultaneous production of four top quarks, which is one of the energetically highest processes accessible at the Large Hadron Collider. Possible decay channels of this process and their categorization will be explained, in order to outline the focus of this analysis.

2.1 Standard Model

The Standard Model of particle physics describes two types of elementary particles, namely *fermions* and *bosons* [1]. These names come from the spin statistics that these particles obey. Fermions have a half-integer spin and therefore are described by Fermi-Dirac statistics. Bosons have an integer spin and are in turn described by Bose-Einstein statistics. Particles with half-integer spin obey the Pauli exclusion principle, while the particles with integer spin do not. Because of this property, fermions are the building blocks of matter, while bosons are the force-mediating particles of 3 fundamental forces of our universe.

There are 12 fermions, each with a corresponding anti-particle. Anti-particles have similar properties to particles, only differing in the sign of their additive quantum numbers (e.g. electric charge, baryon number, lepton number). These 12 particles are further subdivided into two classes, *quarks* and *leptons*. Both of these classes have pairs of particles arranged in 3 generations.

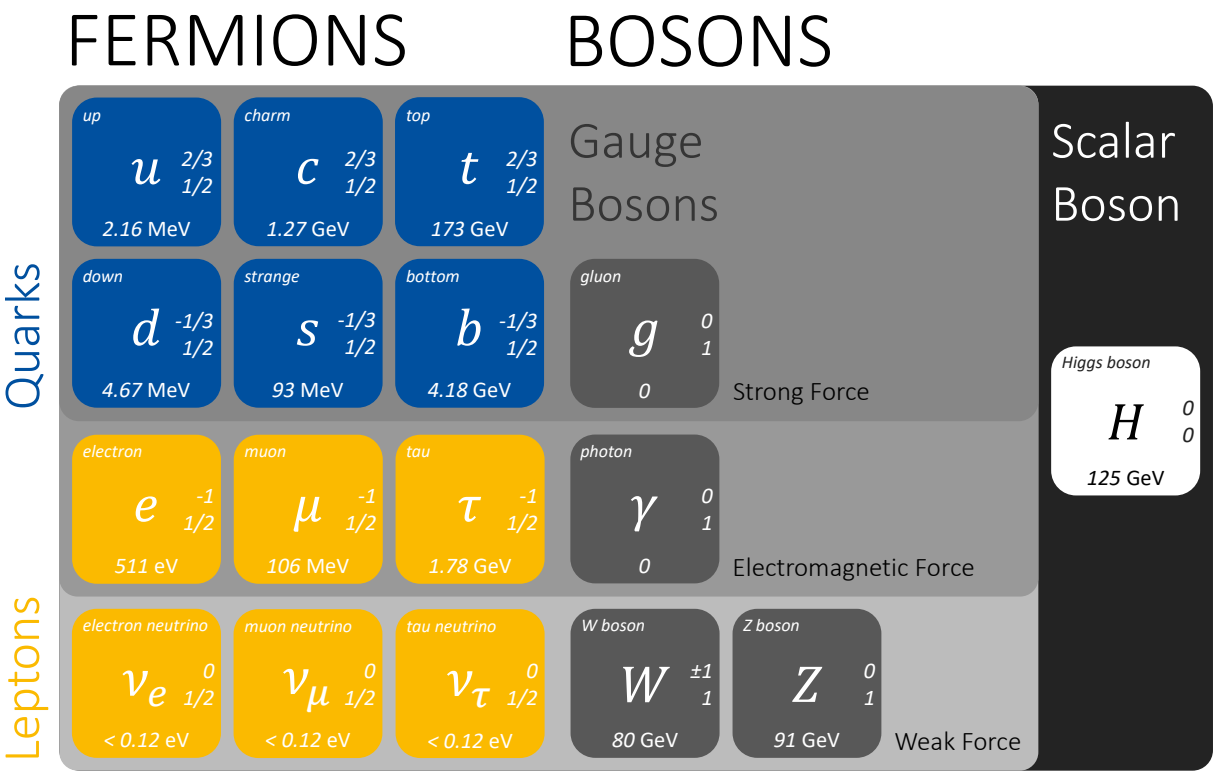


Figure 2.1: Elementary particles of the Standard Model of particle physics.

Quarks carry color charge (r, g, b) and therefore interact via the strong interaction. Because of the nature of the strong interaction, quarks are confined in colorless particles and are never found in nature independently. These composite particles are called *hadrons*, and exist in two types: *mesons* and *baryons*. Mesons consist of a quark and an anti-quark with a color-anticolor combination, and (anti-)baryons consist of 3 (anti-)quarks each with a different color.

Quarks also have an electric charge and interact with the electromagnetic field. Each generation has a quark with electric charge $+\frac{2}{3}$ and $-\frac{1}{3}$. Due to the color confinement described above, hadrons always end up with an integer electric charge.

As it has already been mentioned, quarks come in 3 generations of weak isospin doublets: (u, d), (c, s), (t, b). The strength of the weak coupling of quarks is described by the CKM matrix. Up-type quarks (u, c, t) can only decay to down-type quarks (d, s, b) and vice versa.

Leptons do not carry color charge, hence they do not interact with the strong field. Electron, muon, and tau each have an electric charge of -1, while all neutrinos are electrically neutral, thus they only interact through the weak interaction.

Only first generation fermions are stable. Higher generation fermions quickly decay to the first generation fermions with very short lifetimes. Therefore ordinary matter is composed of particles from the first generation.

The forces in the Standard Model are mediated by gauge bosons. Gluons are the massless electrically neutral mediators of the strong force, with 8 possible color charge configurations. The electromagnetic force is mediated by photons which are also massless, colorless and chargeless. The weak interaction is governed by the exchange of 3 gauge bosons. W^+ and W^- are massive electrically charged gauge bosons with electric charges $+1$ and -1 respectively. Z^0 is also massive, but electrically neutral.

The latest addition to the Standard Model is the Higgs boson. The existence of this particle was theoretically proposed in 1964, and experimentally confirmed in 2012 [2]. The Higgs boson is a massive, spin-zero boson. It is the only particle in the Standard Model without spin. This particle is an evidence of the existence of the scalar Higgs field, which is responsible for giving mass to all the massive particles in the Standard Model through the Higgs mechanism.

Despite being the best description of particle physics to date, the Standard Model fails to explain a number of phenomena. The biggest shortcoming is the failure of describing the fourth fundamental force of gravity, and its interaction with SM particles. Some other issues that are yet to be explained are the nature of Dark Matter and the reason behind the baryon asymmetry.

2.2 Top Quark

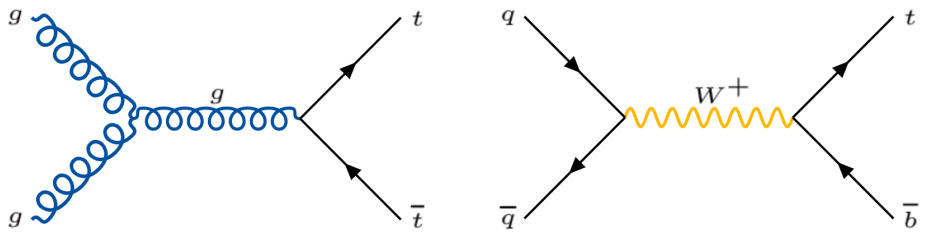
The top quark was the last quark to be discovered, in 1995 at Fermilab [3, 4]. It stands out as the most massive elementary particle of the Standard Model [1]:

$$m_t = 172.8 \pm 0.3\,\text{GeV}, \tag{2.1}$$

making it more than 40 times heavier than the bottom quark, and even slightly heavier than a whole Tungsten (^{74}W) atom.

There are several relevant processes for the production of top quarks, assuming the center-of-mass energy is sufficiently high. These processes can be classified based on the amount of final state top quarks in two categories: top-quark pair production, and single-top production.

$t\bar{t}$ pair production via the strong interaction is the dominant mode. Example of this process can be seen on Figure 2.2a below. The 3-gluon vertex can also be replaced by a $q\bar{q}$ annihilation into a highly energetic gluon. The gluon propagator can in turn be replaced by a photon or a Z^0 boson, however these processes are highly suppressed.



(a) $t\bar{t}$ pair production (b) Single t production

Figure 2.2: Main channels of top-quark production

Single top quark can only be produced via weak interaction, making this process relatively unlikely compared to the $t\bar{t}$ pair production, since the weak interaction is suppressed by the W boson mass. Due to this reason, the production of a single top quark was experimentally confirmed 11 years later in 2009, also at Fermilab [5, 6].

Because of its large mass, the top quark has a very short lifetime [1]:

$$\tau \approx 5 \cdot 10^{-25}\text{s}. \tag{2.2}$$

This value is so small that t quarks decay before they hadronize. The only known decay channel is through the weak interaction, where a t quark decays into a down-type quark and a W boson. Since, the diagonal terms of the CKM matrix are significantly larger compared to the off-diagonal terms, i.e $|V_{tb}| \gg |V_{ts}| \gg |V_{td}|$, the t-to-b transition is much more likely compared to the other two quarks, with a relative branching ratio of [1]:

$$\frac{\Gamma_{t\to Wb}}{\Gamma_{t\to Wq}} = 0.957 \pm 0.0340. \tag{2.3}$$

2.3 $t\bar{t}t\bar{t}$ Production

The simultaneous production of four top quarks is one of the most energetic final state seen at the Large Hadron Collider. Owing to the incredibly high center-of-mass energies required for the production, this process is very rare. The expected SM cross section for the $t\bar{t}t\bar{t}$ production at $\sqrt{s} = 13$ TeV, according to the latest calculation at Next-to-Leading Order (NLO) in QCD and EW couplings [7], is:

$$\sigma_{t\bar{t}t\bar{t}} = 11.97^{+18\%}_{-21\%} \text{ fb}. \tag{2.4}$$

A precise measurement of this process is of particular interest, since many Beyond Standard Model (BSM) theories predict the enhancement of this cross section [8, 9, 10].

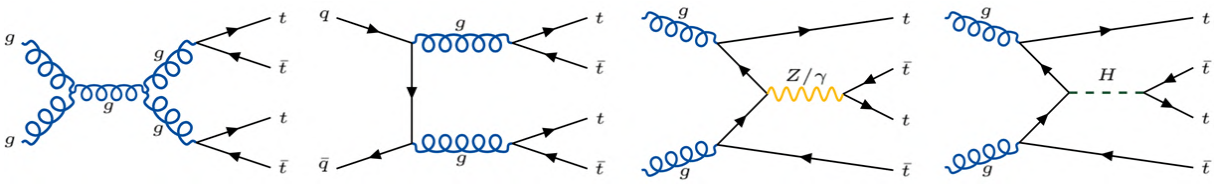


Figure 2.3: Feynman diagrams of Leading Order (LO) $t\bar{t}t\bar{t}$ production in the Standard Model.

As it was already mentioned in the previous section, each one of the 4 quarks will decay into a bottom quark and a $W^{\pm}$ boson. $W^{\pm}$ bosons in turn decay either hadronically, into a quark-antiquark pair, or leptonically, into a lepton and a neutrino of the same flavour.

$$t \to b + W^{\pm} \qquad \begin{array}{lll} \text{hadronic:} & W^{\pm} \to q + \bar{q} & 67\% \text{ [1]} \\ \text{leptonic:} & W^{\pm} \to l + \nu & 33\% \text{ [1]} \end{array}$$

As a result, the final state consists of at least four b-jets, and several leptons. The branching ratios for various hadronic and leptonic decay combinations can be seen in Table 2.1. These decay channel combinations can be classified according to the number and charge of the final state leptons. The **1LOS** (one lepton, opposite sign) channel includes decays with only one charged lepton, or 2 charged leptons with opposite sign. The **SSML** (same-sign, multilepton) channel consists of decays that result in at least 2 leptons of the same sign in the final state.

		1LOS		SSML		
	hhhh	*hhhl*	*hhll* OS	*hhll* SS	*hlll*	*llll*
BR	0.311	0.422	0.143	0.072	0.049	0.004

Table 2.1: Branching ratios of the $t\bar{t}t\bar{t}$ decay [11].

Note that the branching ratios for hadronic and leptonic decays cannot simply be combined to get the values of the table. The reason behind this is that a portion of leptonically decaying W's decays into a τ lepton, which in turn can decay hadronically. The goal of this research will be to improve identification accuracy of $t\bar{t}t\bar{t}$ events that decay through SSML channel.

3

ATLAS Experiment

The ATLAS experiment was built and is operated by a global collaboration involving roughly 3000 physicists from 40 different countries and is aimed at discovering new particles and physics phenomena by taking advantage of the unprecedented energies achieved by the Large Hadron Collider.

Section 3.1 provides basic information about the Large Hadron Collider itself. Section 3.2 introduces the ATLAS detector, the largest among the four general-purpose particle detectors located around the accelerator ring. Firstly, the coordinate system used for documenting events in the ATLAS detector will be explained, which will be followed by a description of the various components of this detector and the detector technologies that are implemented for particular tasks.

Lastly, Section 3.3 will detail the types of signatures left by elementary particles of the Standard Model, and the method of accounting for neutrinos, which escape the detector cavern undetected. The information provided in this chapter is based on Ref. [12].

3.1 Large Hadron Collider

The Large Hadron Collider or LHC is the largest and most powerful particle accelerator in the world. The accelerator sits in a 27-kilometer tunnel approximately 100 meters underground on the Swiss-French border near CERN headquarters.

The accelerator utilizes an array of superconducting dipole and quadrupole magnets, which are cooled down to -271.3°C, in order to confine the proton beams within the beam pipe and direct them around the accelerator. Protons travel at close to the speed of light in two separate beam pipes in opposite directions. When the beams are fully accelerated at the center-of-mass energy of 13 TeV, protons travel just 11 km/h slower than the speed of light, resulting in 11245 revolutions per second, and a collision rate of 40 MHz. Particles travelling through the beam pipes are concentrated in 2808 bunches, with each bunch containing approximately 1.15×10^{11} protons.

Such high energies and particle numbers are required for producing enough statistics for the observation of extremely rare events like $t\bar{t}t\bar{t}$. The number of collisions produced in a collider can be quantified by the **Luminosity** L:

$$L = \frac{1}{\sigma}\frac{dN}{dt} \qquad [L] = \mathrm{m}^{-2}\mathrm{s}^{-1} \tag{3.1}$$

where, σ is the cross-section for a given interaction and dN/dt describes the number of events per unit time. The LHC achieved luminosities of 10^{34} $\mathrm{cm}^{-2}\mathrm{s}^{-1}$ in June 2016, and has since surpassed this value.

Luminosities of this magnitude result in many interactions per bunch crossing. In general only a single interaction with the highest energy, called the *hard-scattering event*, is of interest, while the rest are background which needs to be removed. These background events are termed *pile-up*, and their removal is a challenging task.

Beams intersect at 4 distinct points along the circular trajectory. These intersection points are aimed to be at the center of 4 particle detectors located around the accelerator ring: ATLAS, CMS, ALICE and LHCb. These detectors surround each intersection point, and have layered structures optimal for detecting various types of secondary particles produced after the collision.

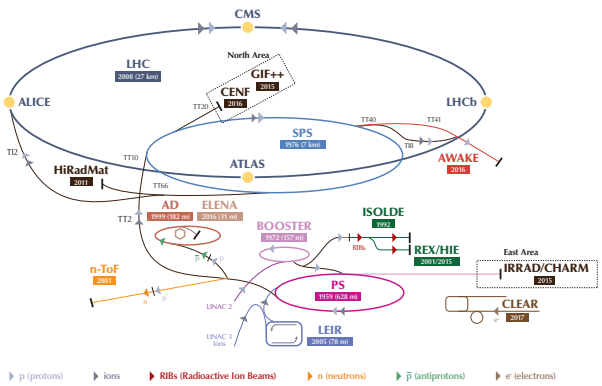


Figure 3.1: Diagram of the CERN accelerator complex [13].

