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

MASTER THESIS

Nicola Vanoli

Particle Tracking through Machine Learning

Relatore: Prof. Daniele Tessera
Correlatore: Chiarissimo Prof. Fausto Borgonovo

~~Brescia, December 2020~~

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 amounts 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

Introduction

LAST THING TO WRITE

Chapter 1

Machine Learning

Over the past two decades, with the ever increasing amounts 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 interesting ways 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 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.[2]

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.[2] The most known discipline belonging to this area is the so called *Machine Learning*.

1.1 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).[3] 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.

1.2 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

$$y = f^*(\underline{x}) \ , \ \underline{x} \in \mathbb{R}^d \tag{1.1}$$

A first approximation function could be of the type

$$\tilde{y}(\underline{x}) = \underline{w} \cdot x + b \ , \underline{w} \in \mathbb{R}^d \ , \ b \in \mathbb{R} \tag{1.2}$$

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**.

1.3 Evaluation and Optimization

The next step is answering the following question: how do we tell that some parameters are better than others? Is there a mathematical metric function that can help us?

In some way the goal is trying as explained above 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 we define

$$L(\underline{x}^{(i)}, y^{(i)}) := (\tilde{y}(\underline{x}^{(i)}) - y^{(i)})^2 \tag{1.3}$$

$$L(D) := \frac{1}{N} \sum_{(\underline{x}^{(i)}, y^{(i)}) \in D} L(\underline{x}^{(i)}, y^{(i)}) \tag{1.4}$$

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 (1.4) 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.(1.2) 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**

$$\theta^* := argmin_\Theta L(D, \theta) \tag{1.5}$$

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

$$\frac{\partial}{\partial \theta} L(D, \theta) := \frac{1}{N} \sum_D \frac{\partial}{\partial \theta} L(\hat{y}^{(i)}, y^{(i)}, \theta) \tag{1.6}$$

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 by the user is reached. In eq.(1.6) 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:

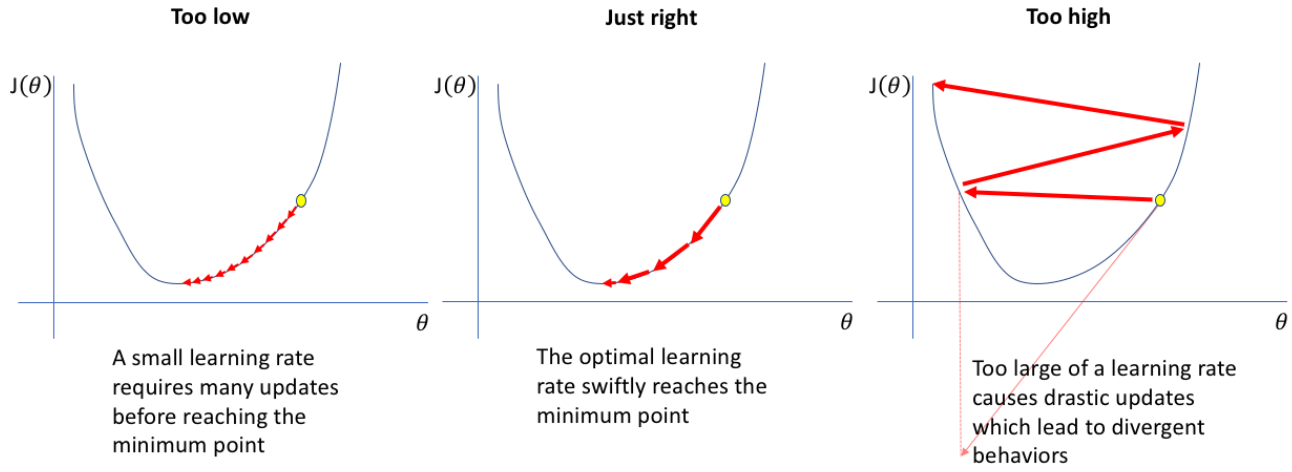

Figure 1.1: Effects of the learning rate's choice on gradient descent method: red arrows represent the various step of the training process. Minimum is reached in computable times only in the middle graph.

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.

