Università Cattolica Del Sacro Cuore
Facoltà di Scienze Matematiche, Fisiche e Naturali

LAUREA MAGISTRALE IN FISICA

Nicola Vanoli

Particle Tracking through Machine Learning

Relatore: Chiar.mo Prof. Daniele Tessera
Correlatore: Chiar.mo Prof. Fausto Borgonovi

Frase

Abstract

In order to find out the composition of our universe, scientists at CERN are colliding elementary particles, essentially recreating mini big bangs, and meticulously recording these collisions with sophisticated silicon detectors. While orchestrating the collisions and observations is already a massive scientific accomplishment, analyzing the enormous amount of data produced from the experiments is getting harder and harder. Given the incredible grow of artificial intelligence and machine learning techniques in the last decade, by using a deep neural network we try to reconstruct particle tracks from 3D points left in these detectors.

Contents

Chapter 1

Introduction

Thanks to the scientific and technological development that took place in recent years, with the possibility of performing increasingly complex calculations in ever shorter times, we have witnessed the rise of new data analysis techniques including machine learning. These artificial techniques have a very wide range of applications, including physics.

In fact, the growing number of data available to researchers is leading them to explore new study techniques: this is what is happening at CERN at the ATLAS experiment. ATLAS explores a range of physics topics, with the primary focus of improving our understanding of the fundamental constituents of matter. Some of the key questions that ATLAS addresses are: What are the basic building blocks of matter? What are the forces that govern their interactions? What was the early universe like and how will it evolve?

Atlas in particular is made up of several silicon detectors, able to detect the passage of charged quantum particles through them. Millions records per second are collected and being able to investigate and extrapolate information from such a huge number of data is not easy. One of the hardest task is in fact reconstruct the trajectory each particle has travelled.

One of the most interesting ways to face this type of problem is using machine learning: we want to build a neural network able to extricate itself in such a dense number of points in order to connect them and reconstruct the trajectory the particles have travelled.

Our goal is achieved through the use of a deep neural network made up of 4 layers and 2200 neurons: in average our best model miscalculates particles' trajectories with an error lower than 5%, demonstrating how artificial intelligence techniques can benefit the world of particle physics.

Chapter 2

Machine Learning

Over the last two decades, with the ever increasing amount of data becoming available it is reasonable to believe that smart data analysis will become a fundamental ingredient for technological progress.

One of the most promising way to achieve this goal is using the Artificial Intelligence (AI), a term that refers to the simulation of human intelligence in machines that are programmed to think like humans and to mimic their actions. The aims of artificial intelligence include learning, reasoning, and perception and its applications are endless: from healthcare industry for dosing drugs and different treatments in patients to self-driving cars, from automated algorithms playing chess at the finest level to the finance world.

Artificial intelligence can be divided in two big groups: weak and strong. Weak AI embodies a system designed to complete a specific task; it includes for example personal assistants such as Amazon's Alexa or Apple's Siri, where you ask the assistant a question and it answers it for you. On the other hand, strong AI includes all algorithms programmed to handle more difficult situations in which they may be required to solve problems without having a person intervene. The most known discipline belonging to this area is the so called *Machine Learning*.

2.1 What is Machine Learning?

The term *machine learning* refers to the automated detection of meaningful patterns in data. Taking example from intelligent beings, many of our skills are acquired or refined through learning from our experience (rather than following explicit instructions given to us). Similarly the basic idea behind Machine Learning is to build a program that can analyze a large amount of data and learn by itself similar patterns, common features and schemes.

The most common Machine Learning technique is the so called *Supervised Learning*: let us make an example to make things clear. Let us suppose that our program is asked to classify different animals (let's say cats, dogs and birds). During the training session the algorithm receives pictures of animal as input; to each picture a label describing the class (in this case "cat","bird" or "dog") is associated. Eventually the model will be able to classify new images of cats, birds and dogs without knowing their label apriori.

2.2 Approximating a Target Function

In supervised learning, a dataset is comprised of inputs and outputs, and the supervised learning algorithm learns how to best map examples of inputs to examples of outputs. We can think of this mapping as being governed by a mathematical function, called the mapping function, and it is this function that a supervised learning algorithm seeks to best approximate.

Neural networks are an example of a supervised learning algorithm and seek to approximate the function represented by your data. This is achieved by calculating the error between the predicted outputs and the expected outputs and minimizing this error during the training process.

The true function that maps inputs to outputs is unknown and is often referred to as the target function. It is the target of the learning process, the function we are trying to approximate using only the data that is available. If we knew the target function, we would not need to approximate it, i.e. we would not need a supervised machine learning algorithm.

Imagine there is a certain unknown function that given a set of parameters as input returns a certain value y, such as

$$y = f^*(\underline{x}) \ , \ \underline{x} \in \mathbb{R}^d$$

A first approximation function could be of the type

$$\tilde{y}(\underline{x}) = \underline{w} \cdot x + b \ , \underline{w} \in \mathbb{R}^d \ , \ b \in \mathbb{R} \tag{2.1}$$

where the vector $\underline{w}$ and the scalar b are the parameters that in principle one can change in order to get the best possible result, i.e. $y = \tilde{y}$.

This method it the so called linear approximation method and can be considered the starting point of much more difficult methods, such as the deep neural networks.

2.3 Evaluation and Optimization

The next step is answering the following question: how well can we approximate a target function? Is there a mathematical metric function that can help us?

In some way the goal is trying to get as close as possible to the point where $y = \tilde{y}$. In order to do that we introduce a **Loss Function** that estimates the distance of our results from the correct values. Given a dataset D, for each data item i we define

$$L(\underline{x}^{(i)}, y^{(i)}) := (\tilde{y}(\underline{x}^{(i)}) - y^{(i)})^2 \tag{2.2}$$

$$L(D) := \frac{1}{N} \sum_{(\underline{x}^{(i)}, y^{(i)}) \in D} L(\underline{x}^{(i)}, y^{(i)}) \tag{2.3}$$

as the Loss Function. In this case the loss function corresponds to the Mean Squared Error (MSE): the closest the loss function is to 0, the better my model is.

Once the loss function as expressed in (2.3) is defined, the whole problem is reduced to an optimization task: in order to obtain the model that best describes the observed phenomenon the Loss Function must be minimized. Let's take eq.(2.1) as example: in this case, for each input vector $\underline{x}$, we obtain an output $\tilde{y}(\underline{x})$. The output depends on the parameters $\underline{w}$ and b that were chosen initially. These parameters are the ones that change during the training process: let us call them θ for now.

The goal of the training process is to find those θ^* that minimize the Loss Function

$$\theta^* := argmin_\Theta L(D, \theta) \tag{2.4}$$

The most common method used to achieve these results is the so called **Gradient Descent (GD)**, an iterative procedure that can be explained as follows:

1. **Initialize $\theta^{(0)}$ at random**: at the beginning of the training process (step 0) the parameters are chosen randomly

2. **Update**

$$\theta^{(t)} = \theta^{(t-1)} - \eta \frac{\partial}{\partial \theta} L(D, \theta^{(t-1)})$$

where

$$\frac{\partial}{\partial \theta} L(D, \theta) := \frac{1}{N} \sum_D \frac{\partial}{\partial \theta} L(\hat{y}^{(i)}, y^{(i)}, \theta) \tag{2.5}$$

At each step the set of new parameters depends on the gradient of the parameters at the previous step.

3. **Unless some termination criterion has been met, go back to step 2.**

The procedure ends whenever some criterion set apriori is reached. In eq.(2.5) the parameter η is fixed and often referred to as **learning rate**. The choice of the learning rate is of fundamental importance for the purpose of the training, as it scales the magnitude of our weight updates in order to minimize the network's loss function.

If your learning rate is set too low, training will progress very slowly as you are making very tiny updates to the weights in your network. However, if your learning rate is set too high, it can cause undesirable divergent behavior in your loss function. Let us give a visualization of how the learning rate choice is crucial:

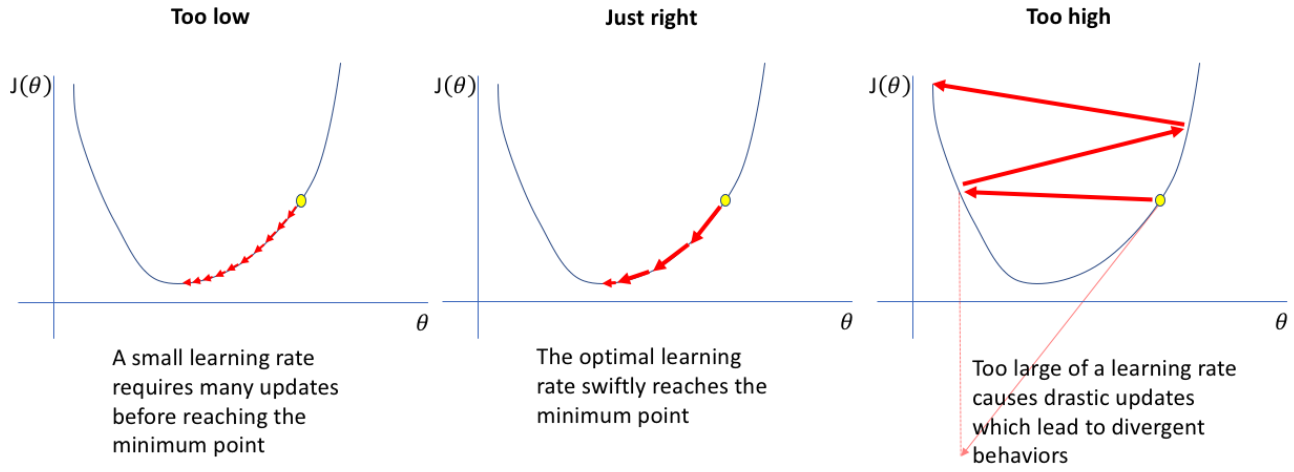


Figure 2.1: Effects of the learning rate's choice on gradient descent method: red arrows represent the various step of the training process. Minimum is reached in computable times only in the middle graph.

It is important to highlight the fact that it does not exist a deafult learning rate valid for all loss functions: finding the optimal learning rate is a difficult task and often requires multiple attempts.

Each observed phenomenon, i.e. the available dataset, will be characterized by a different loss function: if the shape of it was known or easily detectable the training process would be unnecessary. In fact, given the mathematical expression of a certain function, it is only required to compute its derivative in order to find all minima. In real cases, however, this function expressions are not known and the gradient method is the only procedure thanks to which minima can be found.

When setting things up the choice of the initial point is important aswell: although most of the times it is chosen randomly, varying it might bring to very different results. This is due to the fact that the loss function is a multi-dimensional function with many different stationary points: reaching a stationary point of minimum through gradient descent is not sufficient to assert that the global minimum is found.

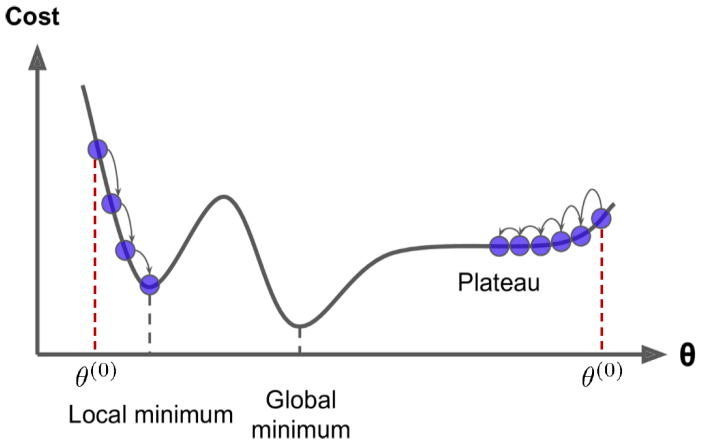


Figure 2.2: Loss in function of parameters θ in one dimension. Purple dots represent the status of gradient descent at each step: depending on the starting point the result may vary and different minima are found.

The optimization process therefore requires many attempts and a bit of luck aswell and although it seems very hard to find the optimal parameters θ^*, there exists a **Convergence Theorem** that asserts the fact that under certain condition the gradient descent method discovers a minimum, as long as the number of steps tends to infinity.

Theorem 1 (Convergence Theorem)**.** *Given a Loss Function L(D,θ) that is convex, derivable and Lipschitz continuous, that is*

$$||\nabla_\theta L(D,\theta_1) - \nabla_\theta L(D,\theta_2)|| \leq C\ ||\theta_1 - \theta_0|| \tag{2.6}$$

the gradient descent method converges to the optimal θ^* for $t \to \infty$ provided that $\eta \leq 1/C$.

When L(D,θ) is derivable and Lipschitz continuous but not convex the gradient descent method converges to a local minimum of L(D,θ) under the same conditions.

2.4 Feed-Forward Neural Network

Once the basic ideas behind **Supervised Learning** are uncovered, we now want to introduce the so called **Feed-Forward Neural Network**, i.e. Deep Neural Networks (DNN). Neural networks are a set of algorithms, modeled loosely after the human brain, that are designed to recognize patterns. They interpret sensory data through a kind of machine perception, labeling or clustering raw input. The patterns they recognize are numerical, contained in vectors, into which all real-world data, be it images, sound, text or time series, must be translated.

Neural Networks consists of a number of simple neuron-like processing units, organized in *layers*. Every unit in a layer is connected with all the units in the previous layer. These connections are not all equal: each connection may have a different strength or *weight*. The weights on these connections encode the knowledge of a network. Often the units in a neural network are also called *nodes*.