3.2 ATLAS Detector

ATLAS (A Toroidal LHC ApparatuS) is the largest particle detector at the LHC. It is designed to take advantage of the unprecedented energies achieved at the LHC and discover unknown massive particles.

The detector is made of concentric cylindrical detecting layers specializing in detecting specific types of particles. The signature left by each particle in these layers allows for identification of the particle type, as well as its properties like mass, charge, energy and momentum. The detector has four major components:

- Inner Detector,
- Calorimeters,
- Muon Spectrometer,
- Magnet System,

each of which have a complex structure that will be discussed below in further detail.

Events in the ATLAS detector are described by the cylindrical coordinates (z, ϕ, η), where the radial coordinate z is chosen to be directed along the beam axis. ϕ is the azimuthal angle in a plain perpendicular to the beam axis. η is the **pseudorapidity**, which is the transformed polar angle θ:

$$\eta = -\ln\left[\tan\left(\frac{\theta}{2}\right)\right]. \tag{3.2}$$

As a result of this transformation, particles travelling perpendicular to the beam axis, correspond to $\eta = 0$, while particles travel along this axis with $\eta = \infty$. This choice has the benefit of having approximately a constant flux of particles per unit η.

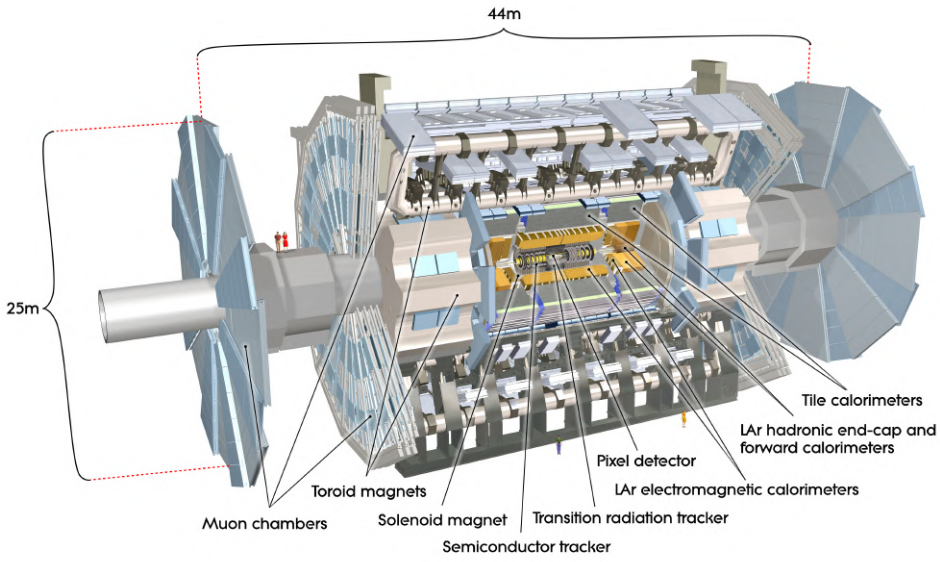


Figure 3.2: Computer generated image of the ATLAS detector [12].

Inner Detector

The inner detector is the first cylindrical layer located closest to the interaction point, where the density of particles is highest. This detector system has the highest granularity, in order to achieve a proper resolution close to the interaction vertex. The detector is used to reconstruct the tracks of the charged particles through their interaction with the medium at discrete points. Reconstructed tracks are used to deduce the particle type and its momentum, as well as for locating the interaction vertex where the particle originated from.

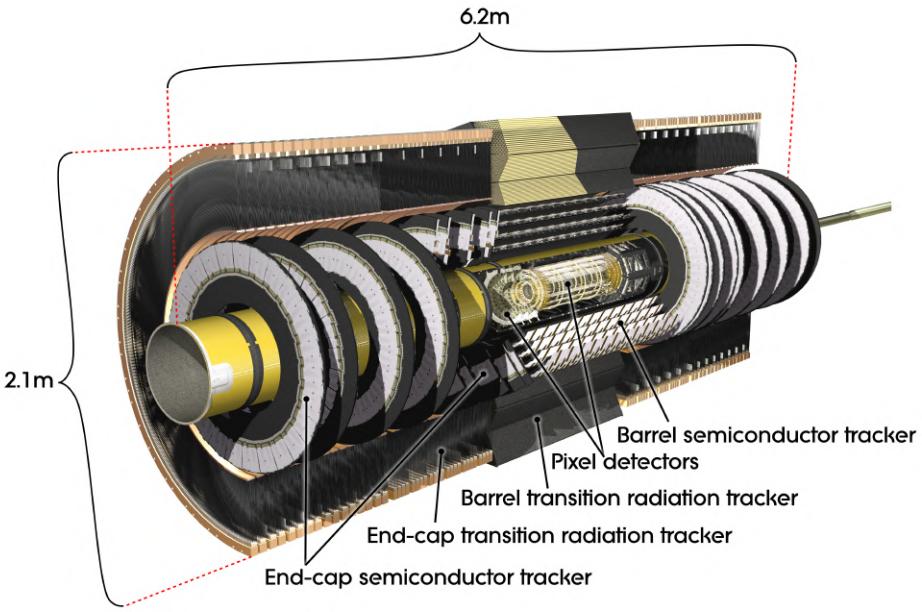


Figure 3.3: Computer generated image of the inner detector [14].

The inner detector consists of 3 parts:

- **Pixel Detector**: the innermost of the 3 detectors consists of 92 million pixels, distributed cylindrically around the beam axis in 3 layers, and in 3 disks for each end-cap region covering high pseudorapidity ranges. Most pixels have dimensions of 50×400 μm.
- **Semi-Conductor Tracker**: it utilizes four double layers of long narrow silicon strips of $80\mu\text{m} \times 12\text{cm}$ in size, in order to detect particle tracks over a larger area compared to pixel detector.
- **Transition Radiation Tracker**: the outermost of the 3 detectors is made up of 4 mm wide drift tubes. Drift tubes are gas filled tubes with a thin wire stretched out at the center, which is held at -1500 V. Charged particles going through the tubes, ionize the gas. Due to high voltage, negative ions drift towards the wire, producing a pulse signal. The resolution of the TRT is lower compared to the two inner detectors, which was a necessary compromise for covering a larger area.

Calorimeters

The calorimeters are situated around the inner detector, and are designed to fully absorb the energies of most particles, in order to bring them to a stop. Calorimeters are made up of alternating layers of high-density metal that slows down incoming particles and active medium used for measuring the deposited energies.

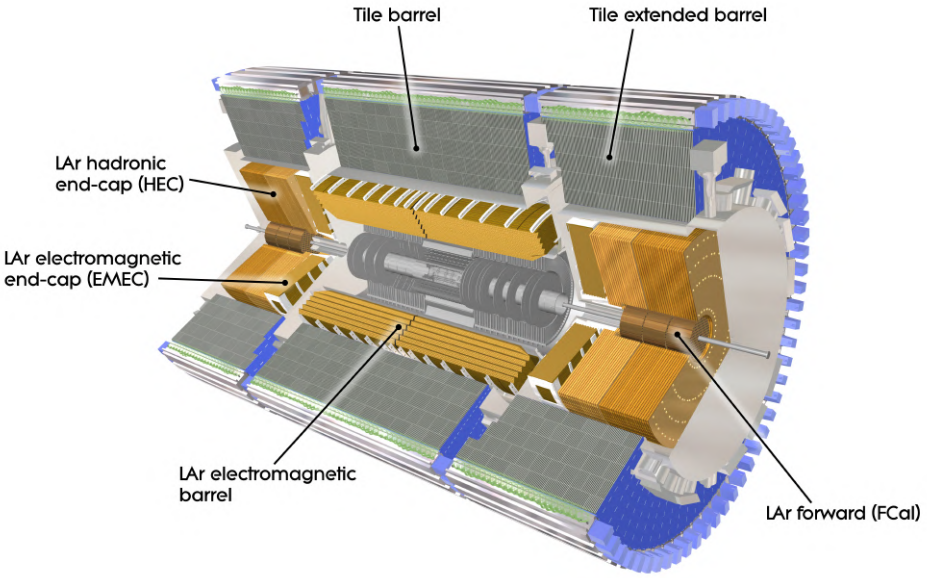


Figure 3.4: Structure of the calorimeter system [15].

The calorimeter system consists of 2 different calorimeters:

- **Electromagnetic (EM) Calorimeter**: a liquid argon calorimeter designed to measure energies of electrons and photons, as well as tau leptons and hadrons. It is composed of layers of metal with liquid argon in between. The particles interact with metal sheets, causing a shower of secondary particles. These particles ionize the liquid argon, which is kept at $-184°C$, and produce an electric signal. Using the signal pulse, the energies of the original particles can be determined.
- **Hadronic Calorimeter**: a tile calorimeter surrounding the EM calorimeter. It is used for measuring energies of particles that are not fully stopped within the previous layers, most of which are hadrons. Unlike the electromagnetic calorimeter, instead of liquid argon, the hadronic calorimeter utilizes plastic scintillating tiles sandwiched between steel plates. Showers produced in the layers of steel cause photon cascades inside the scintillating material, which are in turn converted to an electrical current. The magnitude of this signal is proportional to the energy of the original particle.

The only type of known particles that escape outside of the inner detector and calorimeter system, are muons and neutrinos.

Muon Spectrometer

The muon spectrometer is the last layer of the ATLAS detector. It covers a large cylindrical area, starting at 4.25 m away from the beam axis, right after the calorimeters, and extending up to 11 m. Such a large area coverage is necessary for reconstructing the muon tracks before they leave the detector. Muon detection is a vital factor for distinguishing between various physical processes.

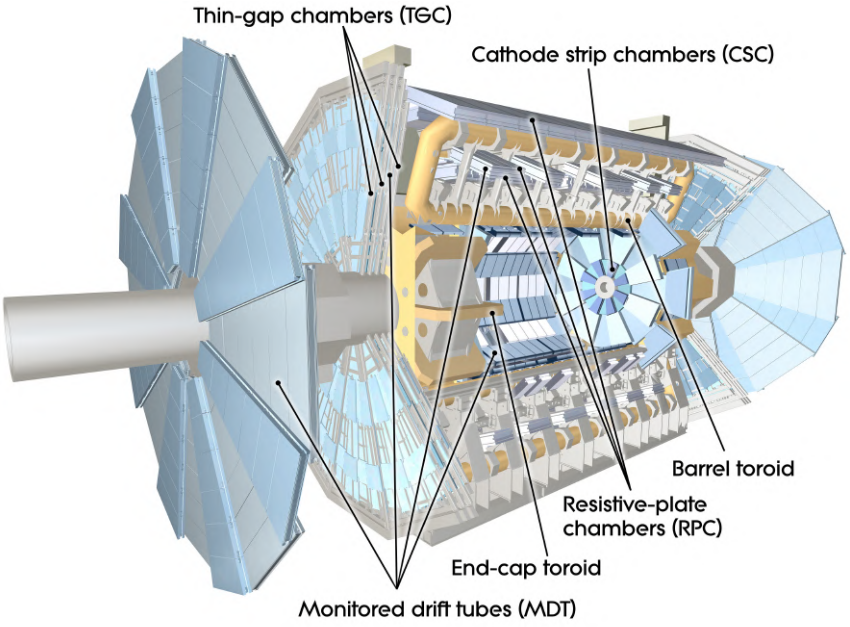


Figure 3.5: Structure of the muon spectrometer [16].

The muon spectrometer consists of 4 different detector technologies:

- For precision measurement:
  - Monitored Drift Tubes
  - Cathode Strip Chambers
- For trigger:
  - Resistive Plate Chambers
  - Thin Gap Chambers

The trigger system provides an accurate time resolution for passing particles, which is crucial for bunch crossing identification, as well as triggering the data recoding for the detecting chambers behind. These chambers measure the amount of ionized charge, giving insight about the energy of the muon. The locations of these signals are used to reconstruct the tracks of the muons and measure their momenta. The muon spectrometer has a 16-fold segmentation in azimuth. The chambers are oriented in such a way that each muon intersects 3 stations of chambers, which is necessary for track reconstruction.

Magnet System

ATLAS detector utilizes a magnetic system, in order to bend the trajectories of charged particles produced inside the detector. The behavior of the charged particles in the magnetic field gives us the information about their charge and momentum.

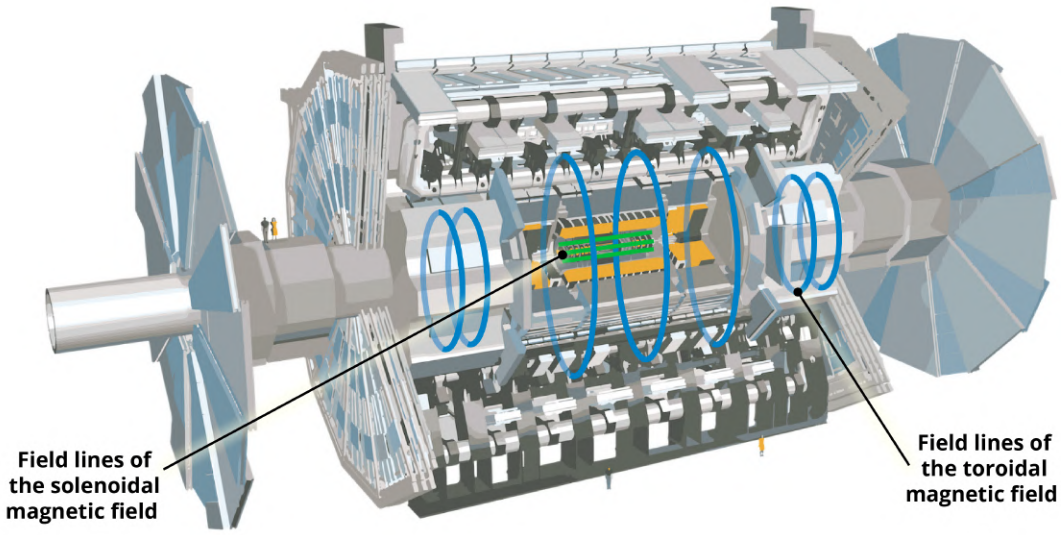


Figure 3.6: Field lines of the central solenoid are shown in green. Field lines of the 3 toroid magnets are shown in blue [17].

There are two different types of superconducting magnets deployed to achieve the desired field lines inside the detector:

- **Central Solenoid Magnet**: it surrounds the inner detector, is 5.6 m long and 2.56 m in diameter. It provides magnetic fields of approximately 2 T.
- **Toroid Magnets**: they are located outside of the calorimeters and are used for bending muons. The main barrel toroid is made up of 8 superconducting coils each in an individual cryostat. The two end-cap toroids, also consisting of 8 coils, which are housed in a single cryostat, are inserted at each end of the detector. Both toroids produce a field of 4 T at the surface of the superconductor.

Strong magnetic fields are necessary for accurately measuring the radius of curvature r, which is used for calculating the charge and momentum of the particle:

$$r = \frac{p}{qB}. \tag{3.3}$$

The direction of the curvature tells us whether we are dealing with a particle or an anti-particle, and the radius of curvature allows us to measure its momentum.