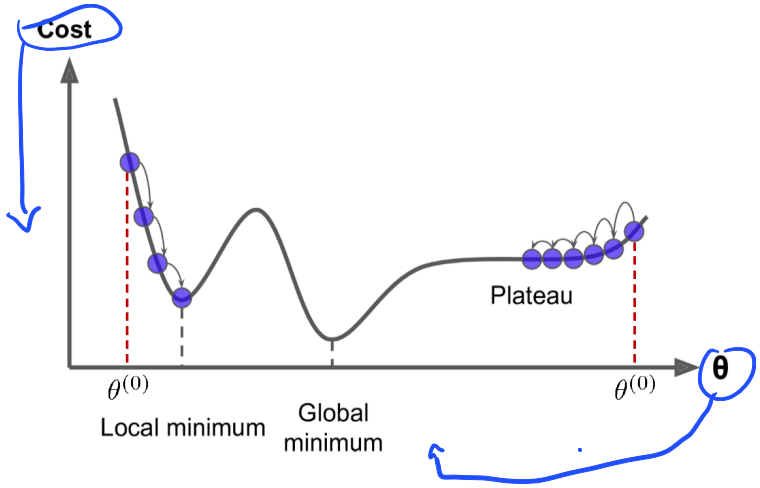

Figure 1.2: Purple dots represent the status of gradient descent at each step: depending on the starting point the result may vary and different minima are found.

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*

$$||\nabla_\theta L(D, \theta_1) - \nabla_\theta L(D, \theta_2)|| \leq C\, ||\theta_1 - \theta_0|| \tag{1.7}$$

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.

1.4 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*.

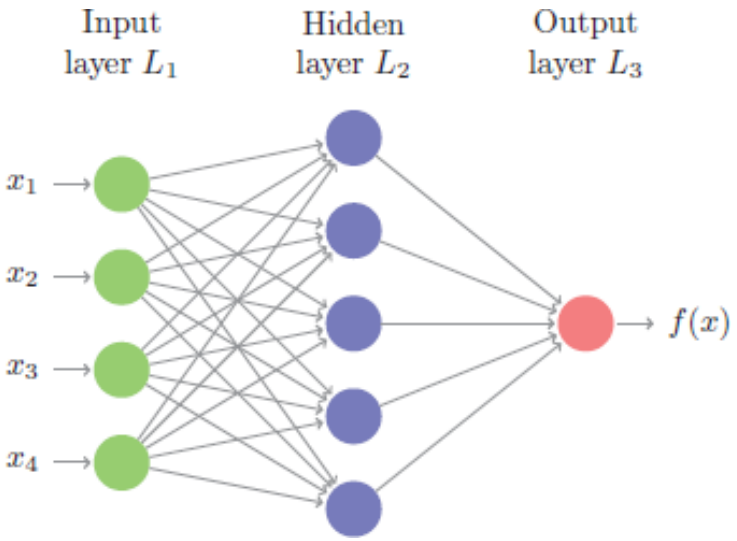

Figure 1.3: Representation of a simple feed-forward neural network. The input $\underline{x} = (x_1, x_2, x_3, x_4)$ goes through the network and the output $f(\underline{x})$ is returned.

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 1.3: unlike the simple case described by (1.2), now the input is processed by the network through the neurons of the hidden layer. We can describe the new **approximation function** as:

$$\tilde{y} = \underline{w} \cdot g(\mathbf{W}\underline{x} + \underline{b}) + b \ , \ \mathbf{W} \in \mathbb{R}^{hxd} \ , \ \underline{w}, \underline{b} \in \mathbb{R}^h \ , \ b \in \mathbb{R} \tag{1.8}$$

$\mathbf{W}$ represents a matrix describing the connections between the input and the hidden layer; in reference to Figure 1.3 $d = 4$ (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*.