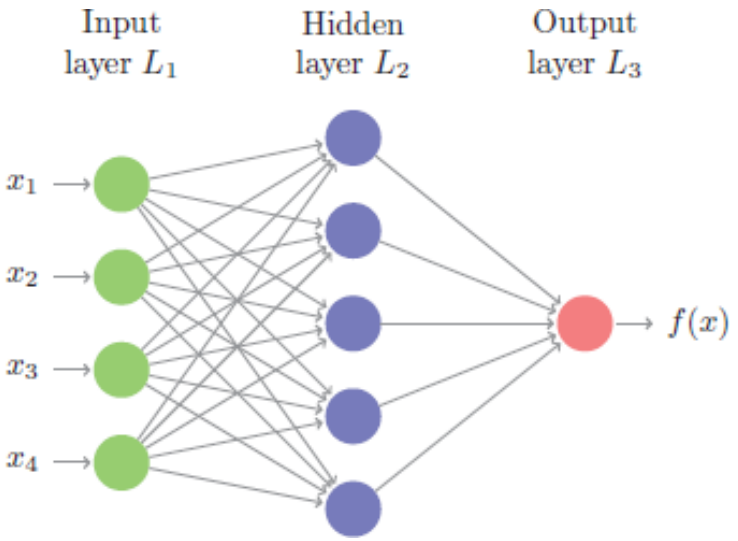


Figure 2.3: Representation of a simple feed-forward neural network. The input $\underline{x} = (x_1, x_2, x_3, x_4)$ goes through the network and the output $f(\underline{x})$ is returned.

Similarly to what happens between neurons in our brains, data enters at the inputs and passes through the network, layer by layer, until it arrives at the outputs. During normal operation, that is when it acts as a classifier, there is no feedback between layers. This is why they are called feed-forward neural networks.

Let us give a mathematical description of these networks starting from Figure 2.3: unlike the simple case described by (2.1), now the input is processed by the network through the neurons of the hidden layer. We can describe the new approximation function as:

$$\tilde{y} = \underline{w} \cdot g(\mathbf{W}\underline{x} + \underline{b}) + b \ , \ \mathbf{W} \in \mathbb{R}^{hxd} \ , \ \underline{w}, \underline{b} \in \mathbb{R}^h \ , \ b \in \mathbb{R} \tag{2.7}$$

$\mathbf{W}$ represents a matrix describing the connections between the input and the hidden layer; in reference to Figure 2.3 $d = 4$ (dimension of the input) and $h = 5$ (dimension of the hidden layer while g is a non-linear function that returns a vector of dimension h and is often referred to as the activation function. An activation function is a function that is added into an artificial neural network in order to help the network learn complex patterns in the data. When comparing with a neuron-based model that is in our brains, the activation function is at the end deciding what is to be fired to the next neuron.

The non-linearity of g is required because most of problems that DNN are asked to solve are simply not linear and since apart from the activation function all computations happening inside the network are linear algebra using a linear activation function would worsen the results. We report here some of the most common activation functions used in *deep learning*.

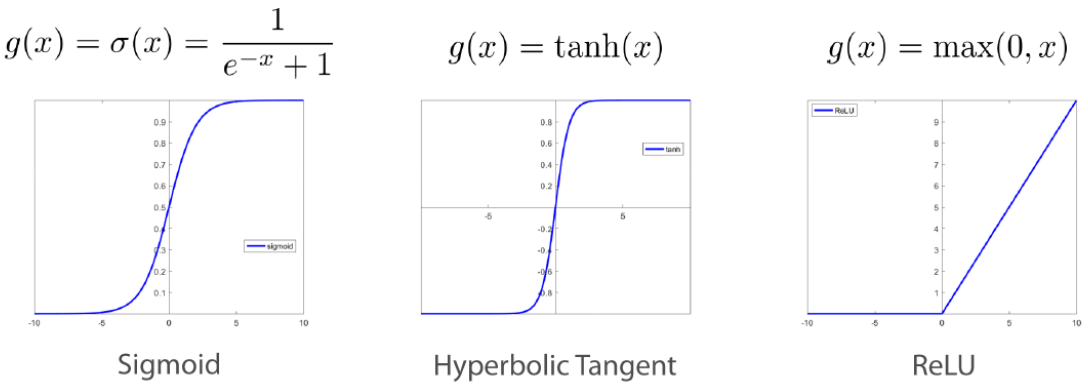


Figure 2.4: Most common choice of g: in addition to adding non-linearity activation function perform a control role avoiding signals to diverge to infinity.

2.5 Universal Approximation Theorem

At this point it is reasonable to answer a legitimate doubt: "Are feed-forward neural networks good approximators? Are they better than the simple ap-

proximation function described in the beginning?". To answer this question, the *Universal Approximation Theorem* comes to our aid.

Theorem 2 (Universal Approximation Theorem)**.** *For any target function* $y = f^*(x)$, $x \in \mathbb{R}^d$ *which is continuous and Borel countable and any* $\epsilon > 0$ *there exist parameters* $h \in \mathbb{Z}^+$, $\mathbf{W} \in \mathbb{R}^{hxd}$, $\underline{w}, \underline{b} \in \mathbb{R}^h$, $b \in \mathbb{R}$ *such thath the feed-forward neural network*

$$\tilde{y} = \underline{w} \cdot g(\mathbf{W}\underline{x} + \underline{b}) + b \tag{2.8}$$

approximates the target function by less than ϵ

$$sup_x |f^*(x) - (\underline{w} \cdot g(\mathbf{W}\underline{x} + \underline{b}) + b)| < \epsilon \tag{2.9}$$

This theorem holds for all g non-linear activation functions discussed above.

This theorem thus suggests that neural networks are suitable for the approximation task. In terms of computational timing and memory usage however, it is often preferable to distribute the neurons between different layers instead of having a single layer containing all neurons. By doing that we build a **deep neural network**, which is the model we used in this work.

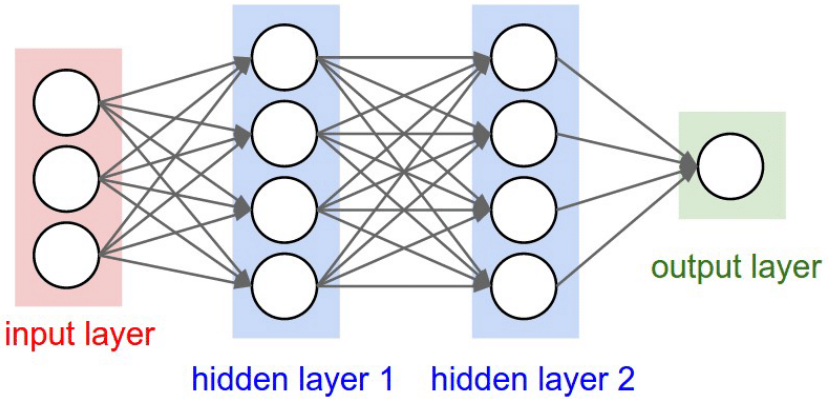


Figure 2.5: Example of deep neural network containing 2 layers.

2.6 DNN approach for 3D points tracking

Before explaining the physical and technical part of the problem we examine in this work we justify here the usage of a DNN. The main goal of this work is to build realistic trajectories performed by elementary particles in silicon detectors. In simple words the dataset is a collection of 3D points, each associated to certain values of energy and momentum. Our goal was building a DNN that is able to distinguish if a segment unifying two points is part of a real trajectory or not.

Chapter 3

The Standard Model

The Standard Model (SM) of particle physics is the best and most sensible theory which summarizes our understanding of the basic components of matter and their interactions in an unified scheme. The fundamental forces described by the theory are the electromagnetic force, the weak force and the strong force. The SM is a relativistic quantum field theory: it combines the fundamental principles of quantum mechanics with those of special relativity. At this moment, gravity is the only fundamental force not described by the SM; there is currently no fully consistent quantum theory of gravity.

3.1 Elementary Particles

Ordinary matter is built up of atoms, with negatively charged electrons attracted to the positively charged nucleus. The electrons are bounded with the nucleus by the electromagnetic force. The nucleus consists of the nucleons: the positively charged protons and the electrically neutral neutrons. These consist of quarks bound together by the strong force: the proton consists of two up (u) quarks and one down (d) quark, while the neutron consists of two d quarks and one u quark. Therefore, an ordinary matter consists only of three elementary matter particles: the electron, the u quark and the d quark. Together with the electron neutrino, the electron and the u and d quarks make up the first generation of the SM matter particles.

These four particles constitute a so-called first generation matter particles. There are also heavier versions of these particles, with exactly the same properties as the first generation particles, except for the mass, making up the second and third generations of matter particles. Each matter particle has its respective antiparticle, which has exactly the same mass, but opposite electric charge. All the elementary matter particles of the SM are spin $-\frac{1}{2}$

neutron has antineutron with 0 charge and opposite baryon number. (Be careful)

	Leptons (ℓ)			Quarks		
	Particle	q_e	Mass [GeV]	Particle	q_e	Mass [GeV]
First	electron ($e^{\pm}$)	$\pm e$	0.0005	down ($d/\bar{d}$)	$\mp\frac{1}{3}e$	0.003
generation	neutrino ($\nu_e/\bar{\nu_e}$)	0	$< 10^{-9}$	up ($u/\bar{u}$)	$\pm\frac{2}{3}e$	0.005
Second	muon ($\mu^{\pm}$)	$\pm e$	0.106	strange ($s/\bar{s}$)	$\mp\frac{1}{3}e$	0.1
generation	neutrino ($\nu_\mu/\bar{\nu_\mu}$)	0	$< 10^{-9}$	charm ($c/\bar{c}$)	$\pm\frac{2}{3}e$	1.3
Third	tau ($\tau^{\pm}$)	$\pm e$	1.78	bottom ($b/\bar{b}$)	$\mp\frac{1}{3}e$	4.5
generation	neutrino ($\nu_\tau/\bar{\nu_\tau}$)	0	$< 10^{-9}$	top ($t/\bar{t}$)	$\pm\frac{2}{3}e$	173

Figure 3.1: The main properties of the Standard Model matter particles.

fermions, i.e. they obey to the Fermi–Dirac statistics.In quantum physics Fermi–Dirac statistics describe a distribution of particles over energy states in systems consisting of many identical particles that obey the Pauli exclusion principle. For a system of identical fermions in thermodynamic equilibrium, the average number of fermions in a single-particle state i is given by a logistic function, or sigmoid function: the Fermi–Dirac (F–D) distribution,

$$\bar{n_i} = \frac{1}{e^{(\epsilon_i - \mu)/k_B T} + 1}$$

where k_B is Boltzmann's constant, T is the absolute temperature, ϵ_i is the energy of the single-particle state i, and μ is the ~~total~~ chemical potential.

The fermionic elementary particle content of the SM is summarized in Table 3.1.The charged leptons are the electrons ($e^{\pm}$), muons ($\mu^{\pm}$), and tau leptons ($\tau^{\pm}$), all of which have electric charge $q_e = \pm e$. For each charged lepton generation, there is one electrically neutral lepton neutrino partner ($\nu_l/\bar{\nu}_l$). The quarks with electric charge $q_e = \pm\frac{2}{3}e$ are the up ($u/\bar{u}$), charm ($c/\bar{c}$), and top ($t/\bar{t}$) quarks, and those with electric charge $q_e = \mp\frac{1}{3}e$ are the down ($d/\bar{d}$), strange ($s/\bar{s}$) and bottom ($b/\bar{b}$) quarks.

In addition to the matter particles the SM introduces force particles, spin-1 gauge bosons, mediating the interactions between them. The electromagnetic (EM) force is mediated by the photon (γ), which is the quantum of EM radiation. The weak force is mediated by the $W^{\pm}$ and Z bosons, while the strong force is mediated by the gluons. While the photon and the gluons are massless, the $W^{\pm}$ and Z bosons are massive, with masses of 80.4 and 91.2 GeV, respectively. The force carrier content of the SM is presented in Table 3.2.

Force	Carrier	Relative strength	Range [m]
Strong	gluons	1	$\sim 10^{-15}$
Electromagnetic	photon (γ)	$\sim 10^{-2}$	∞
Weak	W^+, W^-, Z	$\sim 10^{-13}$	$\sim 10^{-18}$
Gravitational	?	$\sim 10^{-38}$	∞

Figure 3.2: Table of Forces described by the Standard Model.

3.2 Composite Particles

All the leptons in SM can be observed in nature as free particles, as they do not experience the strong force. On the other hand, quarks are confined by the strong force, and they form bound states called mesons (consisting of quark–antiquark pair) and baryons (three-quark states). The mesons and baryons are collectively referred to as hadrons.As an example, the lightest electrically charged meson, π^+, consists of one up quark and one down antiquark. There are also other mesons with heavier quarks involved (charm, strange and bottom), in combination with lighter quarks as well as with each other.

Examples of baryons are the nucleons such as protons and neutrons, composed respectively by the quarks trio *uud* and *udd*. In particular protons consist of quarks bound by gluons, and in a head-on collision between two protons it is the constituent quarks and gluons that collide. Inside each proton you can find a "sea" of quarks and gluons. This is due to the fact that although the valence quarks are 3, there are a bunch of virtual quarks and anti-quarks stemming from gluons. When protons collide with such large energies as at the LHC, the collision results in a shower of all types of particles, the ones usual matter is made of, and others that only existed just after the Big Bang.

3.3 Conservation Laws

In developing the standard model for particles, certain types of interactions and decays are observed to be common and others seem to be forbidden. The study of interactions has led to a number of conservation laws which govern them. These conservation laws are in addition to the classical conservation laws such as conservation of energy, charge, etc., which still apply in the realm of particle interactions.

One of the most important of these is the conservation of baryon number. Each of the baryons is assigned a baryon number $B = 1$. This can be considered to be equivalent to assigning each quark a baryon number of 1/3. This implies that the mesons, with one quark and one antiquark, have a baryon number B=0. No known decay process or interaction in nature changes the net baryon number. The neutron and all heavier baryons decay directly to protons or eventually form protons, the proton being the least massive baryon. This implies that the proton has nowhere to go without violating the conservation of baryon number, so if the conservation of baryon number holds exactly, the proton is completely stable against decay. One prediction of grand unification of forces is that the proton would have the possibility of decay, so that possibility is being investigated experimentally.