3.3 Particle Signatures

Figure 3.7 below depicts signatures each type of particle leaves inside the detector. Photons do not leave any tracks in the inner detector, since they are not charged, and deposit all their energies in the EM calorimeter. Unlike photons, electrons leave curved tracks in the inner detector, and are fully stopped by the EM calorimeter, as well. Hadrons can only be stopped by the Hadronic calorimeter. Just like photons, electrically neutral hadrons do not leave tracks in previous layers. Muons only interact slightly with various detector components, before exiting the detector. Their momenta are measured according to the curvature of their tracks.

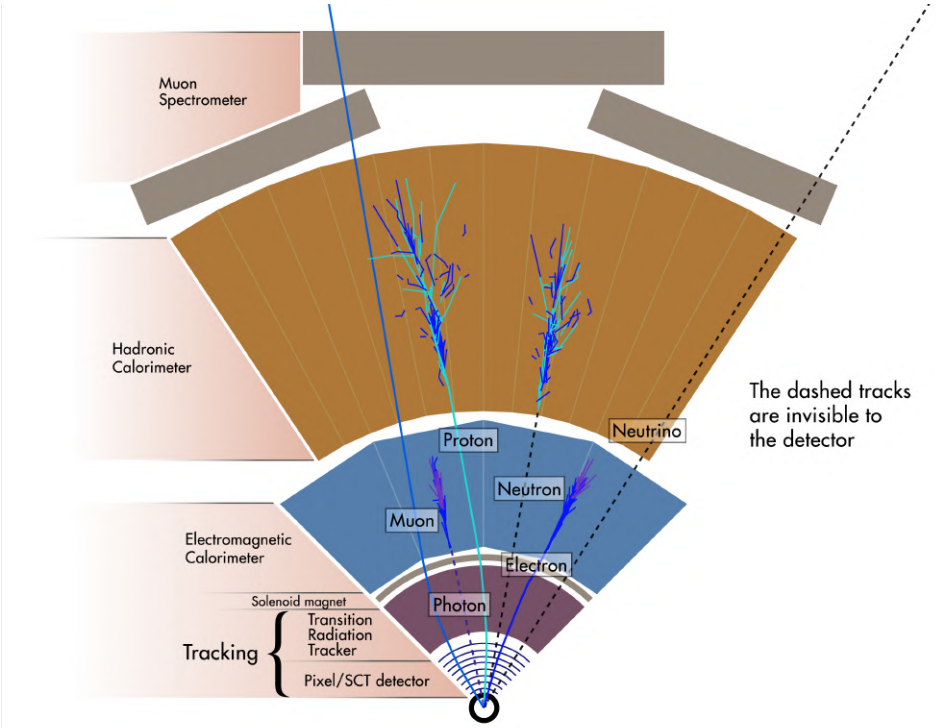


Figure 3.7: Diagram of particle paths in the detector [18].

At this point all Standard Model particles other than neutrinos have left a signature inside the detector. Neutrinos exit the detector cavern without any interactions, and the only way to account for their energies is to precisely measure the tracks of every other type of particle. This information can be used in combination with the law of conservation of momentum, to deduce the **missing transverse energy**:

$$E_T^{miss} = \sqrt{(E_x^{miss})^2 + (E_x^{miss})^2} \tag{3.4}$$

where,

$$E_x^{miss} = -\sum_i (p_x)_i \qquad\qquad E_y^{miss} = -\sum_i (p_y)_i \tag{3.5}$$

and the sum is over all reconstructed particles.

4

Data

The particles created in the proton-proton collisions at the LHC interact with the various components of the detector, and produce signals. This raw data is selectively recorded and put through several steps of pre-processing. Various algorithms are utilized to reconstruct particle types, their trajectories and energies in order to produce the datasets with reconstructed physical objects, such as individual particles or particle jets. This information needs further treatment, in order to identify the original process that resulted in such decay products.

The very point of the machine learning algorithm we are attempting to create, is to assign a correct label to an actual recorded event of the ATLAS detector. To achieve this goal, we need to have true information about the events in our dataset, in order to teach the algorithm what kind of signature each type of event leaves in the detector. Needless to say, that this information is unavailable in ATLAS datasets. To solve this problem, artificial datasets are produced through Monte-Carlo simulation.

This chapter will introduce the dataset used in this research, and briefly describe the methods used for its production. Further, it will present the procedure of event selection, and the definition of the signal region, as well as the concept of event weights and purposes of their application.

4.1 Monte-Carlo Simulation

Event simulation consists of three distinct stages, namely event generation, parton showering, and detector simulation. During the first stage events are generated and the immediate decay and hadronization of the produced particles are simulated. The next stage estimates the interaction of secondary particles with the detector, which is followed by a simulation of the detector response, by converting the interactions into voltages and currents. the final output format is identical to the data recorded by a true detector.

The list of software used for the event generation and parton showering can be seen in Table 4.1 below. Detector simulation of the signal samples ($t\bar{t}t\bar{t}$ LO & $t\bar{t}t\bar{t}$ NLO) was performed by AtlFastII. The detector simulation for the background samples was done with the combination of AtlFastII and FullSim. For details about the simulation process see Ref. [11].

Dataset	Event Generator	Parton Showering
$t\bar{t}t\bar{t}$ LO	MadGraph5_aMC@NLO	Pythia 8
$t\bar{t}t\bar{t}$ NLO	MadGraph5_aMC@NLO	Pythia 8
$t\bar{t}W$	Sherpa 2	Sherpa 2
$t\bar{t}WW$	MadGraph5_aMC@NLO	Pythia 8
$t\bar{t}Z$	MadGraph5_aMC@NLO	Pythia 8
$t\bar{t}H$	Powheg-Box v2	Pythia 8
$V+$ jets	Sherpa 2	Sherpa 2
VV	Sherpa 2	Sherpa 2
$t(\bar{t})X$	Powheg-Box v2	Pythia 8
$t\bar{t}$	Powheg-Box v2	Pythia 8
Other	MadGraph5_aMC@NLO	Pythia 8

Table 4.1: List of software used for event generation and parton showering.

The analysis uses two signal datasets: $t\bar{t}t\bar{t}$ produced at the LO and $t\bar{t}t\bar{t}$ at NLO QCD precision. Background samples consist of associated production of top quark with bosons, top quark pair production, diboson production, and associated production of gauge bosons and jets. The "Other" dataset in Table 4.1 includes the following events: $t\bar{t}t$, t, $t\bar{t}ZZ$, $t\bar{t}HH$, $t\bar{t}WH$, $t\bar{t}WZ$.

Generator Event Weights

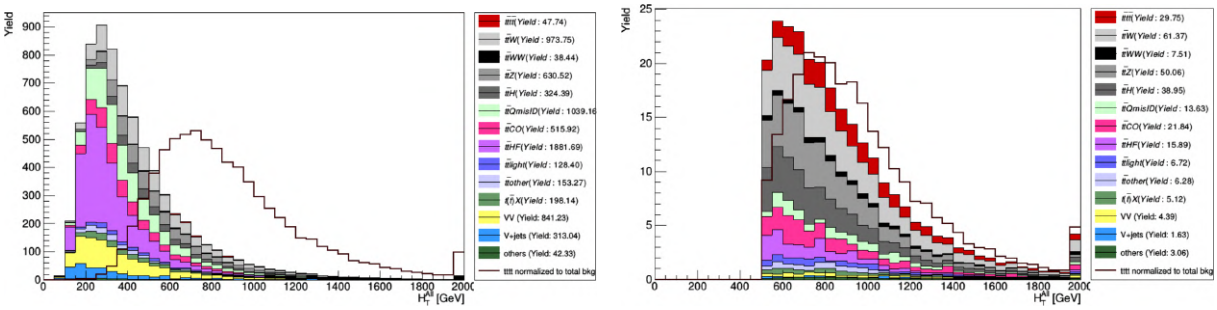
The relative number of events in the dataset for different processes is not proportional to the respective cross-sections, meaning the dataset is not a good representation of the expected signature in the ATLAS detector. Hence the number of generated events in the dataset needs to be scaled accordingly. Further scaling needs to be applied to account for variable factors like detector configuration, pile up, lepton scaling factors, b-tagging scaling factors, etc. To combat this, events are scaled with event weights. This scaled number of events is called *yield*.

Certain simulated events come with negative weights. During the event generation, the generators encounter divergences while integrating over virtual emissions. Because of this, the negative weights are utilized to make the overall event distributions correct.

4.2 Event Selection and Signal Region

The simulated datasets are subjected to a sequence of quality checks. Firstly, the overlap removal procedure is applied, to make sure for instance that energy deposited in calorimeters is not assigned to two different reconstructed objects. Next, events need to pass the preselection criteria, which imposes certain cuts on reconstructed particles in order to improve the signal-to-background ratio. Further details about this procedure can be found in Ref. [11]. Finally, cuts are applied for the selection of the signal region.

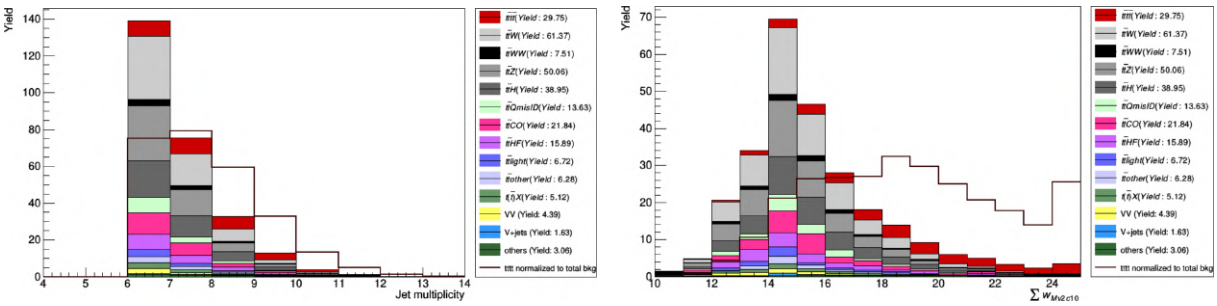
As it has already been mentioned in Section 2.3 our analysis is only concerned with $t\bar{t}t\bar{t}$ events in the SSML channel. This imposes the constraint of having at least 2 leptons with the same sign. In order to further filter the signal region and remove background events, additional cuts are placed. Event are required to have at least 6 reconstructed jets, 2 of which are tagged as b-jets. A cut is also placed on the variable H_T, which is the scalar sum of lepton and jet transverse momenta, to be greater than 500 GeV.



(a) Preselection (b) Signal Region

Figure 4.1: H_T distributions after applying preselection 4.1(a), and after applying signal region cuts 4.1(b). The dark red line shows the signal scaled to the total background yield. The last bin contains all events with $H_T > 2\,000$ GeV [19].

As it can be seen in the figure above, the relative contribution of background events in the signal region is suppressed. These cuts decrease the amount of data from 2.7 million down to 700 thousand events. In the signal region, $t\bar{t}W$, $t\bar{t}Z$, and $t\bar{t}H$ are the most dominant backgrounds.



(a) Jet multiplicity N_j (b) Sum of the b-tagging scores $\sum w_{\mathrm{MV2c10}}$

Figure 4.2: Distributions of jet multiplicity and sum of b-tagging scores in the signal region [19].

4.3 Data Splitting Strategy

The total number of events remaining after the signal selection cuts is 718231, as it can be seen in Table 4.2. For a proper evaluation of the performance of the algorithm, the dataset needs to be split into three portions: training, testing, and validation. The reason behind this will be discussed in Chapter 5.2.

Machine Learning algorithms have a problem with learning from data with negative weights. To combat this issue, the training dataset, which is the dataset from which the algorithm learns, uses only the $t\bar{t}t\bar{t}$ LO dataset, since this dataset only has positive generator weights. Background events with negative weights are ignored for the training dataset.

Sample	Total	Training	Testing	Validation
$t\bar{t}t\bar{t}$ LO	283457	226765	0	56692
$t\bar{t}t\bar{t}$ NLO	301420	0	241136	60284
$t\bar{t}W$	14170	5668	5668	2834
$t\bar{t}WW$	876	350	350	176
$t\bar{t}Z$	63621	25448	25448	12725
$t\bar{t}H$	41043	16417	16417	8209
$V+$ jets	81	32	32	17
VV	1890	756	756	378
$t(\bar{t})X$	926	370	370	186
$t\bar{t}$	2294	918	915	461
Other	8453	3381	3381	1691
Total	718231	280105	294473	143653

Table 4.2: Number of events for each dataset in the signal region.

5

Introduction to Machine Learning

The field of Machine Learning (ML) has been going through a renaissance over the last decade. The progress has been accelerated due to the increased accessibility to computer resources, and the accumulation of large amounts of data.

Machine learning has been an integral part of particle physics for a long time. Methods like Boosted Decision Trees and Support Vector Machines have been utilized for various classification and regression tasks. The recent success of deep learning techniques in computer vision, natural language processing and other fields has compelled scientists to explore the effectiveness of these methods when applied to particle physics data [20].

This chapter introduces the general concept of supervised learning, and describes the operation of various components of a machine learning algorithm. Formal definitions presented in this chapter are taken from Refs. [21, 22].

5.1 Supervised Learning

Supervised learning is concerned with data that can be represented as a set of tuples:

$$\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}, \tag{5.1}$$

where N is the total number of data entries. Each data point is described by two objects:

- **Feature vector x**, where each $x^{(j)}$ for $j = 1, ..., D$ is a description of a particular feature of a data point. For example, if the data describes various countries, $x^{(1)}$ can be used to contain the area in km^2, $x^{(2)}$ for population size, etc. In a feature vector, the j^{th} feature for each data point should always contain the same type of information.
- **Label** y contains information about the class of the data point. It can be a single number or a more complex object. In a binary classification problem, y can be 0 or 1; for a multi-class problem, y can be a vector containing a one-hot encoding[1] representation of classes.

The goal of supervised learning algorithms is to create a model which correctly predicts the labels of the data y_i from input feature vectors $\mathbf{x}_i$. Such an algorithm trains the model by comparing the output to the true values of y_i, and once the model is fully trained it can be used to classify new data whose label is not known.

5.2 Train – Test – Validation Split

The necessity for splitting up data in several parts comes from the need of properly assessing the performance of the model, in order to generalize it to previously unseen datasets. The data internally consists of signal (i.e. the patterns we want to study) and background (i.e. patterns that minimally deviate from the signal). If we train the ML algorithm on the whole dataset, the model will fit the signal as well as background, and we will have no way of generalizing its performance on new data. The process of improving the training performance at the cost of generalization is called **overfitting** and is an undesirable characteristic.

The testing set is a portion of the dataset, set aside during the training. At every iteration, the model runs through the training dataset, predicts output values, and then updates the parameters of the model based on the accuracy of this prediction. We can evaluate the performance of the model after every training step by feeding it with the testing dataset. This dataset is purely used for evaluation, and the model doesn't learn anything from it.

However, there is yet another type of bias we need to account for. After evaluating the performance of the testing dataset, we, as the optimizers of the algorithm, take this information and tune the algorithm accordingly. This introduces a bias of tuning the model to a specific testing dataset, which might result in overfitting. Therefore we need a validation set, a portion of the data set aside from the beginning, and never used for either training or testing. Once the full optimization of the model is complete and the maximum of the testing performance seems to be reached, we can feed the model with the validation dataset, in order to objectively evaluate its performance on a previously unseen dataset.

[1] $c_1 = (1, 0, \ldots, 0)$, $c_2 = (0, 1, \ldots, 0)$, $\cdots$

5.3 Neural Networks

As the name suggests, the creation of Neural Network algorithms was inspired by the desire to formulate a mathematical representation of information processing in the human brain. Our brains have an ability of inductive thinking and pattern recognition, which are a desireable characteristic for a machine learning algorithm.