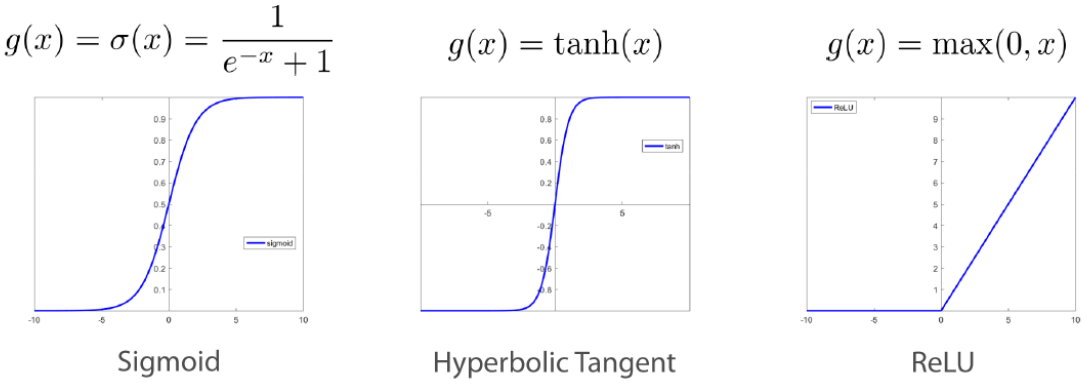

Figure 1.4: Most common choice of g: in addition to adding non-linearity activation function perform a control role avoiding signals to diverge to infinity.

1.5 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*

$$\tilde{y} = \underline{w} \cdot g(\mathbf{W}\underline{x} + \underline{b}) + b \tag{1.9}$$

approximates the target function by less than ϵ

$$sup_x |f^*(x) - (\underline{w} \cdot g(\mathbf{W}\underline{x} + \underline{b}) + b)| < \epsilon \tag{1.10}$$

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.

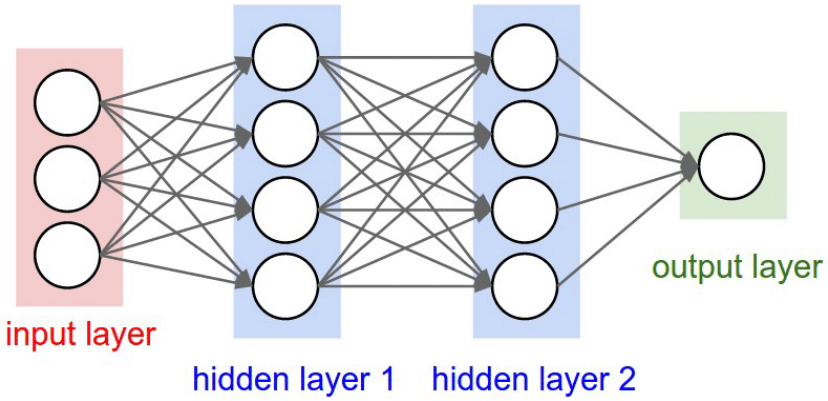

Figure 1.5: Example of deep neural network containing 2 layers.

1.6 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 2

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.

2.1 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

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.