Whenever two protons collide many particles product can be obtained, depending on the initial conditions. For this reason experiments such as ATLAS try to measure all of the particles which are produced and reconstruct what was created in the initial interaction. We report in Figure 3.3 an example of a particle shower, i.e. is a cascade of secondary particles produced as the result of a high-energy particle interacting (in this case protons).

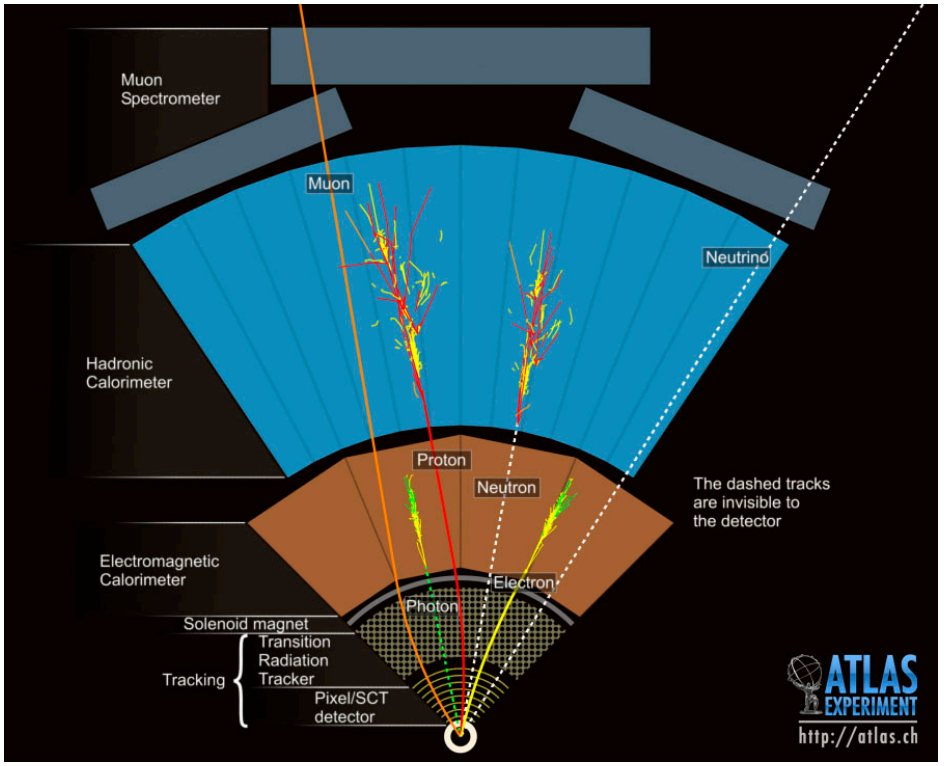


Figure 3.3: Schematic example of a particle shower observed at ATLAS experiment. The bottom centered point is the collision point: all generated particles move away from it.

Chapter 4

Experimental Setup

Considering that while doing any kind of data analysis an essential prerogative is to know the data you are dealing with we introduce here the experiment that produced our dataset.

In the previous century physics a drastic revolution which brought to light new theories that aim to describe our universe. Although the Standard Model is nowadays well established in the field of physics and has precisely predicted a wide variety of phenomena and so far successfully explained almost all experimental results in particle physics, many questions still have to be answered. Between these we report here the most important:

- What is the origin of mass?
- Will we discover evidence for supersymmetry?
- Why is there far more matter than antimatter in the universe?

In order to try to answer these questions the Large Hadron Collider(LHC), the world's largest and highest-energy particle collider and the largest machine in the world was built by the European Organization for Nuclear Research (CERN) between 1998 and 2008 in collaboration with over 10,000 scientists and hundreds of universities and laboratories all around the world.

4.1 LHC

The LHC is a particle accelerator that pushes protons or ions to near the speed of light. It consists of a 27-kilometre ring of superconducting magnets with a number of accelerating structures that boost the energy of the particles along the way. The accelerator sits in a tunnel 100 metres underground

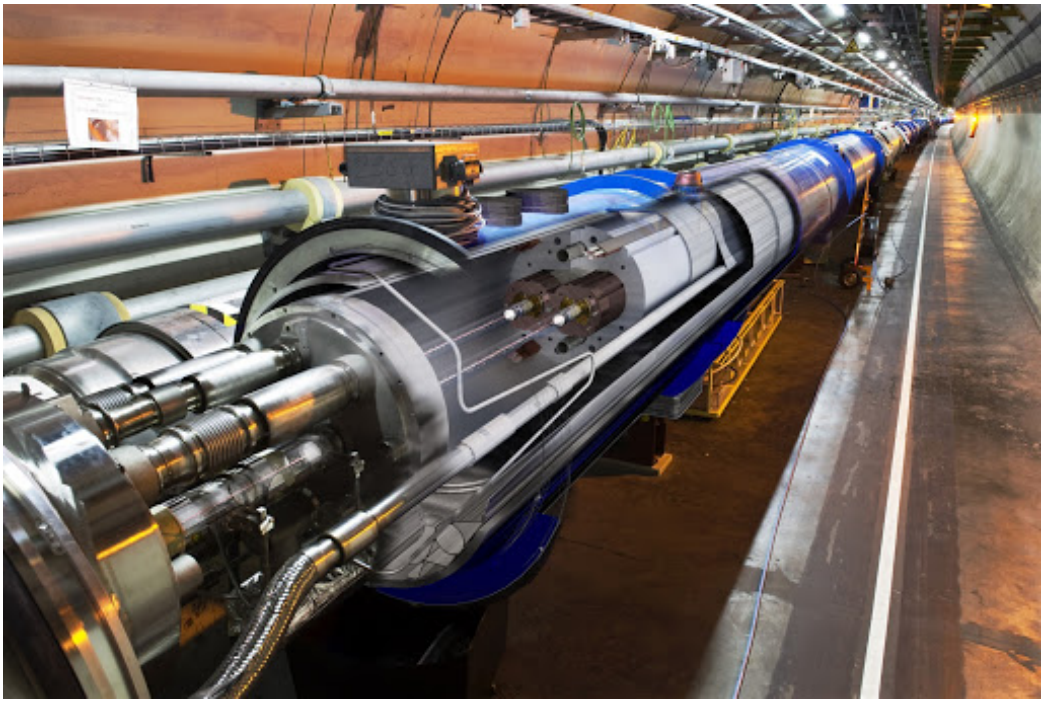


Figure 4.1: Section of the LHC. The two adjacent beams running in opposite directions can be seen in the middle of the picture.

at CERN, on the Franco-Swiss border near Geneva, Switzerland. The collider tunnel contains two adjacent parallel beamlines (or beam pipes) each containing a beam, which travel in opposite directions around the ring. The beams intersect at four points (that are the so called *experiments*) around the ring, which is where the particle collisions take place.

A schematic view of the LHC ring and its pre-accelerator complex is shown in Figure 4.2.In the first step, protons are extracted from a hydrogen source and injected into the linear accelerator LINAC, where they are accelerated up to an energy of 50 MeV. Next, three circular accelerators are used to further increase the energy: first, the Proton Synchroton Booster allow to accelerate the particles up to 1.4 GeV. It is followed by the ProtonSynchroton (PS), which creates 72 particles bunches with a length of 4 ns and time spacing of 25 ns. In the PS, the protons are further accelerated to 25 GeV. Before the particles are injected into the LHC ring, the Super Proton Synchroton (SPS) combines six fillings from the PS into 216 proton bunches and increases the energy to 450 GeV in the last step.

Experiments take place in four different location, visible on Figure 4.2: ATLAS, ALICE, CMS and LHCb. The dataset analyzed in this thesis was

recorded by the ATLAS experiment.

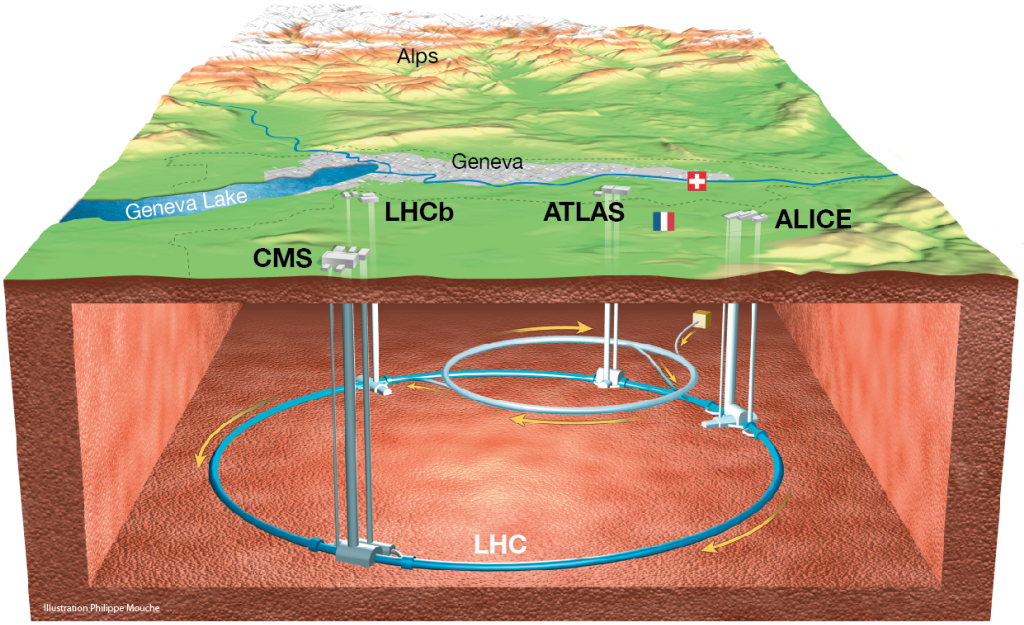


Figure 4.2: Schematic view of the LHC accelerator ring and the location of the four main experiments ATLAS, CMS, ALICE and LHCb.

4.2 The ATLAS Experiment

The ATLAS detector is a multi-purpose experiment, which is designed to explore a wide range of physics processes at high collision energies. It enables the search for new physics phenomena as well as precision measurements of elementary particles and their interactions. The overall structure of the ATLAS detector is illustrated in Figure 4.3. Its cylindrical design covers almost all space around the axis and the high-density detector material enables the precise reconstruction of hard scattering events up to low scattering angles.

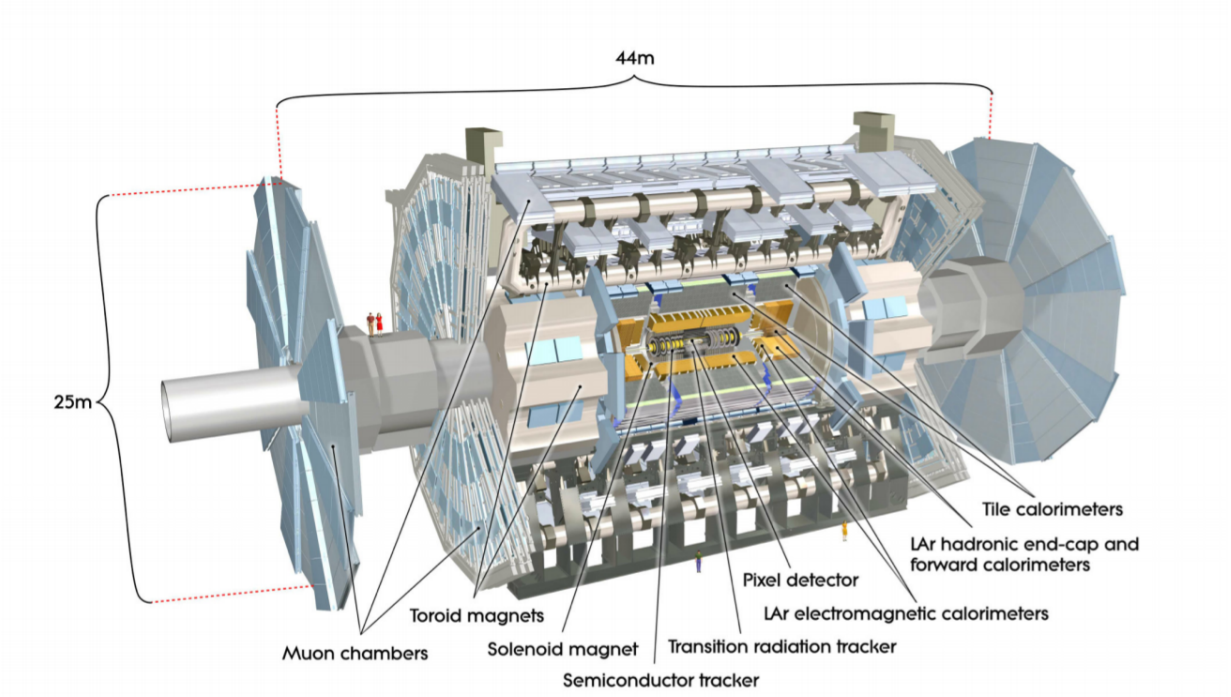


Figure 4.3: Overview of the ATLAS detector showing the muon system, the toroidal and solenoid magnets, the calorimeters and the inner detector.

The ATLAS experiment consist of several sub-detectors, which are arranged in a cylindrical symmetric way around the beam pipe. The inner detector is build closest to the collision point and is embedded in a solenoid magnetic field with a field strength of up to 2 T. Around the inner detector, the electromagnetic and hadronic calorimeters are installed. The outermost system is given by the muon spectrometer, which is surrounded by three large superconducting toroids with magnetic field strengths between 0.5 T and 1 T.