Neural Networks are efficient models for statistical pattern recognition, because of their ability of modeling non-linear processes. In recent years they have found applications in numerous disciplines, and have fueled recent advancement in software development. This chapter will present the inner workings of a Feed-forward Neural Network (FNN) also referred to as a Multilayer Perceptron (MLP) or simply a Neural Network.

The first layer of the neural network is called the input layer. The size of an input layer is determined by the amount of features in a feature vector $\mathbf{x}$. Each input feature is represented by an input neuron. The last layer of the neural network is called the output layer, and has the characteristic shape of the y labels. For a binary classifier usually the output layer comprises a single neuron. Between the input and the output layers we have so-called hidden layers. A neural network with more than one hidden layer is called a Deep Neural Network.

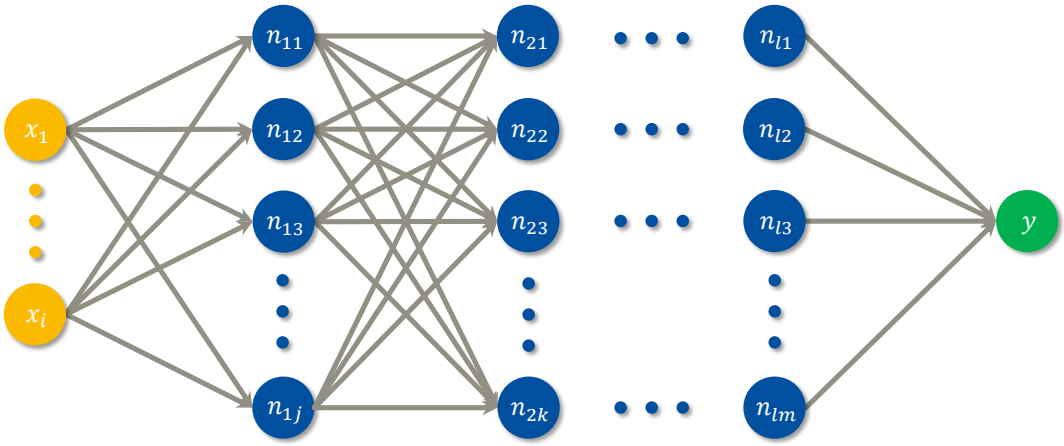


Figure 5.1: Graphical representation of a neural network.

The core building blocks of neural networks are neurons. Each hidden layer of the neural network can be populated with any number of neurons. Neurons take in information from the previous layer, perform mathematical operations, and output a single number, called the *activation* of the neuron. For the purposes of our research we are only going to use fully connected neurons, meaning each neuron is connected with all the neurons of the previous layer. Such a network is called a Dense Neural Network. In order to avoid the loss of clarity for the purposes of generalization, the following shall describe the behavior of a fully connected neural network.

Each neuron operates on its inputs using two parameters:

- **w**, a **weight** vector, describing the sensitivity to each of the neurons in the previous layer. The size of this vector is based on the amount of connections.
- b, the **bias**, a number characteristic of a single neuron.

The activation of a neuron is given by the sum of the inputs multiplied by their respective weights, plus the bias term:

$$a_j = \sum_{i=1}^{D} w_{ji} x_i + b_j. \tag{5.2}$$

In order to introduce non-linearity into our model, activations are fed to a piecewise differentiable, non-linear activation function g:

$$z_j^{(1)} = g(a_j) = g\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + b_j^{(1)}\right). \tag{5.3}$$

Non-linearity helps the model to better fit various types of data and is crucial for generalization. Here, the superscript (1) denotes the number of the hidden layer. z_j is the final output of the j^{th} neuron in the first layer.

All the outputs of the first hidden layer are subsequently fed to the second layer as inputs. Therefore the output of the k^{th} neuron in the second layer is:

$$z_k^{(2)} = g\left(\sum_{j=1}^{D} w_{kj}^{(2)} z_j^{(1)} + b_k^{(2)}\right) = g\left(\sum_{j=1}^{D} w_{kj}^{(2)}\ g\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + b_j^{(1)}\right) + b_k^{(2)}\right). \tag{5.4}$$

Number of neurons and number of hidden layers are among the *hyperparameters* of the model. **Hyperparameters** are fixed quantities governing the model and are used to tune it for specific applications. These values are set in advance by the person conducting analysis, and are kept constant throughout the training process. The weights and biases are free parameters and their values are getting updated as the model trains. The total number of free parameters is governed by the following equation:

$$N = \sum_{l=1}^{L} n_{l-1} n_l + n_l. \tag{5.5}$$

where L is the total amount of hidden layers, and n_l is the number of neurons in the l^{th} layer. This number will quickly explode with the addition of hidden layers, making the training of the model computationally too demanding to handle.

Activation Functions

One of the common examples for an activation function is a **sigmoid**, defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{5.6}$$

This function squishes the input between 0 and 1 as it can be seen on figure 5.2(a), and is usually used for the last layer of the neural network classifier.

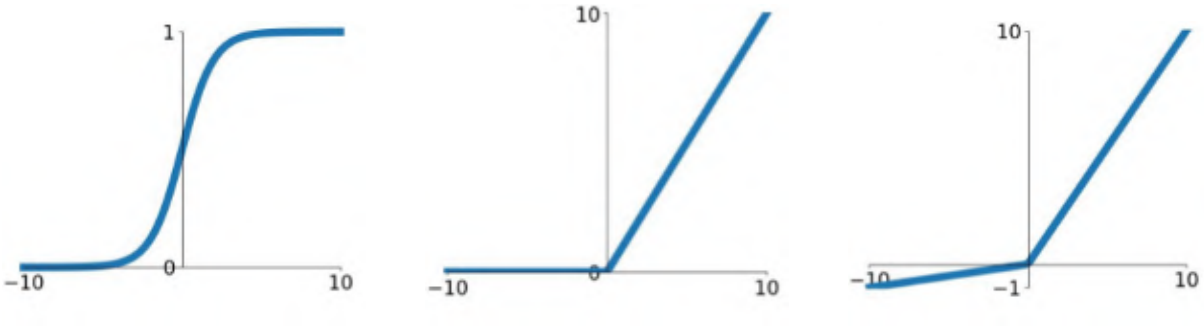
Another commonly used activation function is a **ReLU** (Rectified Linear Unit),

$$\text{ReLU}(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}. \tag{5.7}$$

This function simply nullifies the negative values, and leaves the positive values unchanged. ReLU has the benefit of a reduced computational cost compared to a sigmoid.

Leaky ReLU is a slight modification of the ReLU function. Instead of nullifying the negative values, it scales them with a factor of 0.1.

$$\text{Leaky ReLU}(x) = \begin{cases} 0.1x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}. \tag{5.8}$$



(a) Sigmoid (b) ReLU (c) Leaky ReLU

Figure 5.2: Various activation functions [23].

Weight Initializations

Selecting initial values for the free parameters is another factor that influences the performance of a neural network. Usually these values are drawn randomly from a certain distribution. This research will use the TensorFlow implementation of the **truncated normal distribution** with:

$$\mu = 0 \qquad \sigma = \frac{1}{\sqrt{N}}, \tag{5.9}$$

with the truncation happening at 2 standard deviations away from the mean.

5.4 Building Blocks of a Machine Learning Algorithm

Each machine learning algorithm consists of four main components:

- Model
- Loss function
- Optimization routine
- Performance measure

Since the model has already been discussed in the previous chapter, the remaining three will be explained below.

5.4.1 Loss Function

The loss function, also called a cost function or an objective function, is a function used for evaluating the performance of the model. This function captures the properties of the model and distills them down to a single number. A loss function is a measure of distance between the output and the desired output of the algorithm. The objective of the training is to minimize this value and reach the local minimum of the function.

There are several types of loss functions used in modern machine learning practices. The main distinguishing factor is whether the objective at hand is a classification or a regression.

- **Classification**: process of assigning a class label to an unlabeled data point.
- **Regression**: process of predicting a target value for an unlabeled data point.

Since, the objective of this research is to separate the $t\bar{t}t\bar{t}$ process from the background, we are dealing with a binary classification task. For a loss function our setup will utilize the **Binary Cross-Entropy** also known as the Log loss:

$$L(\theta) = -\frac{1}{N}\sum_{n=1}^{N}\left[y^{(n)}\ln\left(p^{(n)}\right) + \left(1-y^{(n)}\right)\ln\left(1-p^{(n)}\right)\right] \tag{5.10}$$

where, $y^{(n)}$ is the true label (1 for signal and 0 for background) for a specific event, and $p^{(n)}$ is the predicted label (number between 1 and 0) of the same event, i.e.

$$y^{(n)} \in \{0,1\} \qquad \text{and} \qquad p^{(n)} \in [0,1].$$

Each time the model evaluates a true signal event with a probability $p^{(n)}$ we add a $\ln\left(p^{(n)}\right)$ term to the loss, and for each true background we add $\ln\left(1-p^{(n)}\right)$. The further away the predictions are from the true labels, the larger these terms, and vice-versa. This results into an effective penalization of the model by increasing the loss function for misclassification, therefore taking the minimization of this function as the goal of the training, improves the predictive power of the model. Once the parameters of the model are tuned in a way that the loss function is at the local minimum, the model is said to have converged.

Cross-entropy has the benefit of avoiding small gradients, which are important for the optimizer in order to achieve fast convergence of the model.

5.4.2 Optimizer

An optimizer is an algorithm specifically designed for updating the free parameters of the model, in order to improve the performance. After every training epoch, once the performance of the model is evaluated by the loss function, the optimizer takes this value and back-propagates this information by changing the parameters such that the update results into the decrease of the loss value.

Our model utilizes the **Adam** optimization algorithm, proposed in 2015 by D. P. Kingma and J. L. Ba [24]. The name Adam comes from adaptive moment estimation, and stems from the fact that the algorithm estimates moments of the stochastic loss function gradients, in order to adapt the learning rate for each individual parameter. This subsection will concisely summarize the algorithm outlined in the original paper.

Let's start out with our loss function $f(\theta)$, which is differentiable with respect to each parameter θ. After assessing the $f(\theta)$ at a certain timestep t, we can calculate the gradient of the function for each parameter:

$$g_t = \nabla_\theta f_t(\theta). \tag{5.11}$$

Since the gradients are being evaluated for small random batches of data, they will have a stochastic distribution. Adam therefore utilizes an exponential moving average, to calculate the following values:

- m_t, the exponential moving average of the gradient;
- v_t, the exponential moving average of the (elementwise) squared gradient.

The exponential decay rates of these moving averages are controlled by the hyperparameters $\beta_1, \beta_2 \in [0, 1)$ which are by default fixed to $\beta_1 = 0.9$ and $\beta_2 = 0.999$:

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t \qquad v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2 \tag{5.12}$$

m_t is an estimate for the 1^{st} moment of the gradient (mean), and v_t is the estimate for the 2^{nd} raw moment of the gradient (uncentered variance). The values for these variables at timestep $t = 0$ are taken to be the null vectors. This choice introduces an initialization bias, and therefore requires a bias correction. The bias-corrected estimates are calculated the following way:

$$\hat{m}_t = \frac{m_t}{(1 - \beta_1^t)} \qquad \hat{v}_t = \frac{m_t}{(1 - \beta_2^t)}, \tag{5.13}$$

where the exponent t denotes power. Once these values are calculated, the parameters themselves can finally be updated the following way:

$$\theta_t = \theta_{t-1} - \alpha \cdot \frac{\hat{m}_t}{(\sqrt{\hat{v}_t} + \epsilon)}, \tag{5.14}$$

where ϵ is another hyperparameter which by default is fixed to $\epsilon = 10^{-8}$.

The parameter α is the stepsize also called the **learning rate**. This hyperparameter is used to set the effective bound on the stepsize per update. Unlike the hyperparameters β_1, β_2, ϵ, which are usually set to their default values, α depends on the use case and can either be constant, or have a decay schedule as a function of t.

5.4.3 Performance Measures

The output of a binary classifier is a number between 0 and 1, signifying the confidence of the prediction, whether the given event is a signal (1) or a background (0). Because of this, the definition of a discrimination threshold value is required, where all events with the predicted values above this threshold are classified as signal and below this value as background. In order to avoid choosing this threshold value manually a Receiver Operating Characteristic curve, or ROC curve, can be used, which estimates the performance of the model for all possible threshold values, therefore measuring the performance independent of a particular choice of the threshold.

For a binary classifier there are four possible outcome scenarios:

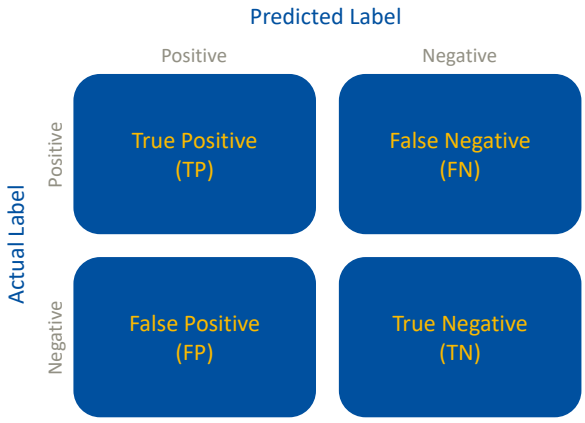


Figure 5.3: Four possible outcome scenarios.

- True Positive (TP): Signal correctly identified as signal
- False Negative (FN): Signal incorrectly identified as background
- False Positive (FP): Background incorrectly identified as signal
- True Negative (TN): Background correctly identified as background

The ROC curve is a plot of a **True Positive Rate** against a **False Positive Rate**, which are defined as following:

- True Positive Rate is defined as the ratio of the number of events correctly identified signal relative to the total amount of signal:

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}. \tag{5.15}$$

- False Positive Rate is defined as the ratio of the number of events incorrectly identified as signal relative to the total amount of background:

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}}. \tag{5.16}$$

Plotting these values for all possible thresholds gives a ROC curve shown below in Figure 5.4. Taking the integral of this curve gives us the Area Under the Curve or AUC. The best possible model AUC is 1, meaning all events are predicted correctly. A random classifier has an AUC Value of 0.5 since it can only correctly guess 50% of the time.

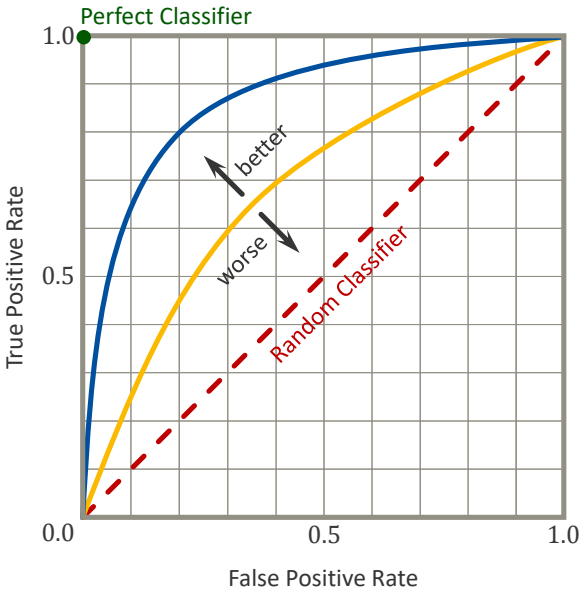


Figure 5.4: Examples for ROC curves.

Another important value to be monitored throughout the research is **overtraining**, which is a measure of the ability of the model to generalize. Overtraining is defined as:

$$O = 1 - \frac{\mathrm{AUC}_{\mathrm{test}}}{\mathrm{AUC}_{\mathrm{train}}}. \tag{5.17}$$

If the performance of the model on the training set is significantly better than on the testing set, overtraining will be closer to 1, which is an indication of the fact that the model is overfitting the training set and because of this losing the ability to generalize. However, if the training and testing values of the AUC are closer, overtraining will be zero. The objective of the optimization of the model is to monitor this value, in order to keep it below 5%.