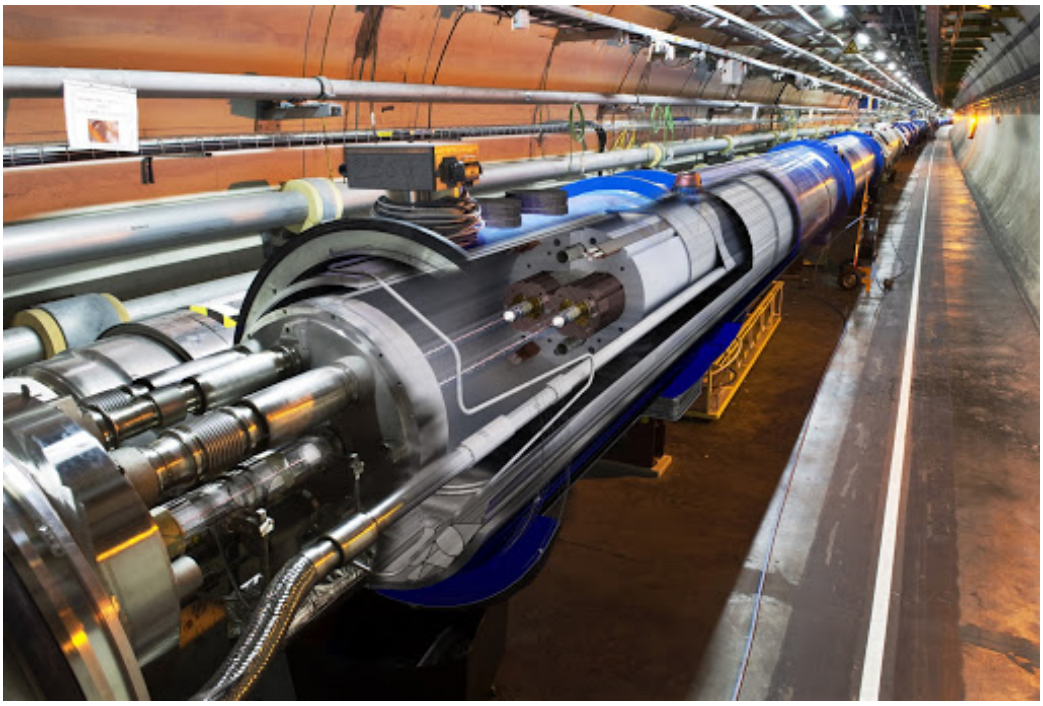

Figure 2.1: Section of the LHC. The two adjacent beams running in opposite directions can be seen in the middle of the picture.

A schematic view of the LHC ring and its pre-accelerator complex is shown in Figure 2.2.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.[1] 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.[1]

Experiments take place in four different location, visible on Figure 2.2: ATLAS, ALICE, CMS and LHCb. The dataset analyzed in this thesis was

recorded by the ATLAS experiment.

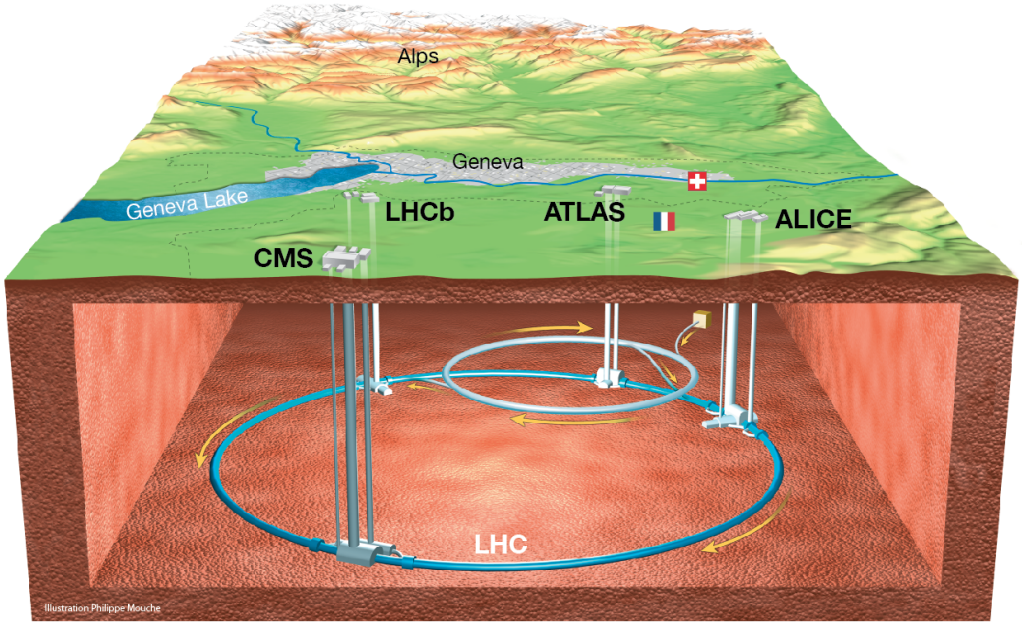

Figure 2.2: Schematic view of the LHC accelerator ring and the location of the four main experiments ATLAS, CMS, ALICE and LHCb.

2.2 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 2.3. 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.