4.3 Inner Detector

The inner detector (ID) of the ATLAS experiment measures the trajectory of charged particles with high precision. It allows for the reconstruction of primary and secondary vertex positions and momenta as well as the identification of the particle charge. For this, the detector has to be constructed to cope with high occupancies and radiation doses and a large data rate.

As shown in Figure 3.3, the ID consists of the pixel detector with the

insertable bLayer (IBL), the semiconducting tracker (SCT) and the transition radiation tracker (TRT).

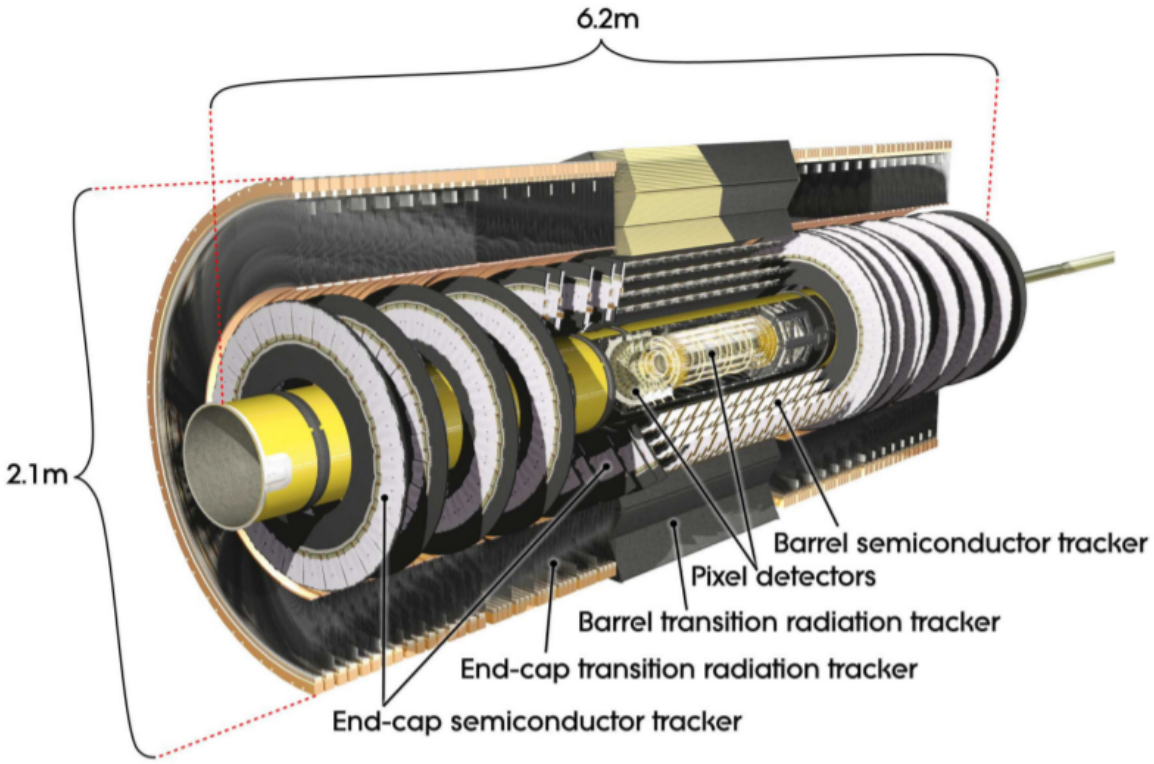


Figure 4.4: Schematic view of the ATLAS inner detector with the pixel detector, the semiconductor tracker and the transition radiation tracker.

The pixel detector is surrounded by the semiconductor tracker (SCT) covering a radial region between $30cm - 52cm$. It consists of four double-sided layers of silicon strip detectors in the barrel region and nine double layers in each endcap region. In each layer, the strip modules are made of two $6.4cm$ long daisy-chained sensors with a strip pitch of $80\mu m$. Charged particles cross at least eight strip layers when traversing the SCT. Due to the double-layer structure of the strips, this results in the measurement of four space points in the barrel region. In total, 258 active strips and 6.3 million readout channels are used.

Further information on what type of data and how the data are collected can be found in the next chapter.

4.4 Pixel Detector

Let us now undercover the physics behind the pixel detectors: understanding their working mechanism plays a fundamental role for the purpose of this thesis.

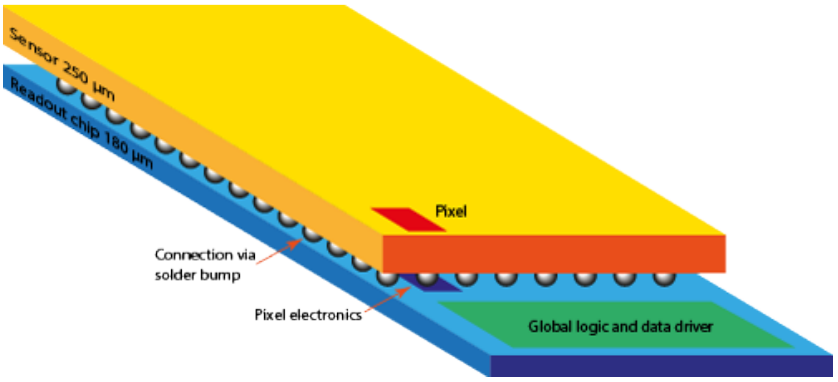


Figure 4.5: Schematic view of a pixel detector inside the ATLAS experiment. The pixel detectors are used to track the trajectories of the particles.

Figure 4.5 represent a schematic visualization of a pixel detector: the top yellow layer of silicon is connected to a lower layer of electronics through a large array of solder spheres (each sphere has a diameter in the order of $10^1 \mu m$). To understand what happens in the silicon when a charged particle punches through, we have to zoom to the scale of the silicon molecules, i.e. $10^{-10} m$. As a charged particle goes through the silicon layer and encounter silicon molecules, it liberates electrons. These electron move to the bottom of the strip creating an electric current through one or more spheres. By seeing which spheres had a signal we can tell where the original particle went. The signal is then converted into binary numbers and stored inside a dataset similar to the one we use.

4.5 Proton-Proton Collisions

Last but not least, we now want to give a theoretical overview of what happens whenever 2 protons collide at the ATLAS experiment. A proton is a stable subatomic particle that has a positive charge equal in magnitude to a unit of electron charge and a rest mass of $1.67262 \times 10^{-27} kg$, which is 1,836 times the mass of an electron. Protons consist of 2 *up* and 1 *down* quarks bound by gluons, and in a head-on collision between two protons it is the constituent quarks and gluons that collide.

The LHC provides a steady stream of protons, half going in one direction, and half going in the other direction around the ring. They're actually in bunches. One way to think about these bunches is to pretend that they're just like enormous swarms of mosquitos. The bunches themselves are tiny but we can cross the beams just right such that bunches pass through one another. But that's still not enough to ensure collisions since most protons in the bunches will simply fly right past each other. The biggest issue in this type of collision consists in the fact that there's a huge amount of empty space in there, so they seem close but on a small scale they're really still incredibly far away from one another.

Of course even though most protons fly by each other, collisions do, from time to time, occur. And since we have $\sim$ 40 milions bunch crossings per second and $\sim 10^{11}$ protons in each bunch, it actually adds up to a huge number of collisions. The term collision is actually misleading: in the case of sub-atomic particles Nothing is really 'colliding' in the way you would imagine things colliding in our macroscopic world. A better term to describe this phenomenon would be interaction.

Depending on the initial energies of the colliding protons many different processes can occur. Each process leads to different products and has a specific probability to occur. The job of the physicist is to search for the interesting events leading to interesting outcomes and to filter out the less interesting ones. Since this is not an easy task, A.I. algorithms might help solving this problem.

Chapter 5

Track Particle Dataset

Before starting analyzing data, it is a good habit to explore and understand the type of data you are dealing with. For this thesis, we took inspiration from a competition promoted by **Kaggle** [1]. Kaggle is an online community of data scientists and machine learning practitioners. Kaggle allows users to find and publish data sets, explore and build models in a web-based data-science environment, work with other data scientists and machine learning engineers, and enter competitions to solve data science challenges.

In 2018, Kaggle partenered with CERN and published a competition called TrackML Particle Tracking Challenge, whose purpose was to challenge data scientist from all around the world to build an algorithm that quickly reconstructs particle tracks from 3D points left in the silicon detectors.

A dataset comprises multiple independent events, where each event contains simulated measurements (essentially 3D points) of particles generated in a collision between proton bunches at the ATLAS experiment. The goal of the tracking machine learning challenge is to group the recorded measurements or hits for each event into tracks, sets of hits that belong to the same initial particle.

5.1 Exploratory Data Analysis

The full training dataset contains 8850 events, while the test dataset is made up of 125 events. Because of the large dimensions of each event file, we only used a small part of these 8850 events to train our neural network. To give an essential idea of what task the partecipants are asked to solve we report here an image of the detected points with some of the trajectories described by the particles.

[1] https://www.kaggle.com

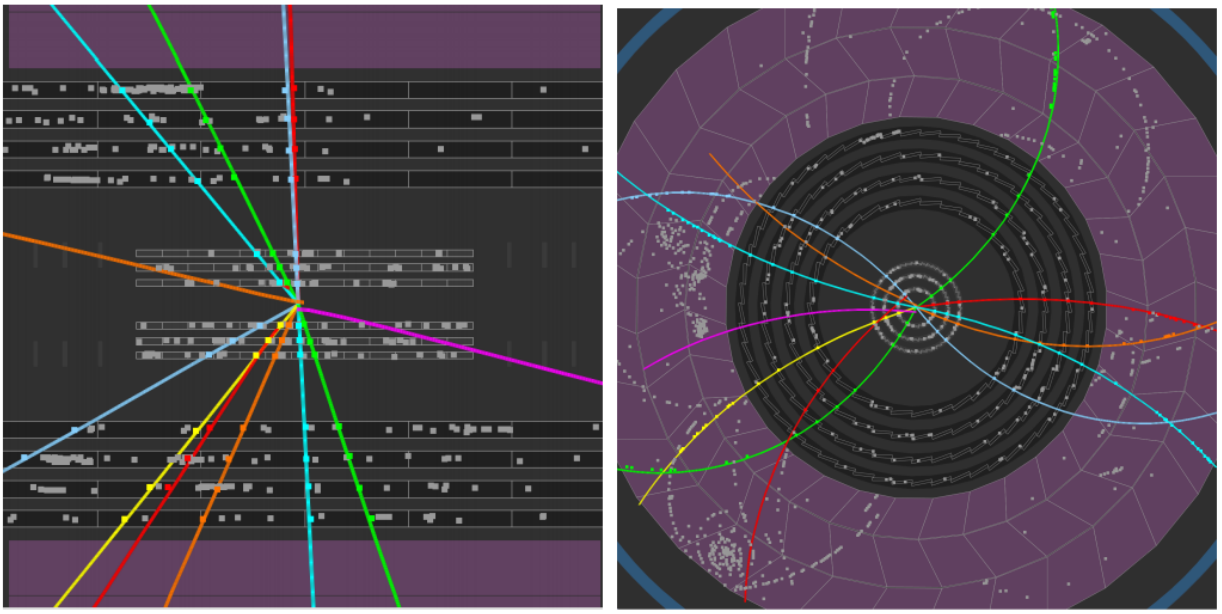


Figure 5.1: Projection of the tracks in the longitudinal and transverse planes, for low multiplicity events in the current detector.

In Figure 5.1 white points represents a measurement recorded by the silicon detectors: after the collision between protons(happening in the middle of the figures) many sub-atomic particles are generated. Each of them moves away from the center with different momentum and direction.

Whenever each particle crosses a silicon layer a white point is recorded. Colored lines represent reconstructed trajectories of some of these particles. The goal of this thesis is trying to explain as much trajectories as possible.

For each event there are 4 *.csv* files, called **cells, hits, particle, truth**. The **cells** file contains the constituent active detector cells that comprise each hit. The cells can be used to refine the hit to track association. A cell is the smallest granularity inside each detector module, much like a pixel on a screen. A cell can provide signal information that the detector module has recorded in addition to the position.

For each hit, the **hits** file contains the numerical identifier for each hit inside the event, associated with the x, y, z position (in millimeters) of the hit in global coordinates.

Furthermore the **particles** file contains the numerical identifier of each particle inside the event, associated with a numerical identifier of the particle type. The initial momentum (in GeV/c) expressed as a 3 components vector, along with the charge of the particles and the number of hits of each particle

are provided, too.

Finally, the **truth** file contains the mapping between hits and generating particles and the true particle state at each measured hit.

5.2 Supervised Learning Approach

In this section we want to explain the reason behind the choice of using a DNN to predict the trajectories of the particles inside the events. As explained above, the ground truth file permits to reconstruct the exact path of each particle across the various detectors: each "*point*" detected is uniquely associated to a single path belonging to some particle. Thus we decided to use this information to train our network to recognize authentic paths.

More precisely, the task our network is asked to solve is the following: given two points, i.e. pixels detected by the silicon layers, the DNN must tell us if the segment connecting the dots is art of a valid trajectory or not. Starting from a single point the DNN is then able to reconstruct the full path of the particle.

Since the training feature, i.e. the validation of a segment, is included in our dataset this approach is in effect a *Supervised Learning Approach*. In particular our solution was inspired by the work of the user *outrunner*, who ended up the competition reaching the second place.

5.3 Other Approaches

In addition to the neural network approach described in this thesis, many other techniques have been proposed to solve the problem. Between these, the most interesting is the one using density clustering.