6

Graph Neural Networks

Most modern deep learning solutions, such as the one outlined in the previous chapter, follow an "end-to-end" design philosophy. These methods try to avoid all kinds of explicit data structures in order to minimize the representational and computational assumptions. Due to the advancement of computational capabilities and increase in the amount of accessible "cheap" data, such methods have been successful in a wide variety of applications.

However, such models tend to struggle with extracting the information from certain types of data with inherent structure. The data used for this analysis contains complex underlying relations and interactions. Most conventional Neural Network architectures require data structured as vectors, grids or sequences, which impose a certain ordering on the data points. For example, particles can be sorted according to their transverse momenta in decreasing order, or based on a different parameter entirely, each choice resulting in a different sequence. Imposing an arbitrary ordering on particles fails to reflect that they are fundamentally unordered.

This analysis will explore the use of Graph Neural Networks, avoiding an artificial ordering of the event data, and employing relational reasoning and combinatorial generalization. Graph Neural Networks are learning functions which operate on mathematical structures of graphs, the sets of elements and their pairwise relations [25]. GNNs are a subfield of a broader field of geometric deep learning. Introducing relational inductive biases can facilitate learning about entities and their relations [26].

For over a decade several new architectures of Neural Networks have been developed, that use mathematical graph structures [27, 28]. Our research uses the Graph Network framework described in [26]. There have been other ongoing efforts concerning the application of GNNs for High Energy Physics analyses [29].

6.1 Definition

This research employs attributed graphs, which are comprised of node-level, edge-level, or graph-level attributes, and are useful for representing event level collider data. In this subsection I will present the formulation of the Graph Network (GN) as outlined in [26]. A graph is defined to be a 3-tuple $G = (\mathbf{u}, V, E)$, where:

- $\mathbf{u}$ is a vector containing the global attributes.
- V is a set containing node vectors $V = \{\mathbf{v}_i\}_{i=1:N^v}$, where N^v is the total number of nodes, and $\mathbf{v}_i$ is a vector describing each node.
- E is a set containing edge vectors $\mathbf{e}_k$ as well as the information about the directionality of each edge: $E = \{(\mathbf{e}_k, r_k, s_k)\}_{k=1:N^e}$, where r_k and s_k are the indices of the receiving and sending nodes respectively.

These three objects allow for the encoding of various types of information. The graph's nodes can describe the entities of the system and the edges can describe the relation between these entities. The global attributes represent system level parameters.

Computations on graphs are performed by a Graph Network block. This algorithm takes the input graph defined above, performs mathematical operations on its elements, and outputs a graph with updated values. The GN block utilizes 6 different functions:

3 update functions:	ϕ^e	ϕ^v	ϕ^u
3 aggregation functions:	$\rho^{e\to v}$	$\rho^{e\to u}$	$\rho^{v\to u}$

The input and the output of the update functions have the same shape. The aggregation functions are order invariant, meaning they can handle the inputs of variable size and output a fixed size representation of the input set. Further details about the inputs of these functions and the sequence of their application will be described below.

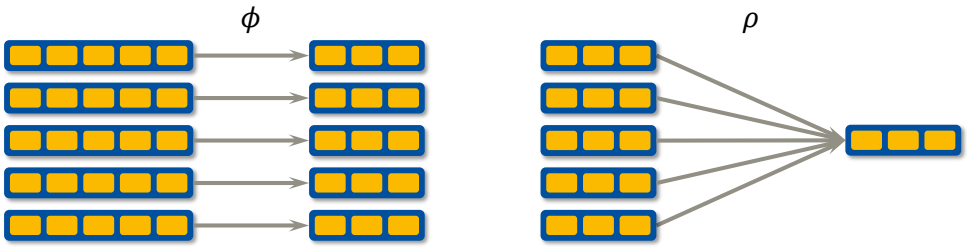
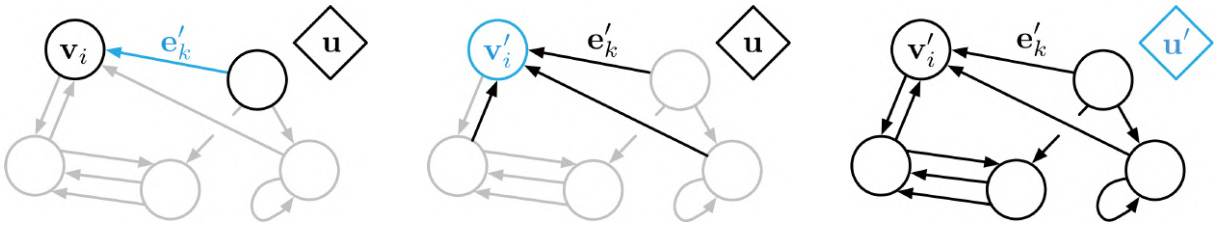


Figure 6.1: Update and aggregation functions.

The sequence of updates in the GN block is as follows:

$$\text{edge} \quad \to \quad \text{node} \quad \to \quad \text{global}$$

Figure 6.2 depicts this process sequentially and shows what information is being used for each update.



(a) Edge Update (b) Node Update (c) Global Update

Figure 6.2: Sequence of updates in the GN block. Elements being updated are represented in blue, while black represents the information used for the update [26].

The algorithm starts out by updating each individual edge with the following function:

$$\mathbf{e}'_k = \phi^e(\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}, \mathbf{u}). \tag{6.1}$$

The function uses the initial edge vector in combination with the node vectors at each end of the edge, as well as the global attribute vector to calculate updated values of the edge vector.

Once all edges are updated the algorithm proceeds to the node update. The first order of business is to aggregate all the incoming edges of the node with the following function:

$$\bar{\mathbf{e}}'_i = \rho^{e\to v}(E'_i), \tag{6.2}$$

where $E'_i = \{(\mathbf{e}'_k, r_k, s_k)\}_{r_k=i,\ k=1:N^e}$ is the set of all edges that the i^{th} node receives. Once the edge aggregate is calculated, we can feed it to the node update function:

$$\mathbf{v}'_i = \phi^v(\bar{\mathbf{e}}'_i, \mathbf{v}_i, \mathbf{u}). \tag{6.3}$$

After calculating the updated vectors for each node individually, the next step is to update the global features. The global feature update requires the aggregation of all edges and nodes of the graph:

$$\bar{\mathbf{e}}' = \rho^{e\to u}(E') \qquad \bar{\mathbf{v}}' = \rho^{v\to u}(V'), \tag{6.4}$$

where, $E' = \{(\mathbf{e}'_k, r_k, s_k)\}_{k=1:N^e}$ is the set of all edges, and $V' = \{\mathbf{v}'\}_{i=1:N^v}$ is the set of all nodes. Using these values we can finally update the global attributes:

$$\mathbf{u}' = \phi^u(\bar{\mathbf{e}}', \bar{\mathbf{v}}', \mathbf{u}). \tag{6.5}$$

As a result of this procedure we get the updated graph $G' = (\mathbf{u}', V', E')$.

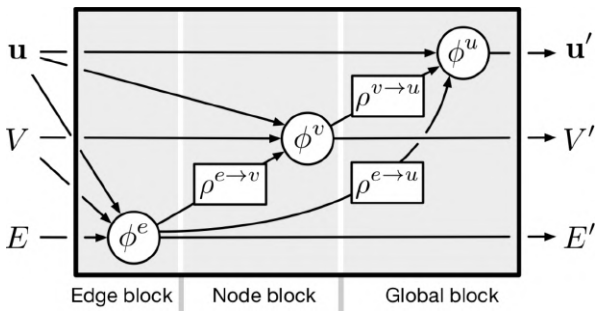


Figure 6.3: Schematic representation of a full GN block [26]

6.2 Experimental Setup

The way event level information is distributed across this structure in our implementation is as follows:

- **Global attributes** encode the event level information, such as the total number of jets, the total sum of b-tagging scores, etc.
- **Nodes** encode the reconstructed objects of the event. We have a node for each jet, each lepton, and one additional node for missing transverse energy. Each individual node contains the object level information, such as momentum, angular information, charge, etc.
- **Edges** encode the angular relation between the above mentioned reconstructed objects. In our model they are automatically generated once the nodes are defined, since the graphs are fully connected, and all edges are bi-directional.

The code developed for conducting our research is split in two parts. The first part takes the `root` files, extracts the information for each event and redistributes the variables across the graph structure. This code also creates the target graphs which are empty except having a single global attribute, populated with either 1 or 0, describing whether the current event is a signal or a background event. Once the input and target graphs are generated for all samples, they are saved as `.dat` files.

The second part of the code, which utilizes the `graph_nets` package by DeepMind [26], imports the graphs from these `.dat` files and feeds them to the Graph Neural Network. The computation proceeds according to the algorithm described above. The update functions in our model are MLPs with the following structure:

$$\begin{aligned} \phi^e &: \quad 256 \times 256 \\ \phi^v &: \quad 256 \times 256 \\ \phi^u &: \quad 512 \times 256 \times 128 \times 64 \times 32 \end{aligned}$$

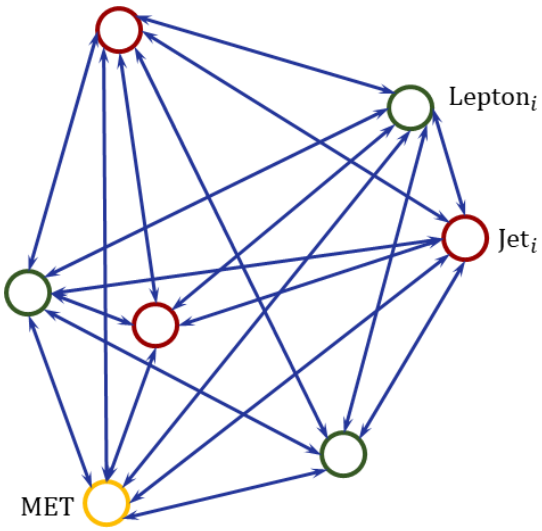


Figure 6.4: GNN Structure.

All three of these MLPs use the `LeakyReLU` as their activation function.

As for the aggregation functions, the model uses the sum over the $\sqrt{N}$ function:

$$output = \frac{1}{\sqrt{N}} \sum inputs, \tag{6.6}$$

where N is the total number of inputs. However, the performance of several other alternatives will be assessed in this research. For the purposes of our application, the GN block should possess permutation equivariance, therefore aggregation functions should be permutation invariant reduction operators.

7

Previous Studies

This chapter briefly summarizes the findings of previous studies performed on the $t\bar{t}t\bar{t}$ event identification. Both of the studies presented below are performed on the same dataset as presented in Chapter 4, and are also concerned with the SSML channel, only.

7.1 BDT Studies

The article published in November of 2020 by the ATLAS Collaboration, provided the first evidence for $t\bar{t}t\bar{t}$ production at LHC with $\sqrt{s} = 13$ TeV pp collisions [30]. The research utilized Boosted Decision Trees (BDT), which are among the most popular machine learning techniques. BDTs are based on a concept of decision trees in combination with gradient boosting. The separating power of the BDT with regards to the $t\bar{t}t\bar{t}$ signal-background separation was measured to be [31]:

$$\text{AUC} = 0.8526 \pm 0.0063. \tag{7.1}$$

7.2 FNN and RNN Studies

$t\bar{t}t\bar{t}$ event classification studies with Feed-forward Neural Networks (FNNs) and Recurrent Neural Networks (RNNs) were performed by Niklas Schwan [19]. The operating principle of FNNs is described in Chapter 5.3. RNNs are another class of artificial Neural Networks which incorporate temporal dynamics [32, 33].

The performance of Feed-forward Neural Networks for the identification of the $t\bar{t}t\bar{t}$ events was measured to be:

$$\text{AUC} = 0.852 \pm 0.005. \tag{7.2}$$

Recurrent Neural Networks underperformed compared to FNNs, with their performance being estimated with:

$$\text{AUC} = 0.838 \pm 0.006. \tag{7.3}$$

8

Results

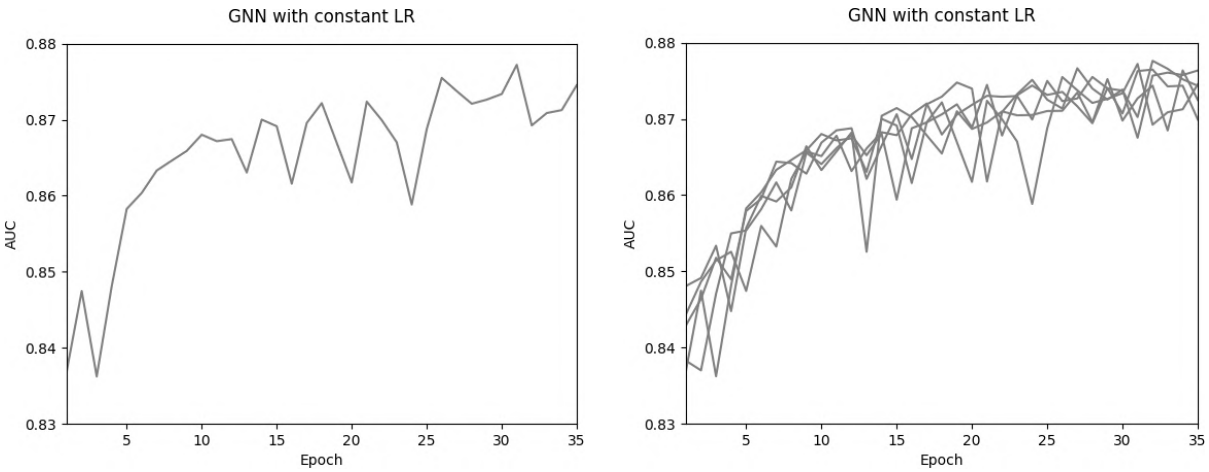
This chapter will present the results obtained in this analysis. Firstly, the necessity of implementing a bootstrapping procedure for a proper performance estimation will be outlined in Section 8.1.

The following three sections will detail the findings of the hyperparameter optimization procedure on the GNNs. Section 8.2 will present the behavior of GNN models for various learning rate values, and provide the reasoning behind the implementation of a learning rate scheduler. Section 8.3 explores the variables used as the global attributes of the graph, and the impact the addition of variables has on the performance of the GNN. Section 8.4 studies five different alternatives for aggregation functions used in a Graph Network block.

Finally, the last section will present the performance of the final model on the validation dataset. The separation efficiency of the GNN will be compared to the performance of the BDT and the FNN, presented in Chapter 7.

8.1 Bootstrapping

The first order of business was to create a stable model that clearly demonstrated the learning behavior. This required the selection of initial hyperparameters based on intuition and previous experience. The models were trained for 35 epochs, since it took approximately 24 hours to complete the training with the available resources. The output of the testing performance of the first model can be seen below in figure 8.1(a). This model clearly demonstrates the learning behavior. Surprisingly, GNNs manage to produce a more competent model from the very first epoch compared to FNNs. In later epochs the performance improves further, and the increase slowly tapers off towards the end of the 35 epochs.



(a) Single run

(b) Five runs

Figure 8.1: The performance of the model with constant learning rate $\alpha = 0.001$.

However, the performance curve contains relatively large oscillations in the model performance from one epoch to the next. These oscillations can be attributed to the internal statistical randomness of the model. The data is being reshuffled after every epoch, changing the order of the events seen by the algorithm. Figure 8.1(b) shows the performance for five models with exactly the same hyperparameters. There are several reasons behind different starting points and fluctuations in these performance curves. Firstly, every time the data is fed to the model, it is getting reshuffled, before being split, meaning the set of events the model trains on is arbitrary and changes for each subsequent run. Secondly, the weight initialization in TensorFlow contains internal randomness, which further influences the performance.