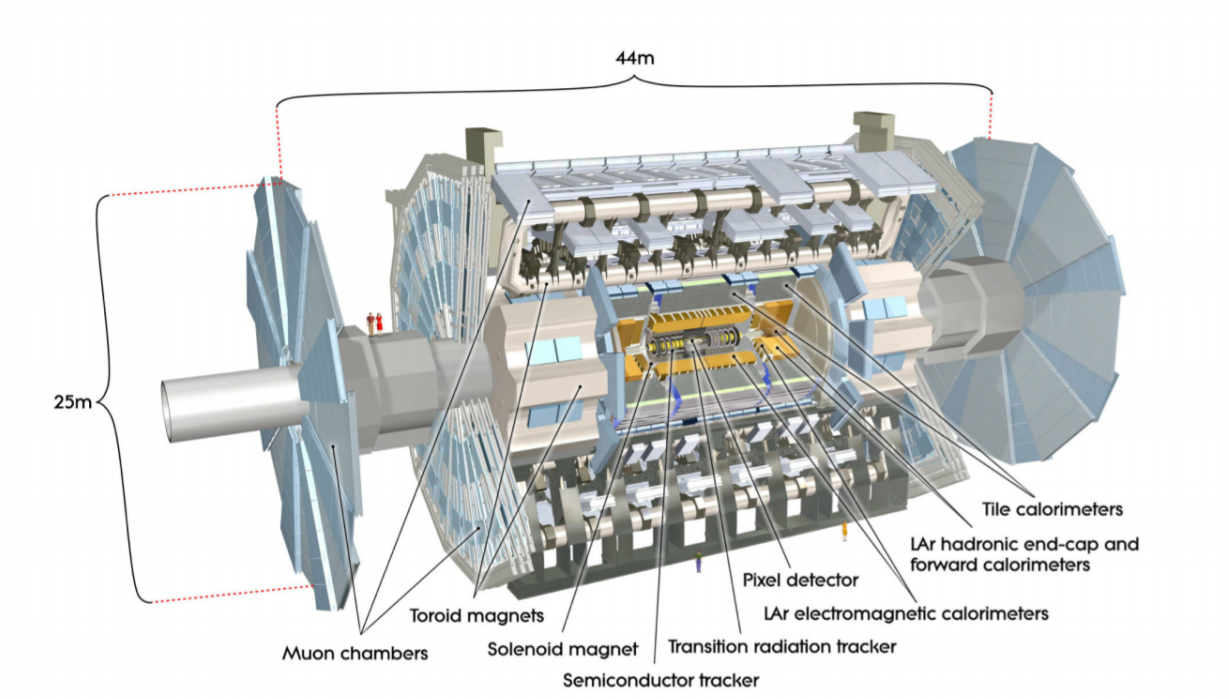

Figure 2.3: Overview of the ATLAS detector showing the muon system, the toroidal and solenoid magnets, the calorimeters and the inner detector.

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.

2.3 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).