In particular, many participants used DBSCAN to explore the training set. Density-based spatial clustering of applications with noise (DBSCAN) is a data clustering algorithm proposed by Martin Ester and Hans-Peter Kriegel in 1996. It is a density-based clustering non-parametric algorithm: given a set of points in some space, it groups together points that are closely packed together (points with many nearby neighbors), marking as outliers points that lie alone in low-density regions (whose nearest neighbors are too far away). DBSCAN is one of the most common clustering algorithms and also most cited in scientific literature. In our study case from the starting point the track pattern recognition is solved as a clustering problem. Each of the clusters corresponds to one track. In order to highlight the fact that a

track is approximately an arc of helix hit coordinates are modified as follows:

$$r_1 = \sqrt{x^2 + y^2 + z^2}$$

$$x_2 = x/r_1$$

$$y_2 = y/r_1$$

$$r_2 = \sqrt{x^2 + y^2}$$

5.4 Detectors Geometry from Data

Before discussing the results it is useful to have a deeper knowledge of the dataset in terms of geometry: we now want to describe the scattered points in the various planes of the experiment's volume. All data refers to the silicon detectors, i.e. the so called *pixel detector*.

For simplicity, we consider the z axis as the axis along which the protons move before colliding. Let us firstly plot the scattered points in the xy plane.

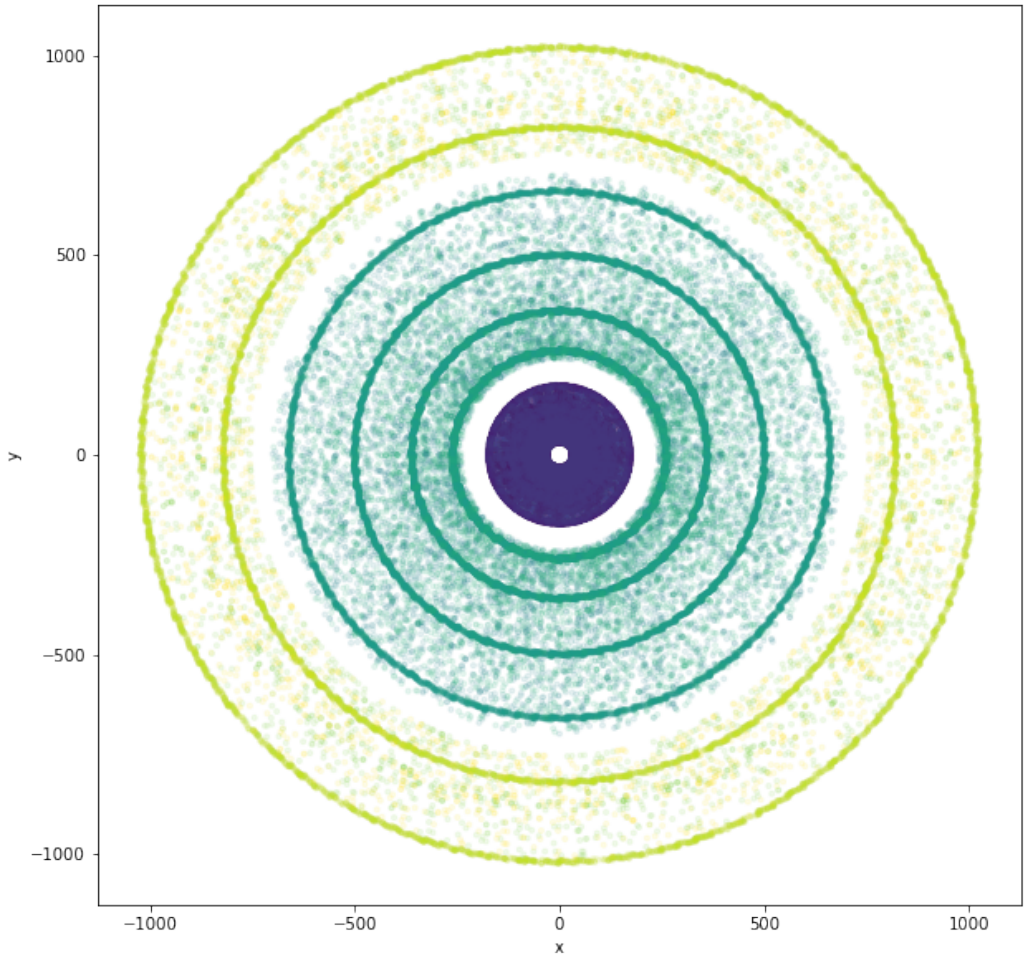


Figure 5.2: 2D projections in the xy plane for all scattered points inside one event. 3 different volumes corresponding to different colors can be distinguished.

As we can see from Figure 5.2 there are a series of concentric rings with some random scattered hits in between the rings. The concentric-ring type geometry is very common in collider physics, as it matches the cylindrical symmetry of collisions well. In order to better understand the detectors geometry and explain the presence of scattered hits between rings let us plot the same graph in the xz axis.

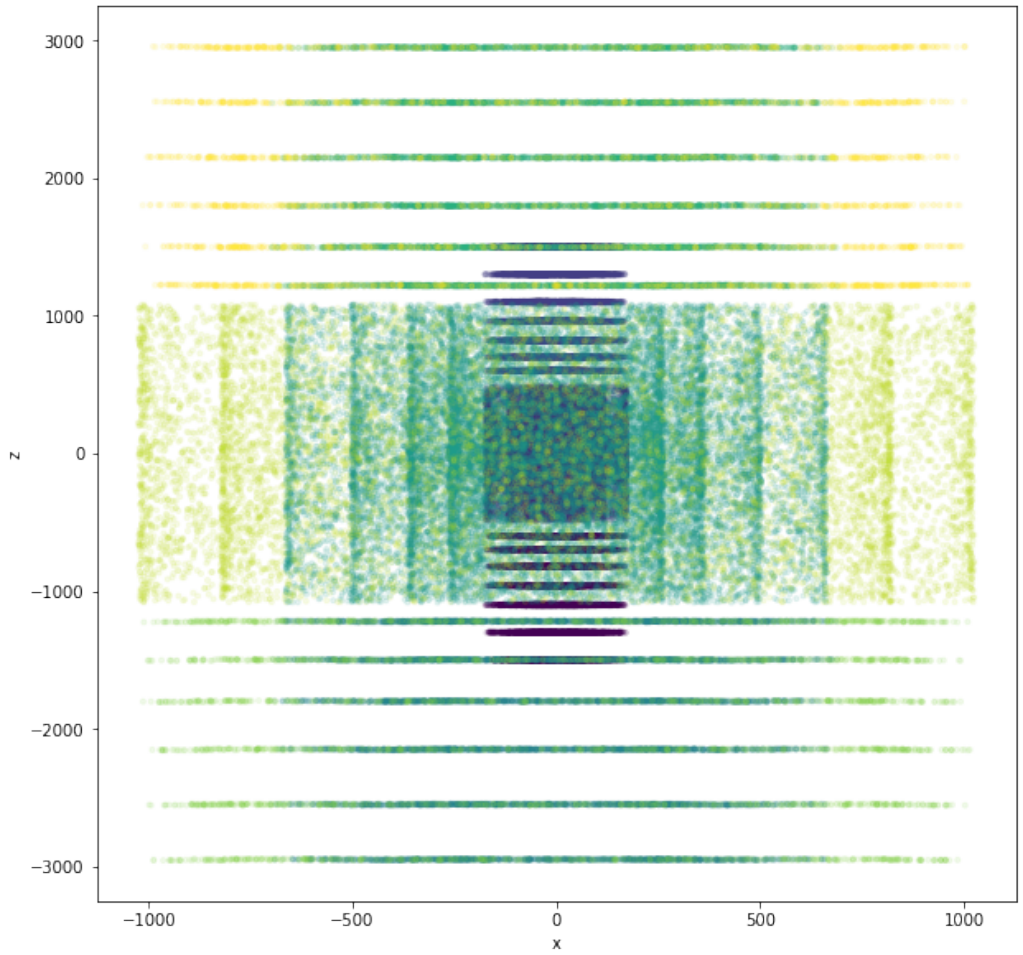


Figure 5.3: 2D projections in the xz plane for all scattered points inside one event. Horizontal bands correspond to concentric-rings described in xy plane.

Figure 5.3 solves our previous doubts: random scattering between the rings in the xy proj ection are from vertically aligned layers at both ends of the tracker, i.e. in the regions for $z < -1000mm$ and $z > 1000mm$.the beam is going in the vertical dimension. The region around $z = 0$ consists of concentric circles around the beam pipe. Moreover the displacement of points in Figure 5.3 traces the geometry of the detectors described in Figure 4.4.

Chapter 6

Discussion and Results

We are finally ready to enter the core of the thesis by presenting our neural network architecture. In this chapter we illustrate the algorithm we implemented to study the trajectories of the particle and we draw conclusions from our results.

We demonstrate that a Machine Learning approach can be really useful for High Energy Physics (HPE) in order to track particles, even when the number of hits inside a single event is very high, as discussed in the previous chapter.

All of the following results are performed using *TensorFlow*, an open source library for numerical computation and large-scale machine learning. TensorFlow bundles together a slew of machine learning and deep learning models and algorithms and makes them useful by way of a common metaphor. It uses Python to provide a convenient front-end API for building applications with the framework, while executing those applications in high-performance C++.

All of the code used in this thesis is written in *Python*.

6.1 Neural Network's Architecture

A common habit while performing machine learning is to search for the best possible model by changing parameters such as the *learning rate*, the number of *neurons* inside the network, the number of training *epochs*.

We report here what is found to be a fairly good network, with a reasonable number of neurons and good performances on the training set.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 400)               4400
_________________________________________________________________
dense_1 (Dense)              (None, 200)               80200
_________________________________________________________________
dense_2 (Dense)              (None, 200)               40200
_________________________________________________________________
dense_3 (Dense)              (None, 200)               40200
_________________________________________________________________
dense_4 (Dense)              (None, 100)               20100
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 101
=================================================================
Total params: 185,201
Trainable params: 185,201
Non-trainable params: 0
_________________________________________________________________
```

Figure 6.1: Schematic view of a neural network proposed for this work. This picture is obtained using the **summary** function in TensorFlow.

In Figure 6.1 it is shown the summary of the network we used to train our model: in particular, it contains 5 layers, each consisting of different number of neurons, for a total of 185.201 trainable parameters. In the following sections we will discuss some variations for this model, including adding specific layers and changing the number of epochs.

6.2 Linking Points Approach

Once the DNN is built, we have to answer the most fundamental question of this work: "What does the neural network learn and how?". In principle the dataset is just a bunch of points distributed in a volume: the goal of this thesis is trying to connect the dots in order to reconstruct a real trajectory of a particle. Thus the task of the network must be recognizing dots belonging to the same trajectories and connecting them. In particular the final model

receives couples of points as input and returns as output the most promising couples for a proper trajectory. Once the promising couples are selected, they are unified in a single list of points that will form the final trajectory.

During the training session each data item that feeds the neural network contains 11 attributes for a couple of points distributed as it follows:

- 3 positional arguments (x,y,z) for both points, i.e. **6** attributes
- the amount of energy recorded by the detectors for both points, i.e. **2** attributes
- the number of points belonging to the point trajectory (for both points), i.e. **2** attributes
- **1** ground truth feature, i.e. y_0 (1 or 0)

In particular, if a data item contains two points belonging to the same trajecotry its ground truth feature is 1, 0 otherwise. During the testing session a data item deprived of the ground truth feature is feeded to the DNN; the latter returns 1 if it classifies the points as part of the same trajecotry, 0 if not.

While it is fairly easy to extract points belonging to the same track from the dataset, we had to think of a way to get data items labelled as 0, i.e. containing points from different trajectories. To do that, we select random points from the whole dataset and create a data item labelled as 0. In fact, the chance of extracting randomly 2 points belonging to the same trajectory is statistically insignificant, i.e. lower than 5%.

Since the training set offered by Kaggle contains an heavy amount of events, it was impossible to fully exploit it. Therefore we randomly selected differents events contained in it and used this new subset as training set. In most of the following results the number of epochs is fixed to 50. An epoch is a term used in machine learning and indicates the number of passes of the entire training dataset the machine learning algorithm has completed. Usually you can't pass the entire dataset into the neural network at once. So, you divide dataset into batches.

To make it clear, let us say we have 2000 training examples that we are going to use . We can divide the dataset of 2000 examples into batches of 500 then it will take 4 iterations to complete 1 epoch.

6.3 Results Visualization

It is now time to show our result, always presented both through numerical and graphical support. Being able to visualize graphically the problem is

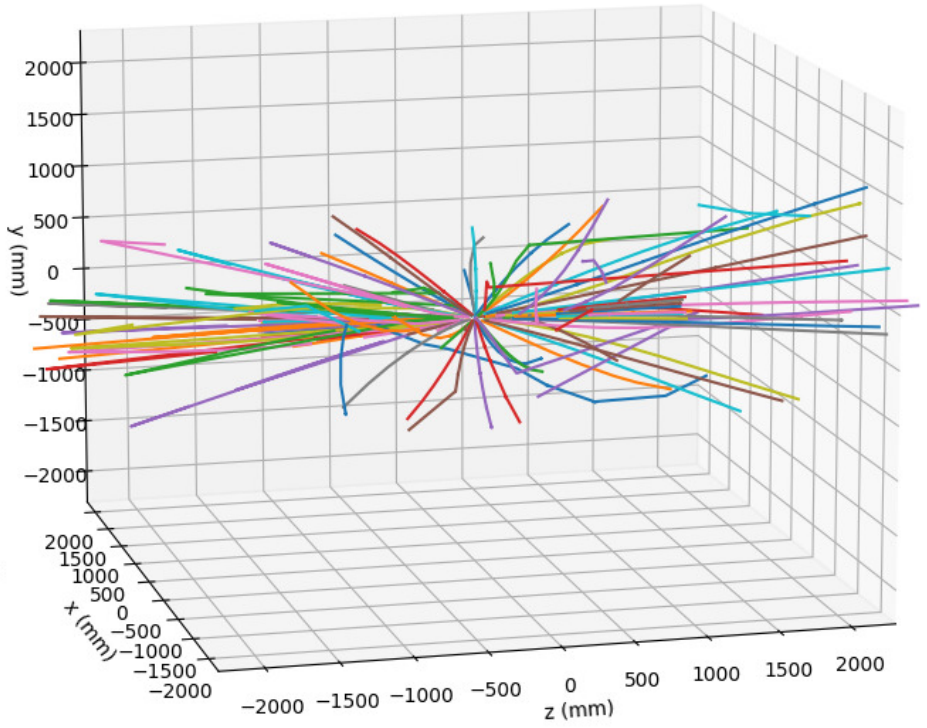


Figure 6.2: Particles' trajectories from a single event represented in the x,y,z axes. Each colored line describe a different particle trajectory starting form the center, i.e. the collision point.

of fundamental importance for this work. In fact a trajectory makes sense if some physical conservation parameters such as momentum, derivative's linearity and continuity are respected.

To give an idea of what happens when two protons collide, we report in Figure 6.2 the graph of all trajectories from event **train1/event000001014**. As you can see from a single proton-proton collision many different particles are created, each one corresponding to a different color. Most of them are characterized by having their highest trajectory's projections along the z axis. This is due to the fact that protons accelerate along the z axis. From now on all particles' trajectories will be studied independently of the others to avoid messy graphs: although a global overview of the experiment will be lost the comparison between real and reconstructed paths will be much easier.

6.4 Accuracy Evaluation Metrics

Before discussing the various models we implemented and experimented it is important to describe the metrics we use to determine a model's fairness. Since no physical quantities can be extracted from the reconstructed trajectories the only observable we can work with is the geometry of the path. In particular, both the ground truth and reconstructed trajectories are lists of 3-dimensional points. For each point of the reconstructed trajectory we compute the minimum distance from the ground truth trajectory. We then sum up all these distances and divide the resulting number by length of the trajectory. By doing that we calculate a percentage error of how good was the model approximation in terms of geometry.

We report here the pseudo code used for this calculation. Given a ground truth set of points G and a reconstructed set R, the percentage error is

Algorithm 1: Percentage error on distances