It is common practice in the field of machine learning to factor out such uncertainties by using a *k-Fold Cross-Validation* method. This method requires splitting the dataset into several portions of the same size. One of the portions is then used for testing the performance of the model, while the remaining portions are use for the training. Once the training is complete, the testing portion is replaced, and the models are retrained for each such portion. Thus, if the testing is done with 10% of the data, the model can be trained 10 times, and by averaging the performance of these models, such uncertainties are reduced. Since the testing and training datasets used for our research are of the same size, we can only run the training twice, which is not sufficient to estimate the statistical uncertainties.

To deal with this issue an alternative *bootstrapping* procedure has been developed. This procedure does not resample the events; instead it exploits the internal randomness by training models without fixing the seed that defines the randomness. The dataset is shuffled before starting the training and split in half, meaning every time the model is trained the contents of the training and testing datasets change. Running the model several times, and averaging the output testing performance, gives us a significantly better estimate of the model's performance, as well as providing us with the distributions of the possible model outputs, which we can use to calculate the uncertainties.

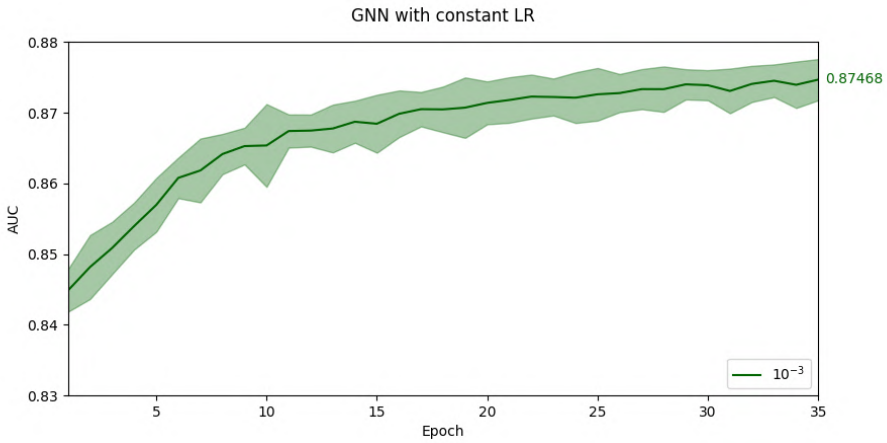


Figure 8.2: Average of the testing performance of 50 models with different randomness. The shaded area depicts the $\pm 1\sigma$ uncertainty around the average.

The result of bootstrapping 50 models can be seen in Figure 8.2. The curve is significantly smoother and a much better representation of the expected model performance is obtained. This clearly demonstrates that the variations in the output performance were the artifacts of internal randomness, and can in principle be completely averaged out with additional runs.

The major disadvantage of this method is a high computation cost. As it has been stated above, each model takes roughly 24 hours to complete the 35 epochs, and the resources allocated for this research allowed for training 10 models in parallel. The bootstrapping procedure was interrupted after 50 models, since there was no significant change detected in the average AUC value or the width of the error bars.

Due to the significant time required for obtaining this result, not all plots presented in this chapter will be bootstrapped equal times. This number will range from 10, when the behavior of the performance curve is clear and further precision is unnecessary to draw a conclusion, to 50, when we need to properly assess the error margins associated with the expected output.

Training models with fixed learning rates has also been attempted for 150 epochs. However, the training usually became unstable beyond 50 epochs and models broke down.

8.2 Learning Rate Optimization

Once the methodology of the performance estimation has been defined, the next challenge is to optimize the hyperparameters of the model. The model presented in the previous section, used the learning rate $\alpha = 0.001$, which was chosen based on previous experience. Figure 8.3 shows the behavior of models for three different values of the learning rates. Models were also trained with learning rate above and below these values, however larger learning rates resulted in unstable models, while the ones with lower values did not learn sufficiently quickly.

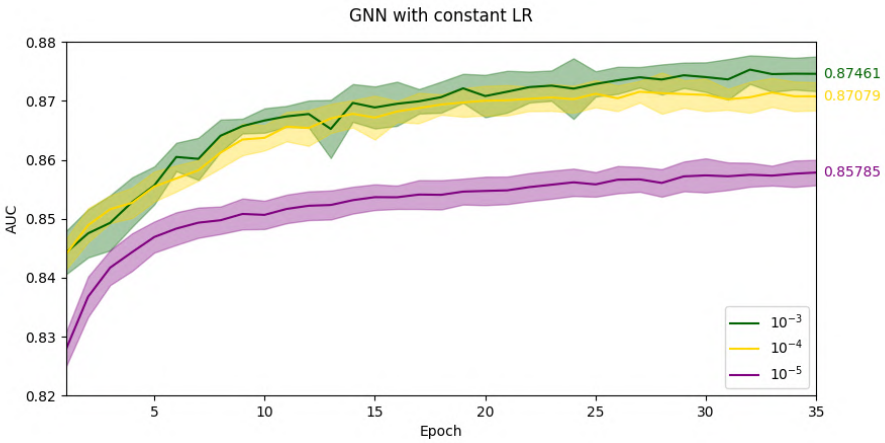


Figure 8.3: Performance curves for different values of the learning rate (×10 bootstrap)

As it can be seen in Figure 8.3, the value of $\alpha = 10^{-3}$ chosen initially displays the optimal performance. Nevertheless, the curves with lower learning rates posses certain desireable features. The purple curve, depicting the model with $\alpha = 10^{-5}$, has a noticeably smoother evolution from one epoch to the next. Because of this, the error bands are smaller, which are expected to grow even wider for larger α values as the results are further bootstrapped.

To take advantage of these benefits a learning rate schedule has been introduced. The scheduler reduces the learning rate by a factor of 10 after every 10 epochs. A comparison of the resulting testing performance curves can be seen in Figure 8.4.

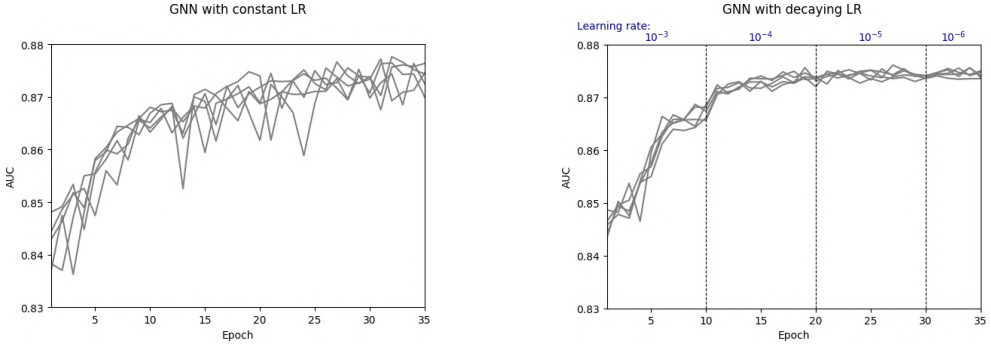


Figure 8.4: Five different outputs for a model with constant LR: $\alpha = 10^{-3}$ (left) vs. a model with piecewise constant decay (right).

This implementation has the benefit of ensuring that the learning rate is sufficiently large during the early stages of the training for the model to learn fast, while reducing it towards the end, to reduce the effect of statistical fluctuations. It is clear from Figure 8.4 that this method significantly reduces the spread of the expected outputs, compared to the model with a constant learning rate.

Figure 8.5 below, shows the average of 50 performance curves. The curve is split in four parts according to the value of the learning rate, which are displayed in blue on top of the plot.

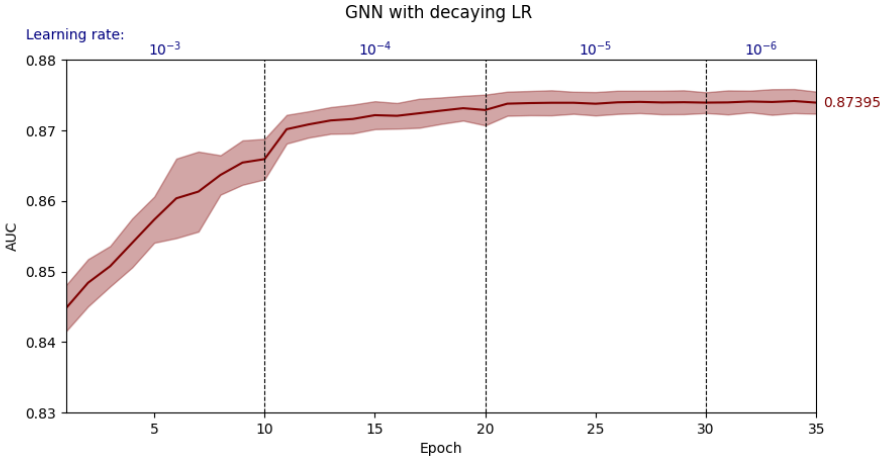


Figure 8.5: Piecewise constant decay (×50 bootstrap).

By comparing this curve to the one in Figure 8.2, it becomes evident that the uncertainties associated with the model output have been significantly reduced, as desired, while maintaining a similar performance.

Checking the behavior of overtraining, demonstrates that similar benefits have been achieved as for the AUCs. Figure 8.6 shows the comparison between the overtraining curves for fixed and decaying learning rates. Error margins are significantly smaller, and the total overtraining after 35 epochs is at an acceptable value.

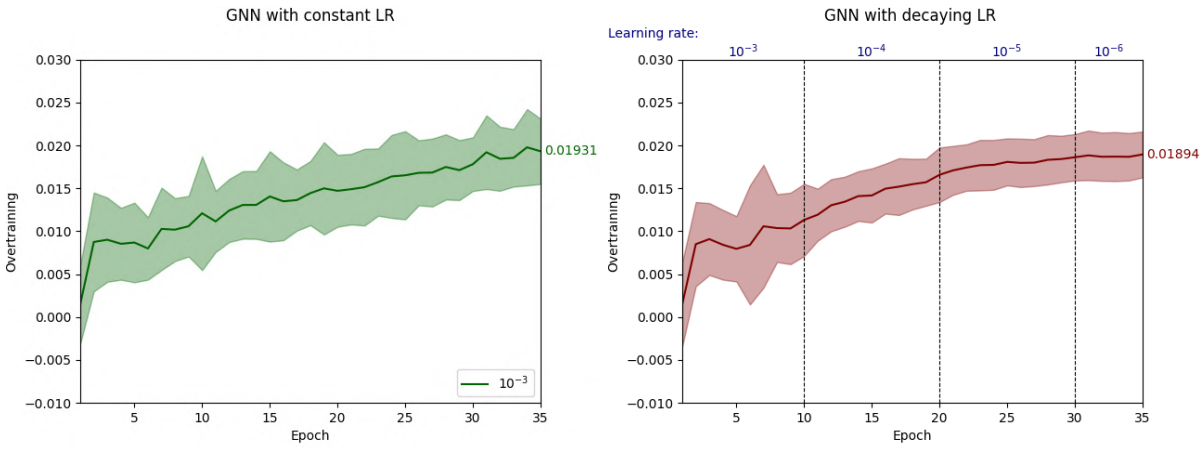


Figure 8.6: Overtraining for fixed (left) and decaying (right) learning rates (×50).

8.3 Variable Optimization

After achieving a desireable outcome with the learning rate optimization, the decision was made to proceed with the optimization of variables used as the global attributes of the model. The models presented above, used three such variables, namely:

- $\sum w_{\mathrm{MV2c10}}$: Sum of b-tagging weights for all jets,
- N_j: Total number of jets in the event,
- H_{T}: Scalar sum of all lepton and jet p_{T}'s.

These event-level variables were chosen, based on a scientific guess, to run initial models. In order to attempt an improvement of performance by adding additional variables, the table from Reference [19], also reported in in Appendix A.1, has been consulted. The table contains the 18 variables with the highest BDT ranking scores. $\sum w_{\mathrm{MV2c10}}$ and N_j are among these 18 variables. Adding the remaining 16 to the 3 used beforehand, results in a model with 19 global variables, the performance of which can be seen in Figure 8.7.

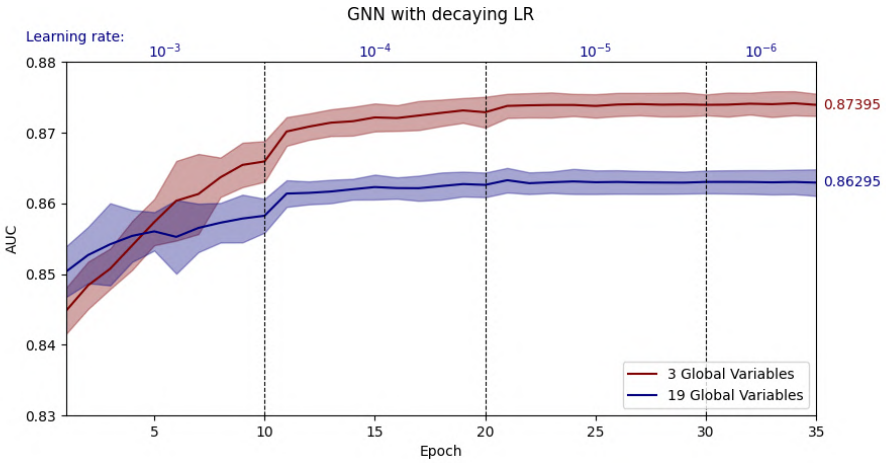


Figure 8.7: Performance curves for 3 vs. 19 global variables (×50 bootstrap).

Surprisingly, providing the GNN algorithm with additional information does not improve the performance. Adding the variables that had the highest separating power for the BDT actually decrease the performance.

In order to explore the behavior of the GNN algorithm for various variable combinations, an iterative addition procedure has been devised. The 16 variables with the highest BDT scores were added one by one to the original three variables, and models were trained for each such configuration. Due to the increase in computation cost and time constraints, the models were only bootstrapped five times. The resulting plot can be seen in Figure 8.8. The plot on the right-hand-side is zoomed on the last five epochs to display the result more clearly. The red curve is the same as the curve in Figure 8.5.

As it turns out, most variables negatively affect the performance individually, and the GNN is unable to extract additional information from them. Only five variables, namely H_{T}^{0}, E_{T}^{miss}, $\Delta R(b,b)_{min}$, $p_{\mathrm{T}}^{(l,0)}$, and $p_{\mathrm{T}}^{(j,0)}$ improve upon the previous results, slightly.

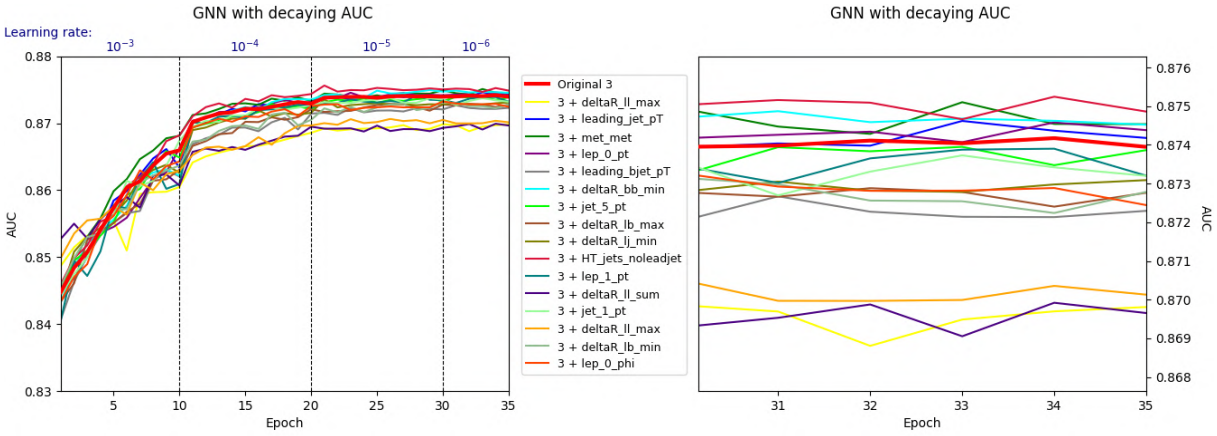


Figure 8.8: Addition of one variable to the original 3 global attributes (×5 bootstrap).