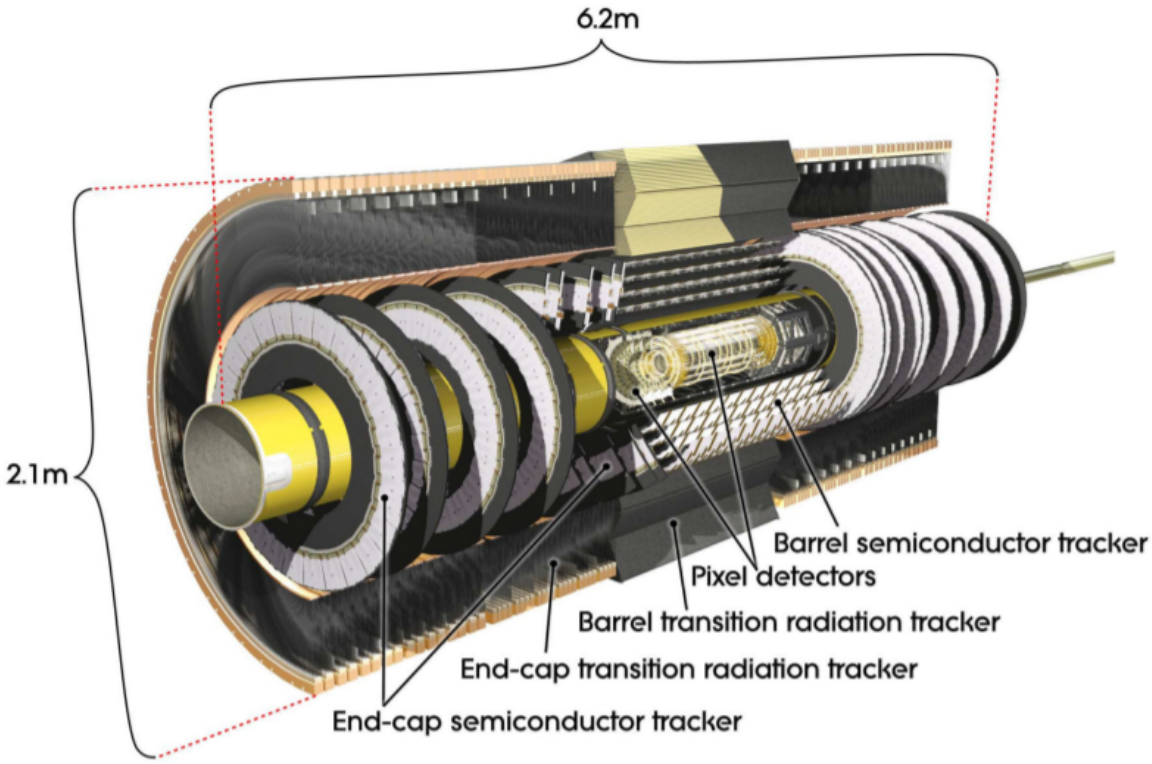

Figure 2.4: Schematic view of the ATLAS inner detector with the pixel detector, the semiconductor tracker and the transition radiation tracker.

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.

2.4 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.

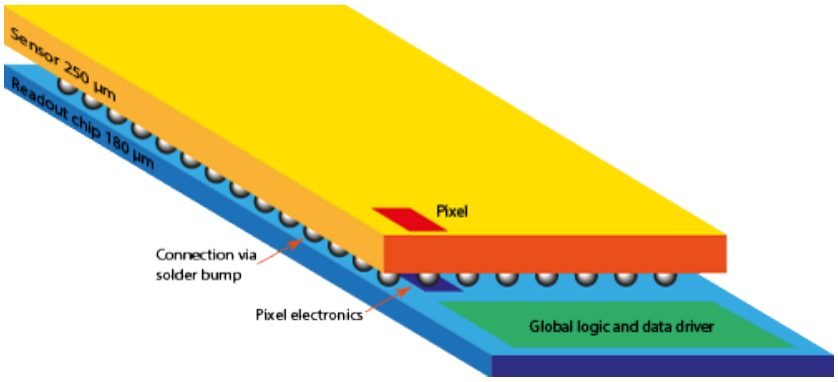

Figure 2.5: Schematic view of a pixel detector inside the ATLAS experiment. The pixel detectors are used to track the trajectories of the particles.

Figure 2.5 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.

2.5 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 3

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**. 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.

3.1 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.