```
err = 0;
for point in R do
    err += min(dist(point,G);
end
percentage = err / length-of-trajectory;
return percentage
```

Since inside a single event there are thousands of trajectories to be calculated, the average of all percentage errors gives the final accuracy estimator. It is of fundamental importance understanding that a low percentage error does not necessarily mean that the model is really good: our accuracy metric does not take into account some physical constraints that a real trajectory must follow, such as the continuity of the first derivative along the entire trajectory.

of what ?

We report in Figure 6.3 an example of the above discussion for hit 9 inside event 1001. The total length of the real trajectory is 805.8mm and the error measure as described previously is equal to 195.2mm. These values raise a percentage error of 24.2%. Although the percentage error is already quite high, the reconstructed trajectory perfectly traces all points of the real trajectory except for a point in the region of (-55mm,1.05mm,-1300mm) in x,y,z coordinates. Because of this wrong predicted point the whole trajectory becomes unreal: such a strong deviation in direction without any kind of collision could not be observed in a real experiment.

Implementing a new code in order to avoid this kind of errors is however

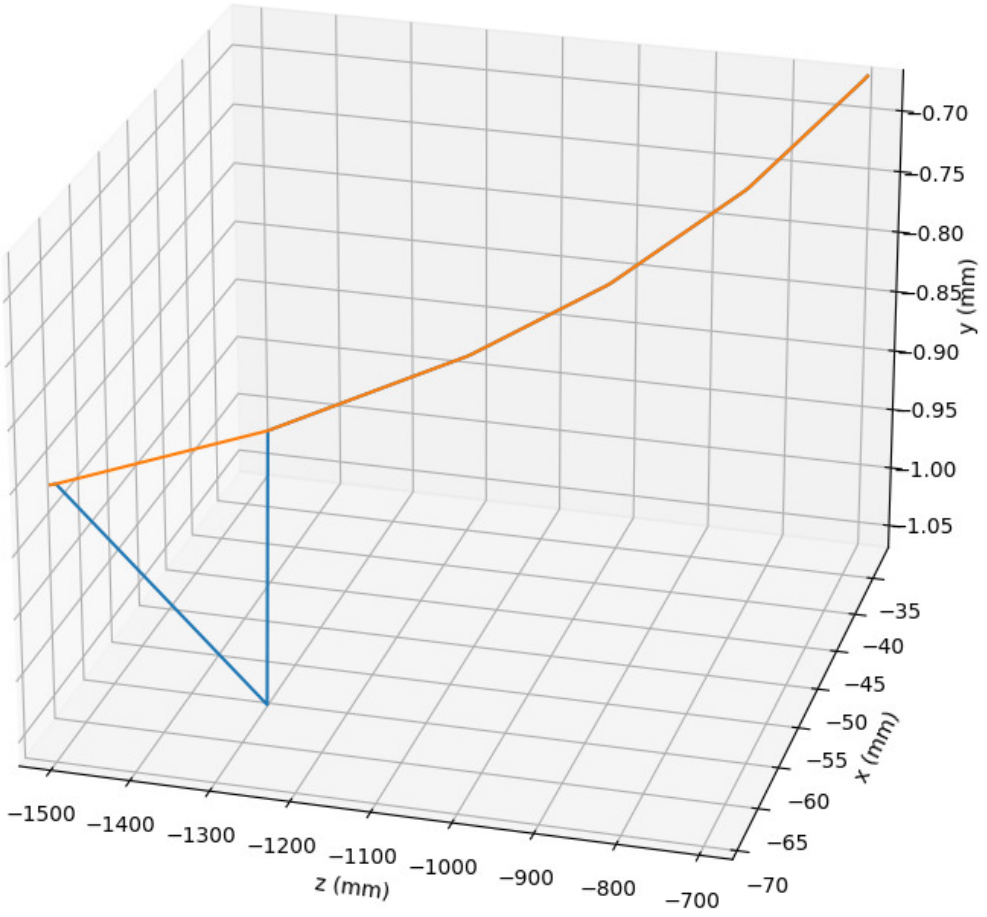


Figure 6.3: Comparison between real (orange) and reconstructed (blue) trajectory with violation of physical constraints.

not part of this work and will be discussed in the Section **Limits and Future Works**.

6.5 Models Classification

It is now time to finally enter the core of this work and discuss the models we implemented using the supervised learning approach. Ascertained that a proper and deep knowledge of a model's goodness would require a list of many physical constraints to be respected, the only criterion we use to determine the best model is by looking at its percentage error when reconstructing new trajectories.

Regardless of the percentage error it is of common use in machine learning to use a small part of the training set (in our case 10%) as a validation set for the model: studying the changing of the loss and the accuracy for the validation set in comparison to the training set is usually a good first indicator of the goodness for the model.

Four different models are here reported and discussed, each one with different features from the others. All models are applied to 4 testing events, never seen during the training phase.

6.5.1 basic10

Let us start from the basic model described in Figure 6.1. From now on we refer to this model as basic10. Its name comes from the fact that the training set for this model is made of 10 different events, chosen randomly from the big training set. This model was trained for 40 epochs and the accuracy and loss changes are reported in Figure 6.4. Although the training loss and validation curves are smooth and rapidly decrease and increase to 0 and 1 respectively, the validation curves are quite noisy and jagged, especially from epoch number 10 on. Fluctuations on the validation set might be a symptom of overfitting. This term is the production of an analysis that corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably. Often an overfitted model is associated with an increasing loss graph, but sometimes a fluctuation in the validation curve could indicate that the model does not work very well with new data, i.e. overfitted.

In addition to this, the average percentage error on the 4 testing events is found to be $26,69\%$. This error tells us that the model "basic10" cannot be considered a good model: since only 10 events were involved during the training phase and a total of 1100 neurons were used to build the neural

network, as described by Figure 6.1, we decided to enlarge both the neural network and the training set.

6.5.2 100 events models

The next 2 models we describe both contain 100 events each as training set. In addiction to that the neural network backbone was modified too: for these models the number of neurons inside each layer was doubled (for a total of 2200 neurons) and a Dropout layer was added. Simply put, dropout refers to ignoring units (i.e. neurons) during the training phase of certain set of neurons which is chosen at random.This means that these ignored units are not considered during a particular forward or backward pass. More technically, at each training stage, individual nodes are either dropped out of the net with probability 1-p or kept with probability p, so that a reduced network is left; incoming and outgoing edges to a dropped-out node are also removed. Dropout helps preventing overfitting: in fact a fully connected layer occupies most of the parameters, and hence, neurons develop co-dependency amongst each other during training which curbs the individual power of each neuron leading to over-fitting of training data.

The big difference between these two models is the batch size, i.e. the number of training examples utilized in one iteration. Whenever a training algorithm has to face big training datasets, it often requires high computational power and times to perform a gradient descent iteration on the whole set. For this reason, the algorithm selects a small number of examples (batch) and performs gradient descent's calculations only on them. Batch sizes for these models are made up of 2000 and 32000 units respectively.