To explore whether these 5 variables interfere constructively, the model was trained with 8 variables (the original 3 plus 5). As it can be seen in Figure 8.9, the model with 8 variables underperforms yet again compared to the original configuration. The green curve displayed below has only been averaged 20 times, since it has been deemed unnecessary to dedicate additional time and resources to this.

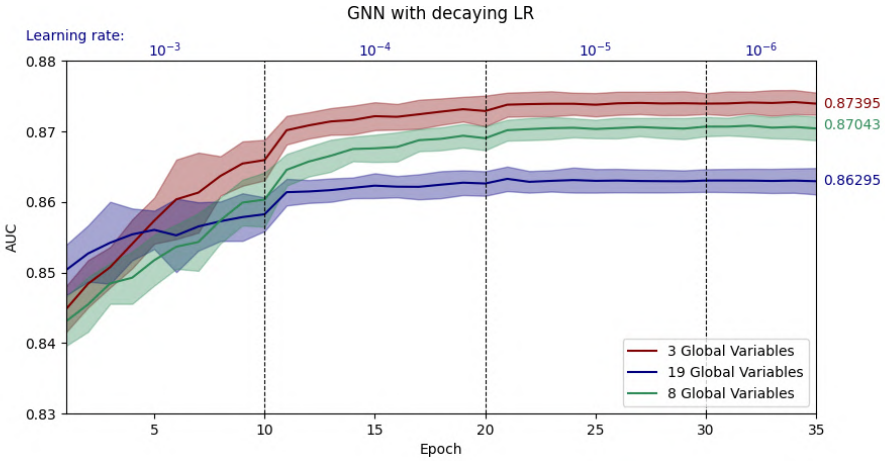


Figure 8.9: Performance curve of the GNN with 8 global attributes (×10 bootstrap) in comparison with the GNNs with 3 and 19 global attributes (×50 bootstrap).

Additional checks have been performed to ensure that the decrease in performance is not due to the sparse distribution of variables. The model with 8 variables was retrained with logarithms of the momentum valued variables, in order to compress these distributions. The resulting outputs can be seen in Appendix A.1. The performance deteriorated even further, indicating that the distribution of variables was not the culprit.

At this point, the variable optimization procedure was terminated, in order to save time, and seek performance gains elsewhere.

8.4 Aggregation Functions

All of the GNN algorithms devised in the previous chapters use (6.6) as the aggregation function, which was chosen based on previous experience as other initial hyperparameters. There are several other options for aggregation functions which are explored in this section:

- `sum`: the sum of the input values,
- `mean`: the mean of the input values,
- `max`: the maximum value among the input values,
- `min`: the minimum value among the input values.

The resulting performance curves for these aggregation functions are displayed in Figure 8.10.

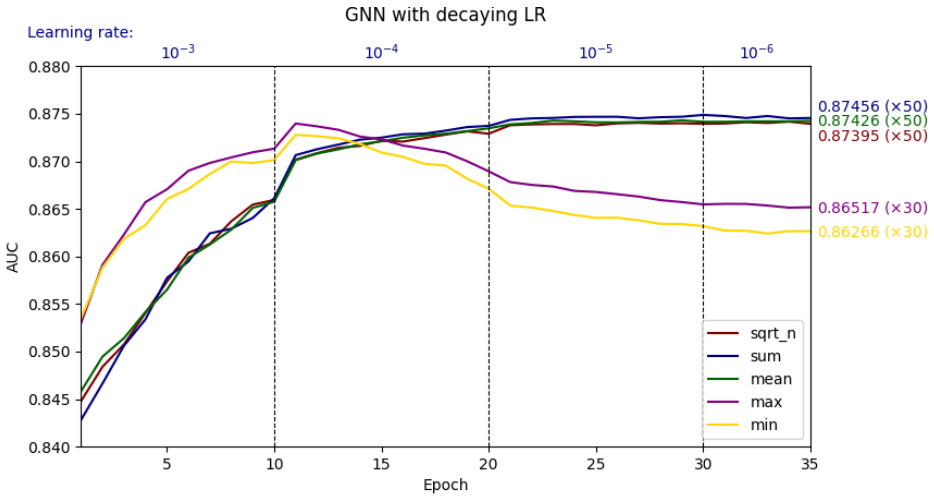


Figure 8.10: Performance curves for five different aggregation functions. Error curves are omitted for clarity. Bootstrapping numbers are indicate next to the curves.

It can be seen that `min` and `max` have a distinct behavior compared to other aggregation functions. Both of these functions achieve the peak performance at the 11^{th} epoch, with their performance gradually decaying at later epochs. As for the behavior of the other three aggregation functions, their performance curves are identical within the error margins, with `sum` having a slightly better performance.

In principle, `sum` should have an advantage compared to other aggregation functions, since the output value of this function is the most sensitive to the variable number of inputs (e.g. the number of nodes in the graph), and because of this it should be able to extract more information from the graphs. Another advantage of this function is that it is computationally less demanding compared to the alternatives, which should have an impact on the total training time. Due to these reasons, it was decided to switch to `sum` as the aggregation function used in our model.

Although, the `max` function might look promising, since it manages to achieve comparable performance in less number of epochs, by taking a closer look at the overtraining curves in the Appendix (Figure A.2), we see that overtraining is significantly higher compared to `sum`.

8.5 Validation

Despite the existence of further potential of performance improvement through optimization, it has been decided to seize the obtained optimization due to time constraints and proceed with the validation of our model.

The final model uses a piecewise constant decay of the learning rate outlined in Chapter 8.2, in combination with `sum` as the aggregation function. The rest of the hyperparameters are kept unchanged as described in Chapter 6.2. The performance curve of this model can be seen in Figure 8.11. The overtraining curve can bee seen in Figure A.3 in the Appendix.

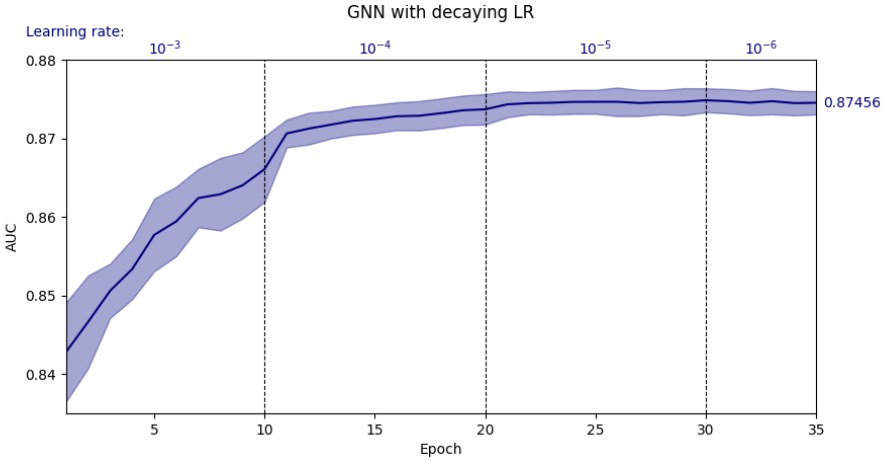


Figure 8.11: Performance curve of the final model (×50 bootstrap)

The total testing performance was evaluated to be:

$$\text{AUC}_{test} = 0.8746 \pm 0.0015. \tag{8.1}$$

The application of these models to the validation dataset resulted in the following performance:

$$\text{AUC}_{Val} = 0.8771 \pm 0.0016. \tag{8.2}$$

This value is noticeably higher then the AUC_{test}, which is counter intuitive. However, the difference can be attributed to the fact that the validation dataset contains $t\bar{t}t\bar{t}$ LO as well as $t\bar{t}t\bar{t}$ NLO events, while the testing dataset only contains $t\bar{t}t\bar{t}$ NLO events as can be seen in Table 4.2. Therefore, it is logical that AUC_{Val} is between the training and testing performance.

These numbers should not be directly compared to the results presented in Chapter 7. The two analyses carried out with BDTs [30, 31] and FNNs [19] used a slightly different data splitting strategy. The only difference from the numbers presented in Table 4.2 is that they used 100% of the $t\bar{t}t\bar{t}$ LO sample for the training. For the sake of consistency, and the ability to make a direct comparison, the 56692 $t\bar{t}t\bar{t}$ events were moved to the training set, and the GNN was yet again trained, tested, and validate on exactly the same datasets as used for the BDTs and FNNs. The resulting performance curve can be seen in Figure 8.12.

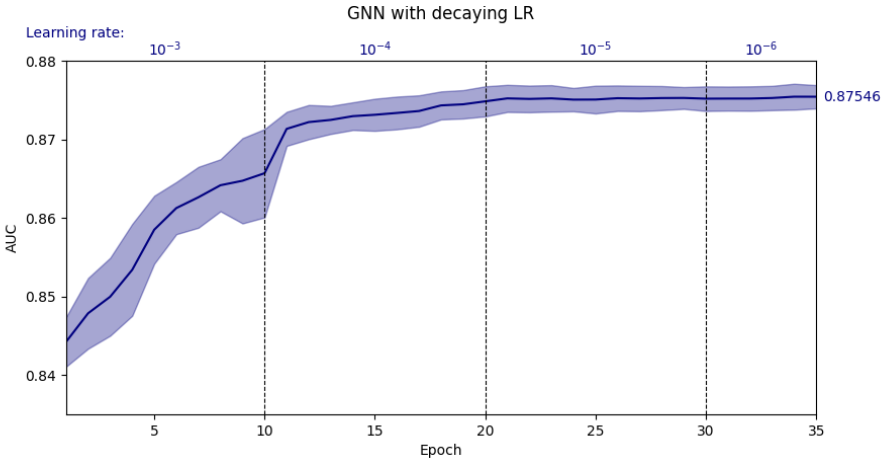


Figure 8.12: Performance curve of the final model with the matching data strategy to the BDT and FNN studies.

As it can be seen in Figure 8.12, the testing performance of the GNN on exactly the same data as used in the BDT and FNN analyses, was evaluated to be:

$$\text{AUC}_{test} = 0.8755 \pm 0.0015. \tag{8.3}$$

This value is slightly larger than the value presented in (8.1), which can be explained by the fact that the model had more data to train on. The validation performance of this model was estimated to be:

$$\text{AUC}_{Val} = 0.8744 \pm 0.0017. \tag{8.4}$$

It is expected, that this value is lower than the one in the (8.2), since the training and validation sets have less similarity, because the validation set does not include a portion of the $t\bar{t}t\bar{t}$ LO sample anymore.

Comparing the result from (8.4) to the results of BDT (7.1) and FNN (7.2), an improvement in performance is evident. The relative increase is roughly $(2.6 \pm 0.7)\%$. This improvement is promising, considering GNNs are not nearly as optimized as BDTs, thus there is still room for further improvement.

9

Conclusion

GNNs are a novel method of machine learning algorithms and their possible applications in high energy physics are yet to be fully explored. The analysis presented in this thesis demonstrates that the introduction of relational inductive bias by representing events as graphs is beneficial for achieving an improved signal-background separation compared to sequential ordering of reconstructed objects.

It is quite remarkable that GNNs manage to outperform most widely used methods of BDTs and FNNs, with so little optimization. There is undoubtedly further room for improvement. The learning rate optimization can be revisited to explore alternative schedules for the learning rate decay. There is no necessity for using the same aggregation function for aggregation nodes, edges, and global attributes, and it might be beneficial to use different functions. It might also be interesting to explore the internal symmetries of the model by redefining angular parameters. A proper variable optimization needs to be conducted for GNNs with regularization, since variables with highest BDT ranking scores do not seem to benefit the performance. Lastly, the alternative NN shapes should be explored for use as update functions.

It should be stated that a large portion of overtraining comes from the the difference between the training and testing datasets. To reduce this effect, the use of $t\bar{t}t\bar{t}$ NLO for training the model should be explored. This should include proper dealing with negative generator weights, or simulating new samples entirely with positive weights. Also the increase in number of events should in principle further improve the performance of the algorithm.

On the analysis side, the potentially different impact of systematic uncertainties between BDTs and GNNs will additionally need to be checked. The calculation of the analysis sensitivity within a full implementation of the statistical analysis using the GNN setup was out of the scope of this thesis, but will be needed for a full optimization of the algorithm.

A

Appendix

A.1 Table of BDT Variables

Feature	Definition	BDT Ranking Score
$\sum w_{\mathrm{MV2c10}}$	Sum of b-tagging weights of MV2c10	0.114
$\Delta R(l,l)_{\mathrm{min}}$	Minimum angular distance between any lepton pair	0.063
$p_{\mathrm{T}}^{j,0}$	p_{T} of the leading jet	0.063
E_{T}^{miss}	Missing transverse energy	0.059
N_j	Jet multiplicity	0.059
$p_{\mathrm{T}}^{l,0}$	p_{T} of the leading lepton	0.059
$p_{\mathrm{T}}^{b,0}$	p_{T} of the leading b-jet	0.058
$\Delta R(b,b)_{\mathrm{min}}$	Minimum angular distance between any b-jet pair	0.055
$p_{\mathrm{T}}^{j,5}$	p_{T} of the sixth highest jet	0.055
$\Delta R(l,b)_{\mathrm{max}}$	Maximum angular distance between any lepton and b-jet	0.054
$\Delta R(l,j)_{\mathrm{min}}$	Minimum angular distance between any lepton and jet	0.051
H_{T}^{0}	Scalar sum of all lepton and jet p_{T}'s excluding the leading jet	0.049
$p_{\mathrm{T}}^{l,1}$	p_{T} of the sub-leading lepton	0.049
$\sum_l \Delta R(l,l)$	Sum of all angular distances between any lepton pair	0.040
$p_{\mathrm{T}}^{j,1}$	p_{T} of the sub-leading jet	0.039
$\Delta R(l,l)_{\mathrm{max}}$	Maximum angular distance between any lepton pair	0.039
$\Delta R(l,b)_{\mathrm{min}}$	Minimum angular distance between any lepton and b-jet	0.039
$\phi^{l,0}$	ϕ of the leading lepton	0.038

Table A.1: The 18 variables with the highest BDT ranking scores [19].