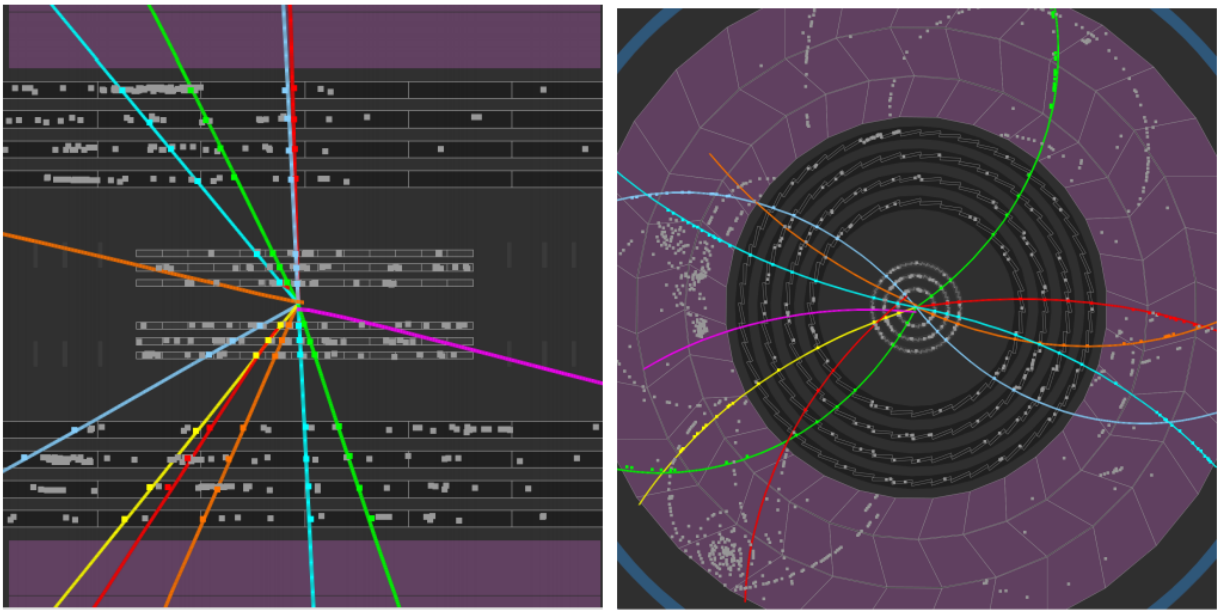

Figure 3.1: Projection of the tracks in the longitudinal and transverse planes, for low multiplicity events in the current detector.

In Figure 3.1 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.

3.2 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.

3.3 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: from the starting point the track pattern recognition is solved as clustering problem. Each of the clusters corresponds to one track. In order to highlight the fact that a track is approximatly an arc of helix hit coordinates are modified as follows:

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

$$x_2 = x/r_1$$

$$y_2 = y/r_1$$

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

3.4 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.

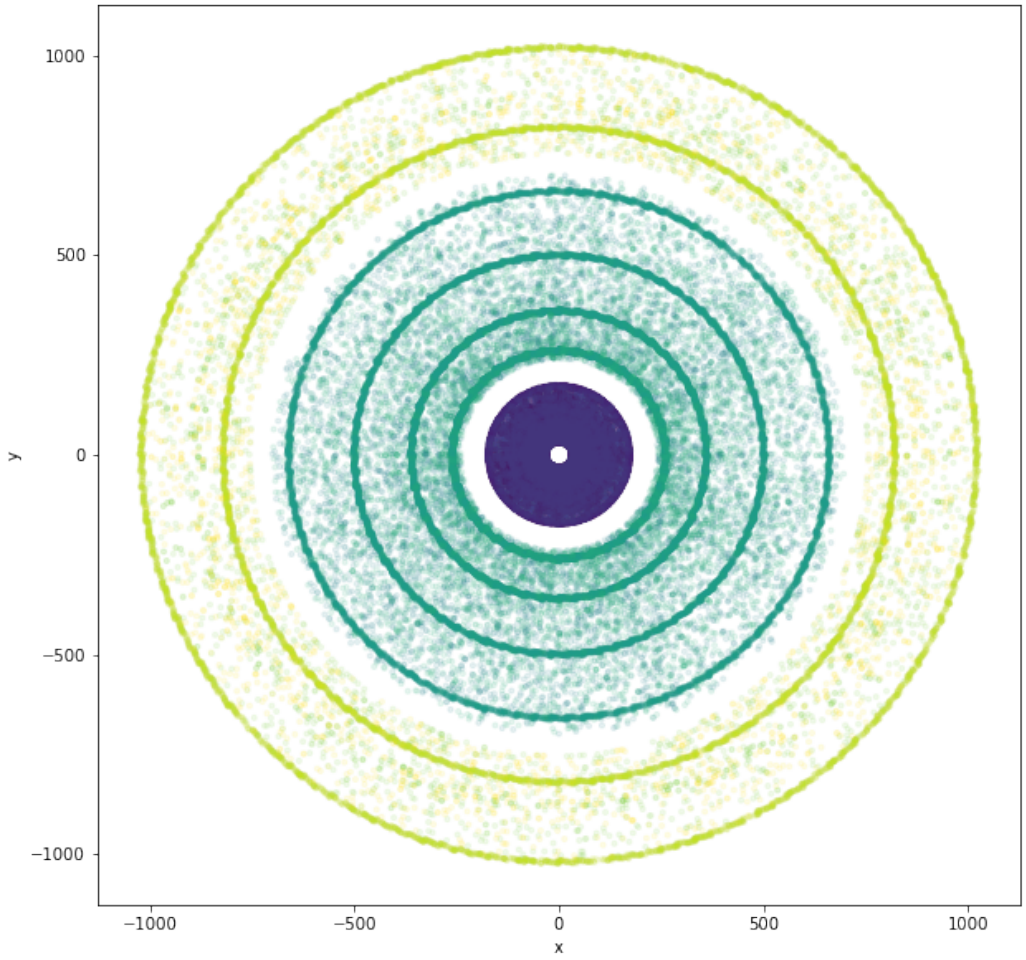

Figure 3.2: 2D projections in the xy plane for all scattered points inside one event. 3 different volumes corresponding to different colors can be distinguished.