The loss graphs are reported in Figure 6.5: despite the two graphs are very similar it is possible to notice more marked fluctuations in the top figure. This is due to the fact that in general higher batch size usually means more homogeneity during the training process. The average percentage error on the four testing events is $7,52\%$ for the 32000 batch size model and $7,59\%$ for the other. As expected, the better model is the one with the larger batch size, but the deviation from the other model is only $0,7\%$. On one hand these results led us to the conclusion that for this kind of problem the model is not affected by the batch size, while on the other hand they confirmed us that a 10 times bigger training set produces a better result (7.52% error against the 26.69% of the basic10).

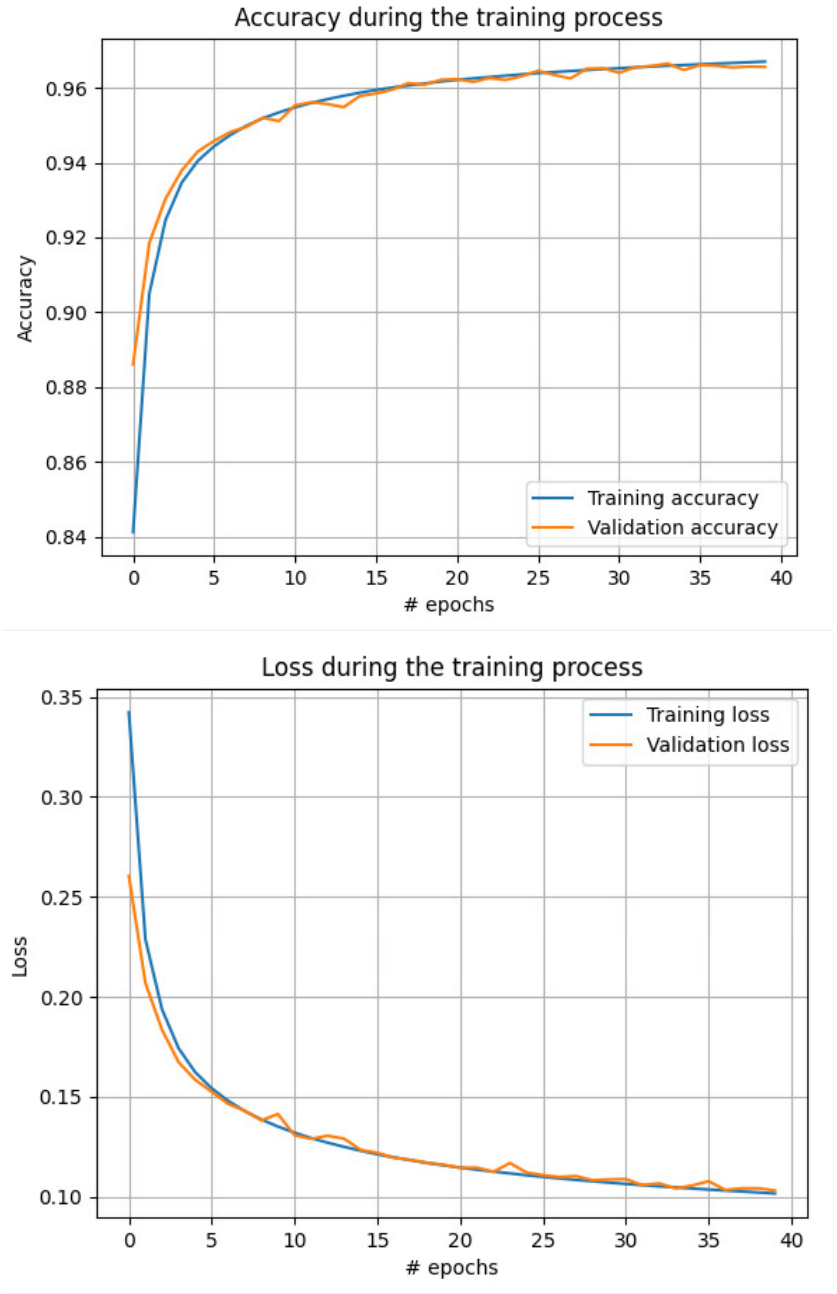
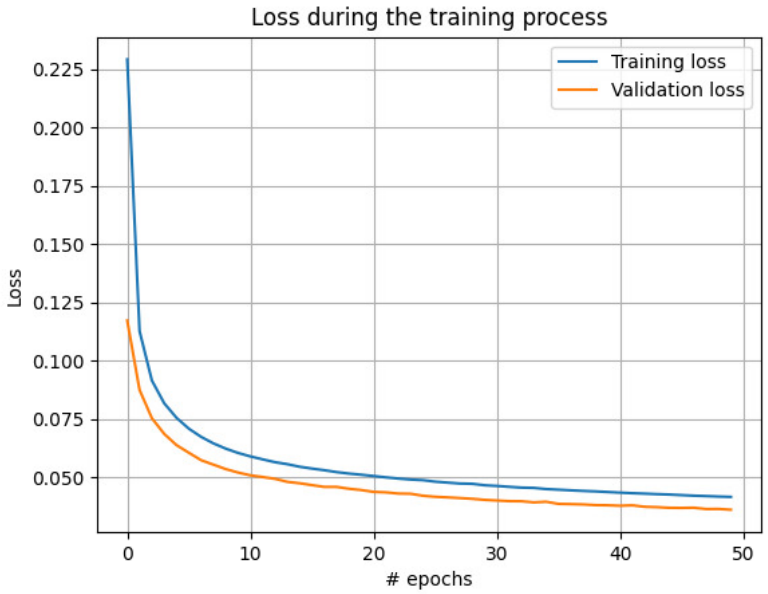
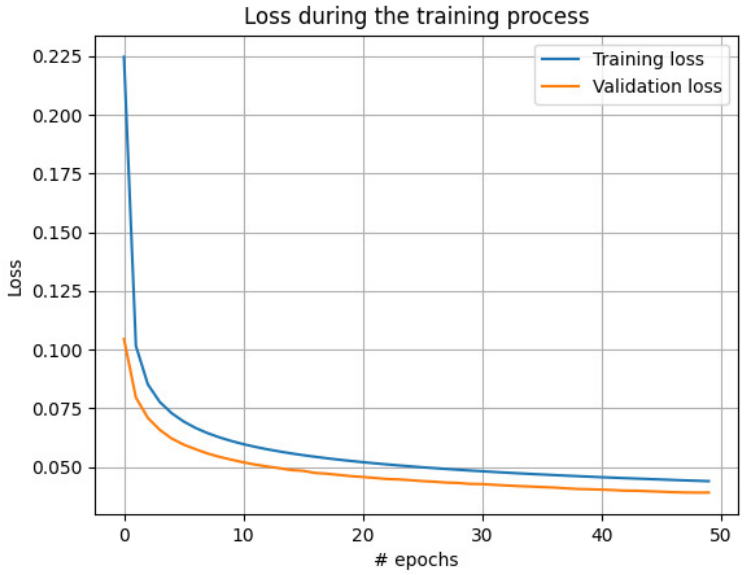


Figure 6.4: From top to bottom: accuracy and loss during the training process for model basic10. Noisy validation curves are indicators of overfitting problems.



(a) Loss for 2000 batch size model



(b) Loss for 32000 batch size model

Figure 6.5: From top to bottom: training and validation loss comparison for 100 events model with 2000 and 32000 batch size respectively.

6.5.3 best80

Last but not least we report here the best model we found out of all. Ascertained that batch size and other tunable parameters such as neurons number, activation function's type and so on did not change the model's performance as much as varying the training number of events, we started to experiment new models by only changing the training set. Interestingly we discovered that the best model is obtained when the training dataset is made up of roughly 80 events.

We report in Figure 6.6 the graphs of accuracy and loss during the training phase: unlike what observed for basic10, the accuracy on the validation set is very close to 1. This fact tells us that the model predicts very well the data contained in the validation set. The average percentage error for this model is equal to $4,05\%$, corresponding to a much lower value compared to the previous models. The reasons behind why this model is better than the others are not easy to find given the so complicated and extended training dataset. One explanation can be the following: since each event is independent from the others and no a-priori guess can be made on how the trajectories would be like after the collision, it could be that having a large amount of events to analyze all at once might confuse the learning algorithm. If all trajectories are different from the others, it is very hard for the neural network to look for common features in order to extract the best possible model. Thus having a restricted amount of data would be better for the training process.

Another possible explanation for this behavior could reside in a disproportionate dataset. Let us suppose that most of the collisions in the dataset produce a certain type of particles A. Accordingly the model will predict very well A-type particles' trajectories and will not be able to identify and characterize different ones. If that is the case, it would be understandable to obtain worse results whenever the training set contains a huge number of events.

Let us know show a recap of the models' performance on the 4 testing events, reported here in Table 6.1. Besides the fact that the model best80 performs much better than the others on all 4 events, it is interesting to see how the 100 event model with the larger batch size does not always predict trajectories better than the other. This is a confirmation of the fact that batch size does not affect in a sensible way the results.

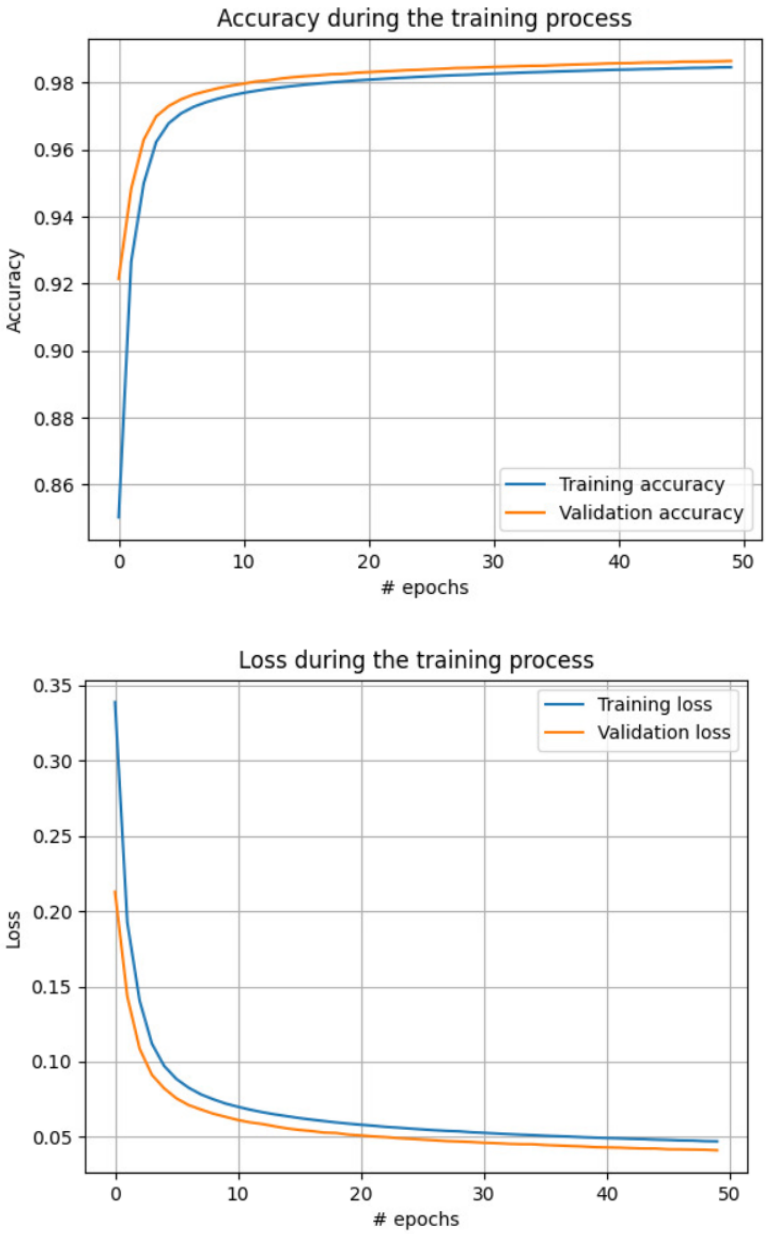


Figure 6.6: From top to bottom: accuracy and loss during the training process for model best80. Curves are flat and no evident fluctuations are observed: the model is not overfitted.

Model	event A (%)	event B (%)	event C (%)	event D (%)
basic10	31.19	29.02	24.03	22.52
100 large bs	4.79	12.24	7.29	5.79
100 small bs	5.56	11.11	8.62	5.09
best80	2.99	4.49	6.88	1.86

Table 6.1: Percentage error on 4 different testing events for each model discussed in the thesis.

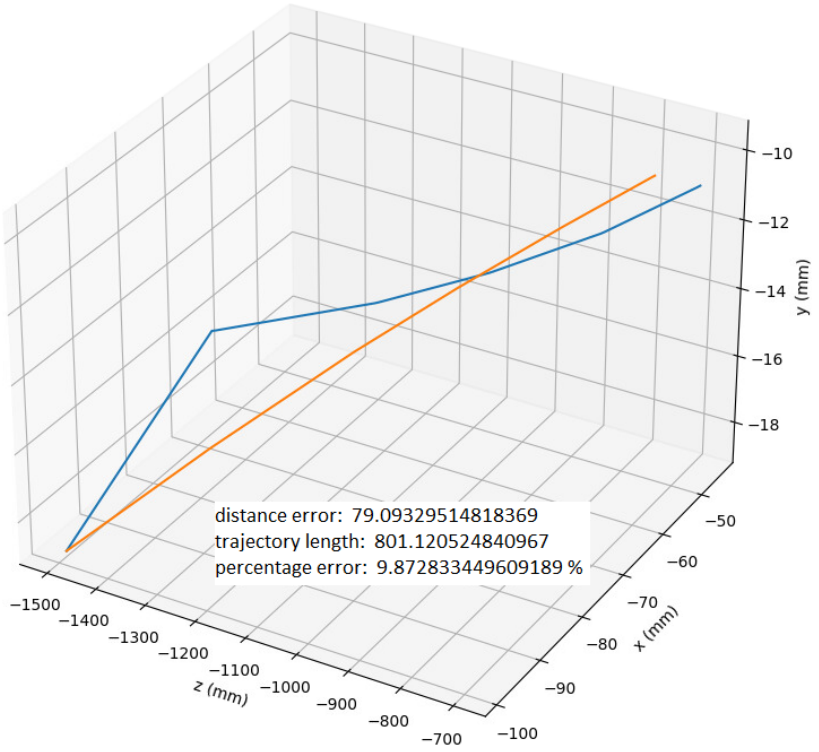
6.6 Predicted Trajectories for best80 Model

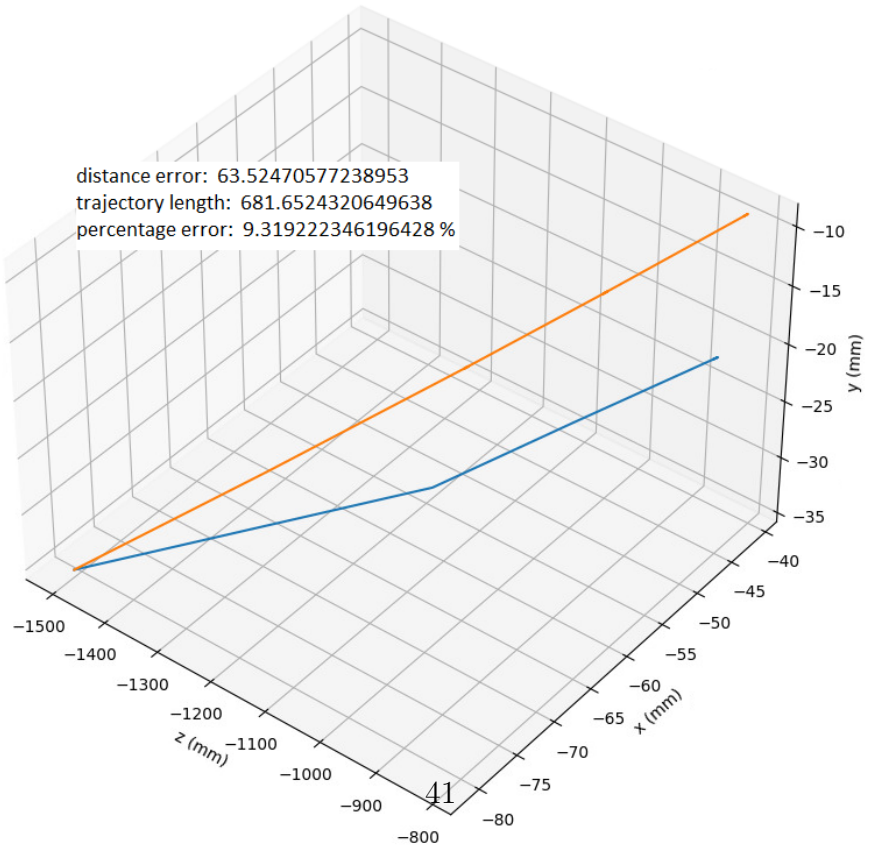
The final step before discussing ideas for the future and possible improvements is to give the reader a visual feedback on the trajectories reconstructed by the neural network. All the images reported below contains a reconstructed trajectory (blue) and its respective original one (orange). It is important to underline that this model is able to predict many trajectories with a 0% error: we decided not to show these pictures because the two lines would be overlapped. For each trajectory we describe its length (mm), its distance error(mm) and the percentage error.

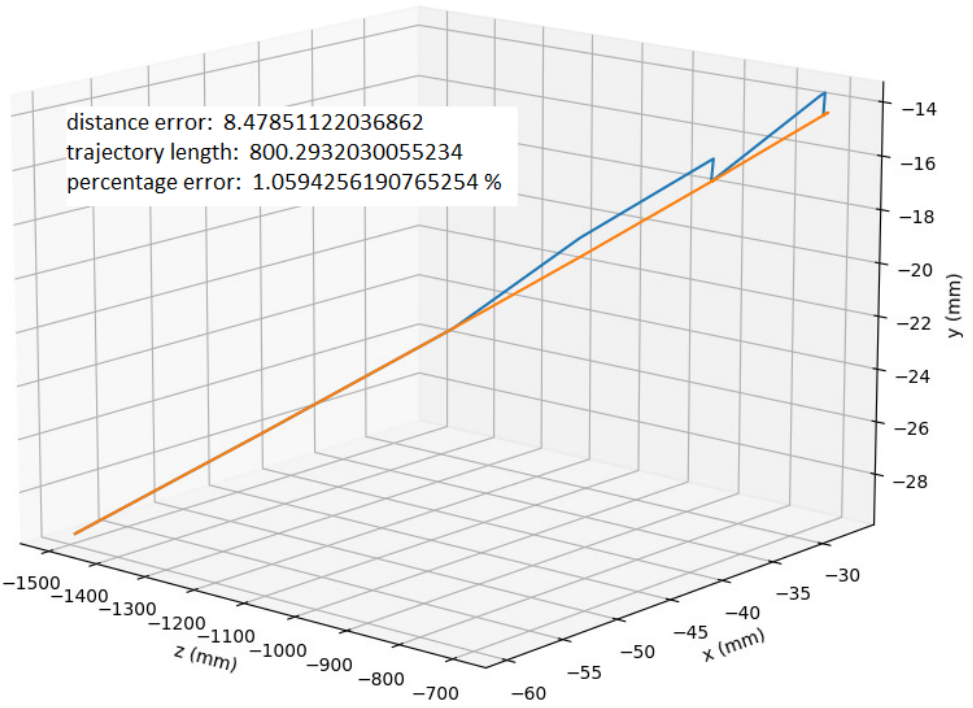
The following paths are selected randomly and through them the reader should be able to approximately associate the magnitude of the error to the type of trajectory that is predicted. Although it is clear that some jagged blue lines cannot describe a real trajectory, it is also evident that the neural network can predict pretty well the region of space that the particle will cross. This is very important because many of these particles are charged and bend whenever they undergo an electromagnetic field. Being able to predict the direction that will be taken by the particles means being able to know its charge and possibly its mass.