A.2 Additional Plots

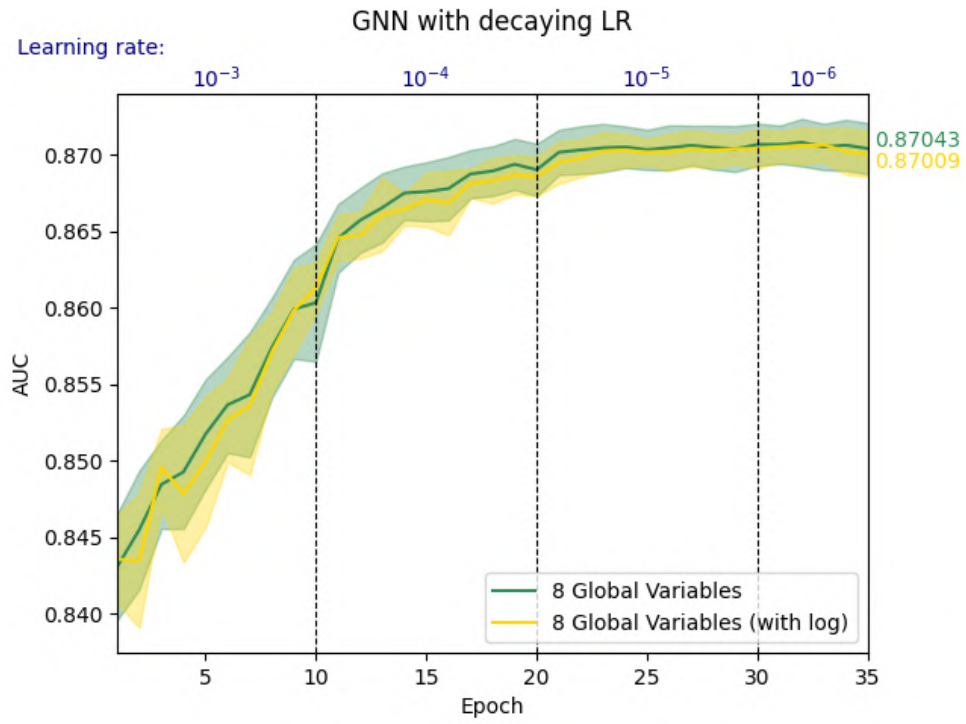


Figure A.1: Performance curves of GNNs with 8 global variables, with and without logarithmic scaling.

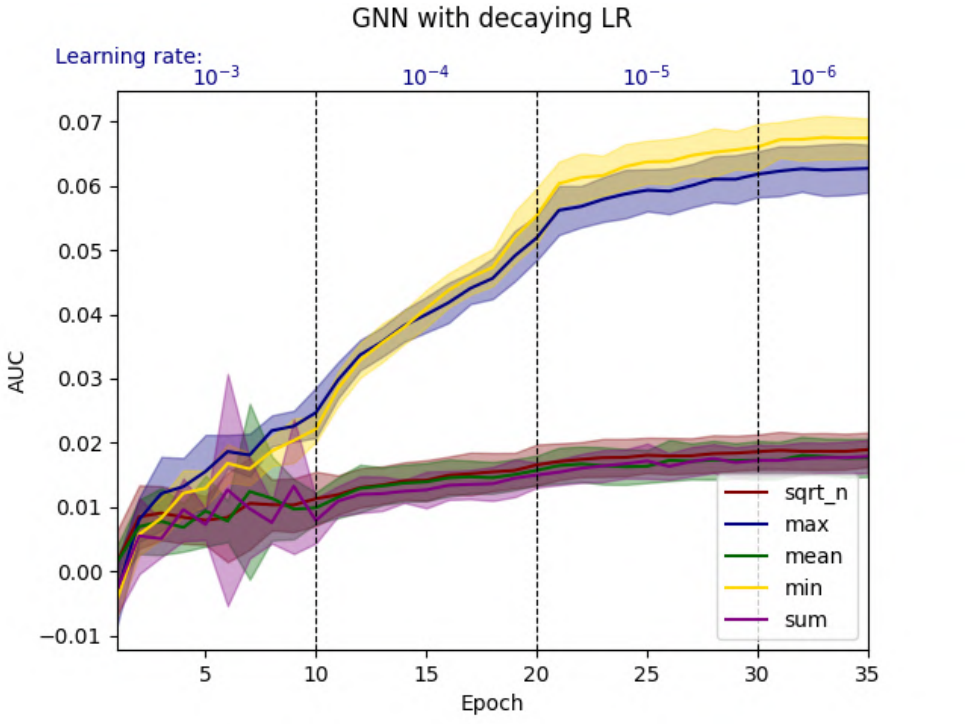


Figure A.2: Overtraining curves for GNNs with five different aggregation functions.

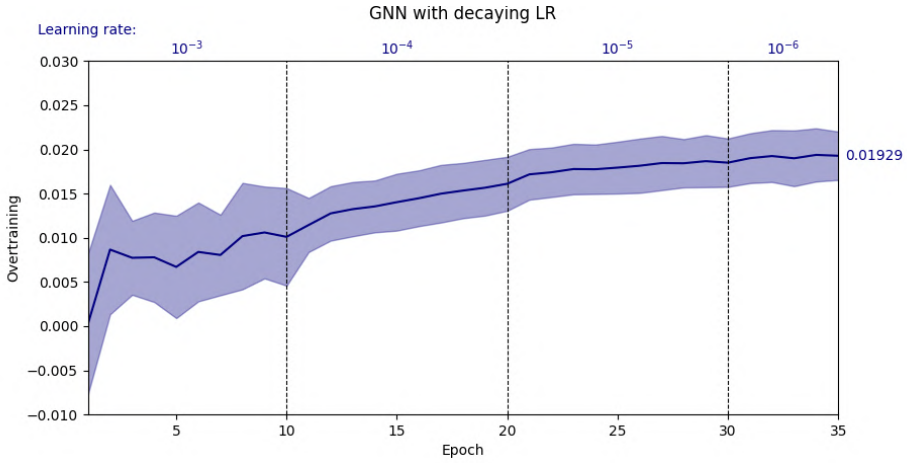


Figure A.3: Overtraining curve of the final model used for validation.

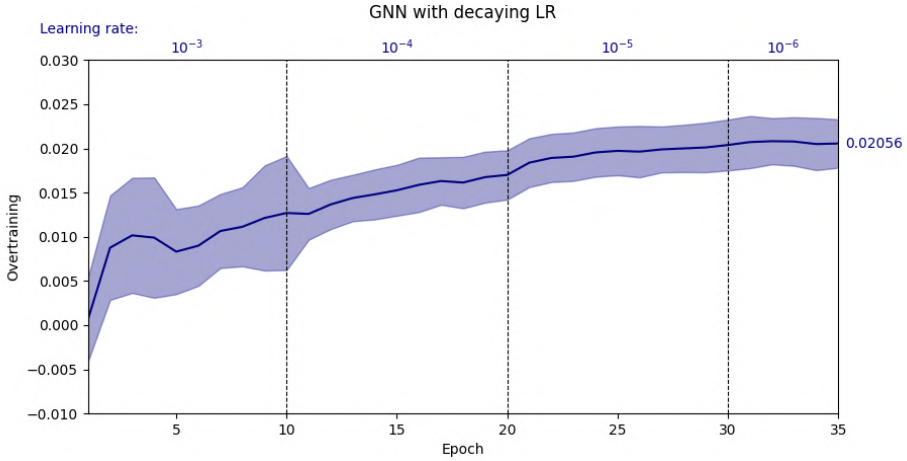


Figure A.4: Overtraining curve of the final model used for validation with the data setup similar to BDTs and FNNs.

Bibliography

[1] P.A. Zyla et al. "Review of Particle Physics". In: *PTEP* 2020.8 (2020), p. 083C01. DOI: 10.1093/ptep/ptaa104.

[2] ATLAS Collaboration. "Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC". In: *Physics Letters B* 716.1 (Sept. 2012), pp. 1–29. ISSN: 0370-2693. DOI: 10.1016/j.physletb.2012.08.020. URL: http://dx.doi.org/10.1016/j.physletb.2012.08.020.

[3] CDF Collaboration. "Observation of Top Quark Production in $\overline{p}p$ Collisions with the Collider Detector at Fermilab". In: *Physical Review Letters* 74 (14 Apr. 1995), pp. 2626–2631. DOI: 10.1103/PhysRevLett.74.2626. URL: https://link.aps.org/doi/10.1103/PhysRevLett.74.2626.

[4] DØ Collaboration. "Observation of the Top Quark". In: *Physical Review Letters* 74 (14 Apr. 1995), pp. 2632–2637. DOI: 10.1103/physrevlett.74.2632. URL: http://dx.doi.org/10.1103/PhysRevLett.74.2632.

[5] CDF Collaboration. "Observation of Electroweak Single Top-Quark Production". In: *Physical Review Letters* 103.9 (Aug. 2009). ISSN: 1079-7114. DOI: 10.1103/physrevlett.103.092002. URL: http://dx.doi.org/10.1103/PhysRevLett.103.092002.

[6] DØ Collaboration. "Observation of Single Top-Quark Production". In: *Physical Review Letters* 103.9 (Aug. 2009). ISSN: 1079-7114. DOI: 10.1103/physrevlett.103.092001. URL: http://dx.doi.org/10.1103/PhysRevLett.103.092001.

[7] Rikkert Frederix, Davide Pagani, and Marco Zaro. "Large NLO corrections in $t\bar{t}W^{\pm}$ and $t\bar{t}t\bar{t}$ hadroproduction from supposedly subleading EW contributions". In: *Journal of High Energy Physics* 2018.2 (Feb. 2018). ISSN: 1029-8479. DOI: 10.1007/jhep02(2018)031. URL: http://dx.doi.org/10.1007/JHEP02(2018)031.

[8] Hans P. Nilles. "Supersymmetry, supergravity and particle physics". In: *Physics Reports* 110.1 (1984), pp. 1–162. ISSN: 0370-1573. DOI: https://doi.org/10.1016/0370-1573(84)90008-5. URL: https://www.sciencedirect.com/science/article/pii/0370157384900085.

[9] Glennys R. Farrar and Pierre Fayet. "Phenomenology of the production, decay, and detection of new hadronic states associated with supersymmetry". In: *Physics Letters B* 76.5 (1978), pp. 575–579. ISSN: 0370-2693. DOI: https://doi.org/10.1016/0370-2693(78)90858-4. URL: https://www.sciencedirect.com/science/article/pii/0370269378908584.

[10] Tilman Plehn and Tim M. P. Tait. "Seeking sgluons". In: 36.7 (Apr. 2009), p. 075001. DOI: 10.1088/0954-3899/36/7/075001. URL: https://doi.org/10.1088/0954-3899/36/7/075001.

[11] Nedaa Asbah et al. *Search for $t\bar{t}t\bar{t}$ Standard Model Production in the Multilepton Final State in proton-proton collisions with the ATLAS Detector*. Tech. rep. Geneva: CERN, Apr. 2019. URL: https://cds.cern.ch/record/2670354.

[12] ATLAS Collaboration. "The ATLAS Experiment at the CERN Large Hadron Collider". In: 3.08 (Aug. 2008), S08003–S08003. DOI: 10.1088/1748-0221/3/08/s08003. URL: https://doi.org/10.1088/1748-0221/3/08/s08003.

[13] Esma Mobs. "The CERN accelerator complex - August 2018. Complexe des accélérateurs du CERN - Août 2018". In: (Aug. 2018). General Photo. URL: https://cds.cern.ch/record/2636343.

[14] João Pequenão. *Computer generated image of the ATLAS inner detector*. 2008.

[15] João Pequenão. *Computer generated image of the ATLAS calorimeter*. 2008.

[16] João Pequenão. *Computer generated image of the ATLAS Muons subsystem*. 2008.

[17] Rodriguez Vera and Ana Maria. *ATLAS Detector Magnet System*. 2021.

[18] João Pequenão and Paul Schaffner. *How ATLAS detects particles: diagram of particle paths in the detector*. Jan. 2013. URL: https://cds.cern.ch/record/1505342.

[19] Niklas W. Schwan. "Improving Four-Top-Quark Event Classification with Deep Learning Techniques using ATLAS Simulation". Presented 2020. 2020. URL: http://cds.cern.ch/record/2751676.

[20] Dimitri Bourilkov. "Machine and deep learning applications in particle physics". In: *International Journal of Modern Physics A* 34.35 (Dec. 2019), p. 1930019. ISSN: 1793-656X. DOI: 10.1142/s0217751x19300199. URL: http://dx.doi.org/10.1142/S0217751X19300199.

[21] Andriy Burkov. *The Hundred-Page Machine Learning Book*. 2019. ISBN: 9781999579517.

[22] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN: 0387310738.

[23] Shruti Jadon. "Introduction to different activation functions for deep learning". In: *Medium, Augmenting Humanity* 16 (2018).

[24] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[25] Jonathan Shlomi, Peter Battaglia, and Jean-Roch Vlimant. "Graph neural networks in particle physics". In: *Machine Learning: Science and Technology* 2.2 (Jan. 2021), p. 021001. ISSN: 2632-2153. DOI: 10.1088/2632-2153/abbf9a. URL: http://dx.doi.org/10.1088/2632-2153/abbf9a.

[26] Peter W. Battaglia et al. *Relational inductive biases, deep learning, and graph networks*. 2018. arXiv: 1806.01261 [cs.LG].

[27] Marco Gori, Gabriele Monfardini, and Franco Scarselli. "A new model for learning in graph domains". In: *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.* 2 (2005), 729–734 vol. 2.

[28] Yujia Li et al. *Gated Graph Sequence Neural Networks*. 2017. arXiv: 1511.05493 [cs.LG].

[29] Shuo Han et al. *Reaching for the Top with GNN*. 2020. URL: https://indico.cern.ch/event/852553/contributions/4057190/attachments/2127774/3582682/Ryan_Roberts_IML_Workshop_211020%20.pdf.

[30] ATLAS Collaboration. "Evidence for $t\bar{t}t\bar{t}$ production in the multilepton final state in proton-proton collisions at $\sqrt{s} = 13$ TeV with the ATLAS detector". In: *Eur. Phys. J. C* 80 (July 2020). DOI: 10.1140/epjc/s10052-020-08509-3. arXiv: 2007.14858. URL: http://cds.cern.ch/record/2725459.

[31] Lennart Rustige. "First evidence of standard model $pp \to t\bar{t}t\bar{t}$ production and performance studies of the ATLAS tile calorimeter for HL-LHC". Thesis. Université Clermont Auvergne ; Universität Dortmund, Oct. 2020. URL: https://tel.archives-ouvertes.fr/tel-03169597.

[32] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". In: *Nature* 323.6088 (Oct. 1986), pp. 533–536. DOI: 10.1038/323533a0.

[33] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. eprint: https://direct.mit.edu/neco/article-pdf/9/8/1735/813796/neco.1997.9.8.1735.pdf. URL: https://doi.org/10.1162/neco.1997.9.8.1735.

Acknowledgements

First and foremost, I would like to thank Prof. Dr. Markus Cristinziani for the opportunity of conducting my research as a member of the ATLAS Collaboration. Prof. Cristinziani was extremely supportive throughout my studies, providing valuable feedback, and sharing knowledge and experience. I am also grateful to Prof. Dr. Florian Bernlochner, for being the second examiner of my master's thesis, and taking time to evaluate my work.

I would like to express my immense gratitude towards PD Dr. Akaki Rusetsky, for granting me the possibility of pursuing my master's degree studies at the University of Bonn, for being a great mentor and always showing support.

I am also very thankful to fellow members of Prof. Cristinziani's group. Dr. Ogul Öncel, was instrumental at the early stages of my research. He was always available for help and was eager to share his vast experience. Niklas Schwan was kind enough to walk me through the research he conducted on a similar topic a year earlier. He shared his code with me, which greatly assisted me in understanding the inner-workings of the Neural Networks. Katharina Voß and Gabriel Gomes helped me with the preparation of my master's thesis colloquium, by giving me valuable feedback.

I am very grateful to Dr. Peter Falke, for introducing me to the Graph Neural Networks and for convincing me to use this Neural Network architecture for my analysis. He guided me throughout my research by providing useful and practical knowledge, and was always available for quick questions as well as insightful discussions.

Lastly, I would like to thank my fellow students, Lado Razmadze and George Chanturia who were by my side throughout my studies, and were always willing to lend a helping hand.