As we can see from Figure 3.2 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.

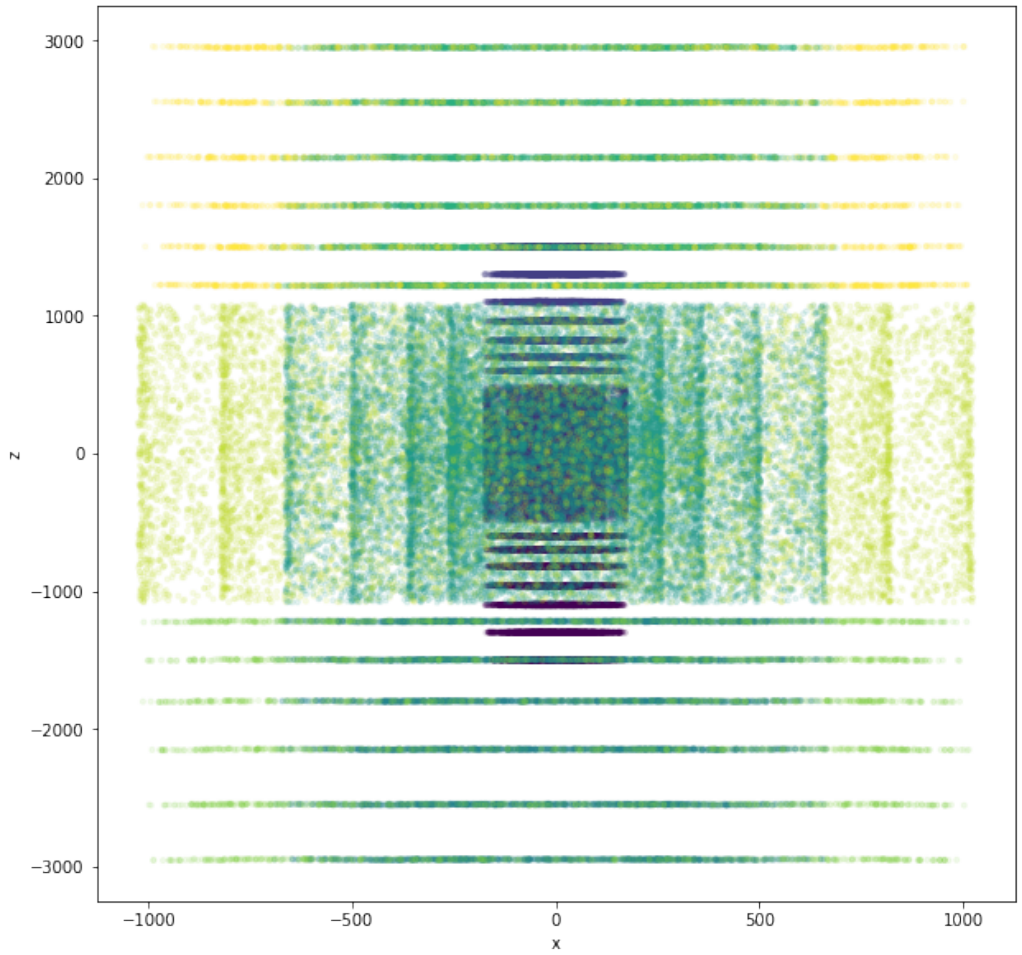

Figure 3.3: 2D projections in the xz plane for all scattered points inside one event. Horizontal bands correspond to concentric-rings described in xy plane.

Figure 3.3 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 3.3 traces the geometry of the detectors described in Figure 2.4.

Chapter 4

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*.

4.1 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 4.1: Schematic view of a neural network proposed for this work. This picture is obtained using the **summary** function in TensorFlow.

In Figure 4.1 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.

Bibliography

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

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

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