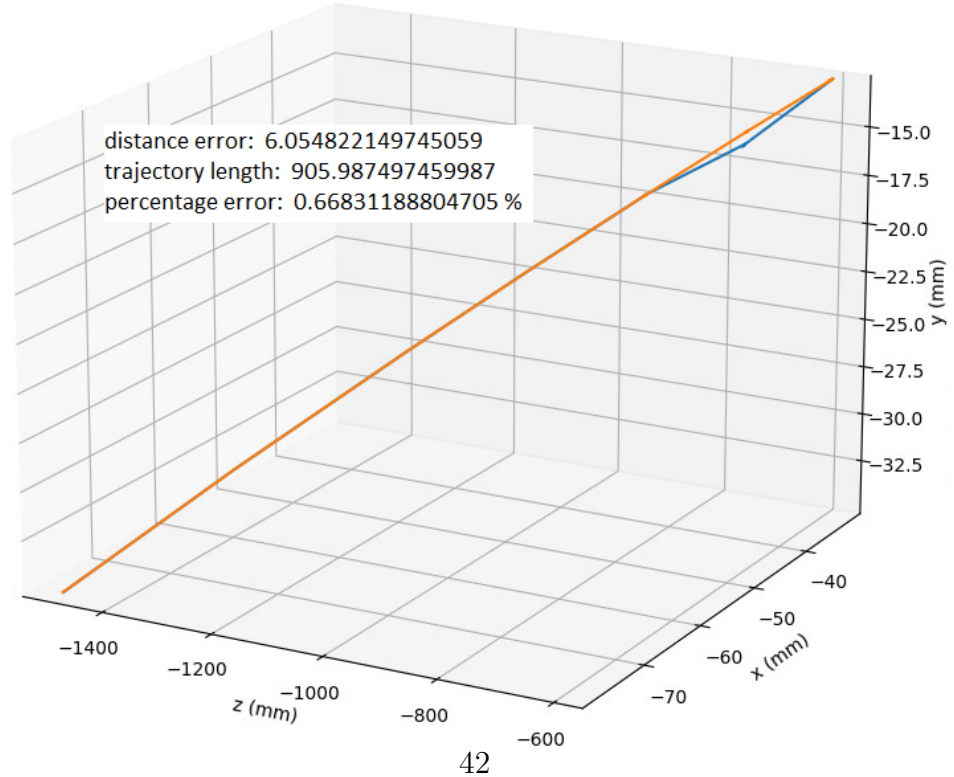
6.7 Limits and Future Works

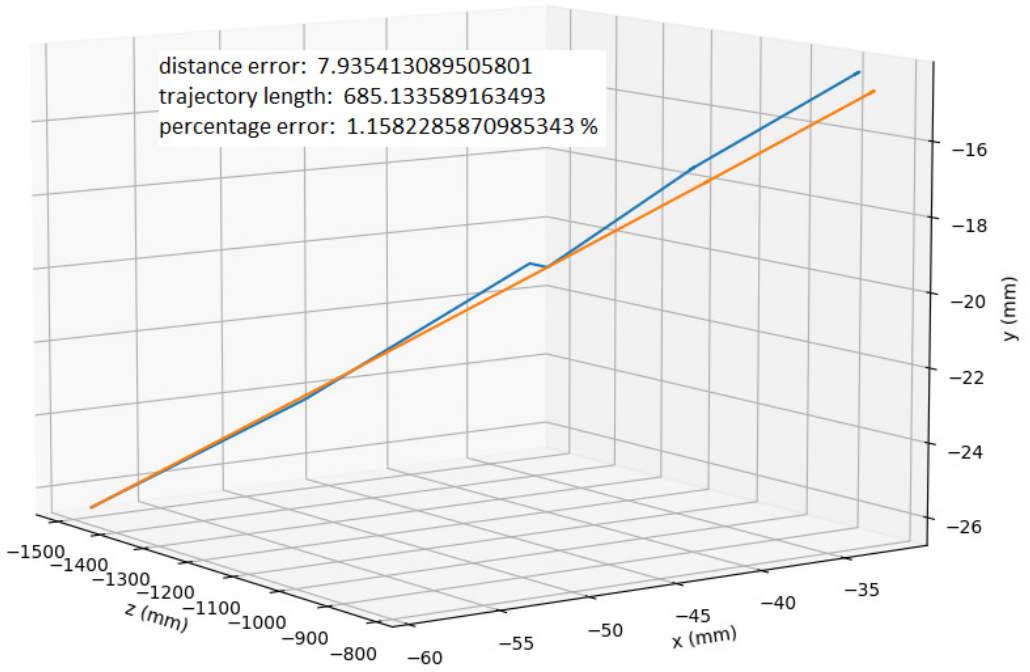
One of the biggest limit of this work lies in the fact that our algorithm does not verify if a predicted trajectory respects all the physical constraints that a real one would have. For example a zig-zag line could never represent a real trajectory: in absence of external forces in fact every body persists in its state of being at rest or of moving uniformly straight forward. In our case the bending of some trajectories is justified by the electromagnetic field inside the ATLAS experiment, but no abrupt change of course should be observed.

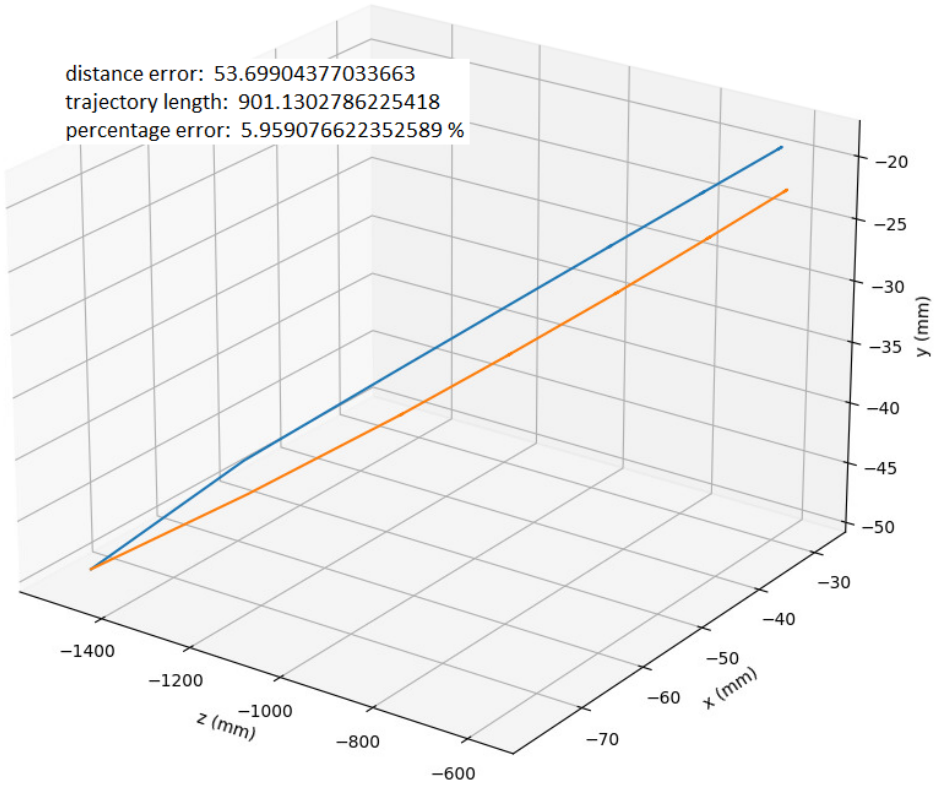

distance error: 79.09329514818369
trajectory length: 801.120524840967
percentage error: 9.872833449609189 %
z (mm)
x (mm)
y (mm)


distance error: 63.52470577238953
trajectory length: 681.6524320649638
percentage error: 9.319222346196428 %
z (mm)
x (mm)
y (mm)


distance error: 8.47851122036862
trajectory length: 800.2932030055234
percentage error: 1.0594256190765254 %
−14
−16
−18
−20
−22
−24
−26
−28
y (mm)
−1500
−1400
−1300
−1200
−1100
−1000
−900
−800
−700
z (mm)
−60
−55
−50
−45
−40
−35
−30
x (mm)


distance error: 6.054822149745059
trajectory length: 905.987497459987
percentage error: 0.66831188804705 %
−15.0
−17.5
−20.0
−22.5
−25.0
−27.5
−30.0
−32.5
y (mm)
−1400
−1200
−1000
−800
−600
z (mm)
−70
−60
−50
−40
x (mm)


distance error: 7.935413089505801
trajectory length: 685.133589163493
percentage error: 1.1582285870985343 %
−16
−18
−20
−22
−24
−26
y (mm)
−1500
−1400
−1300
−1200
−1100
−1000
−900
−800
z (mm)
−60
−55
−50
−45
−40
−35
x (mm)


distance error: 53.69904377033663
trajectory length: 901.1302786225418
percentage error: 5.959076622352589 %
−20
−25
−30
−35
−40
−45
−50
y (mm)
−1400
−1200
−1000
−800
−600
z (mm)
−70
−60
−50
−40
−30
x (mm)

Correcting this kind of problem is not an easy task: since we are dealing with three dimensional spaces, imposing a continuity in terms of direction's variation would mean being able to calculate for each point its differential. A possible future implementation for this work would then be forcing the algorithm to respect some basic motion laws such that those reconstructed lines can really represent particles' trajectories.

In addition to that it would be really interesting to understand if, inside each event, the reconstructed trajectories are such that every conservation's law is respected. This task is however nearly impossible to achieve: no specific information about momentum and time-detection are contained in the dataset. The only available data associated to each point is the amount of energy released by the travelling particle.

6.8 Hardware and Software Specs

All of the work is done using Pyhton 3.8 and all the models are realized using the library tensorflow. All training sessions is performed by a GPU NVIDIA Tesla V100 (32GB). In addiction to Tensorflow the following libraries are needed to extract, read and analyze the dataset: numpy, pandas, matplotlib. For any question or request on the code we used please contact us via e-mail at nicolavanoli96@gmail.com.

Chapter 7

Conclusion

The goal of this thesis was looking for new techniques able to help the exploration and discovery of the quantum world. In chapter 1 we introduced the theoretical concepts behind machine learning and we summarized how neural networks work. We then discussed the standard model in chapter 2, highlighting the fact that there is not a complete theory able to describe all observed quantum phenomena. For this reason the LHC was built and many experiments are being performed in order to discovery new particles and laws. Between these one of the most interesting is called ATLAS, i.e. a pixel detector able to record the passage of new formed quantum particles, described in chapter 3. The main problem with these detectors lies in the fact that it turns out to be very hard to associate to each detected point the exact particle that crossed it. Chapter 4 is thus an introduction to our machine learning approach, where we presented our neural network and discussed its features. Finally in chapter 5 we showed our experimental results and discussed in particular the so called "best80" model, able to reconstruct the particles trajectories from their creation to the edge of the detectors in a very promising way. What we achieved was demonstrating that artificial intelligence, in this case machine learning, can be a very useful tool for particle physics and we hope that in the future these new technologies will be used more frequently by all scientists.

Bibliography

[1] G. Barr, R. Devenish, R. Walczak, and T. Weidberg, *Particle physics in the LHC era*, Oxford master series in particle physics, astrophysics, and cosmology, Oxford University Press, Oxford, 2016.

[2] A. Bettini, *Introduction to particle physics*, Cambridge University Press, 2014.

[3] CERN, *Atlas experiment at cern,atlas.cern.*

[4] A. Collaboration, *Electron efficiency measurements with the atlas detector using the 2015 lhc proton-proton collision data*, Nature Communications, (2016).

[5] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition*, O'Reilly Media, Inc., 2014.

[6] Kaggle, *Track ml challenge, https://www.kaggle.com/c/trackml-particle-identification.*

[7] S. Russel, *Artificial Intelligence: A Modern Approach*, Pearson, 2014.

[8] S. B.-D. Shai Shalev-Shwartz, *Understanding Machine Learning*, Cambridge University Press, 2014.