Alma Mater Studiorum · Università di Bologna

Scuola di Scienze
Dipartimento di Fisica e Astronomia
Corso di Laurea Magistrale in Fisica

CMS Level-1 Trigger Muon Momentum assignment with Machine Learning

Relatore:
Prof. Daniele Bonacorsi

Correlatore:
Dott. Luigi Guiducci
Dott. Carlo Battilana

Presentata da:
Tommaso Diotalevi

Anno Accademico 2017/2018

Alla mia famiglia.

Contents

Sommario

Con l'avvento della fase ad Alta Luminosità di LHC (HL-LHC), la luminosità istantanea del Large Hadron Collider del CERN aumenterà fino a $7,5 \cdot 10^{34} cm^{-2} s^{-1}$. Pertanto, sono necessarie nuove tecniche algoritmiche per l'acquisizione e l'elaborazione dei dati, in preparazione ad un ambiente con elevato pile-up che renderebbe obsoleti gli attuali apparati di elettronica e trigger.
Attualmente, le tecniche di Machine Learning rappresentano un'alternativa promettente al problema, in quanto rendono possibile la selezione di molteplici informazioni - raccolte dal rivelatore - e costruire da esse diversi modelli, in grado di predire con una certa efficienza quantità fisiche fondamentali, tra cui il momento trasverso p_T. L'analisi presentata in questa Tesi Magistrale consiste proprio nel produrre suddetti modelli, con dati ottenuti mediante simulazioni Monte Carlo, in grado di predire l'impulso trasverso dei muoni che attraversano la regione del Barrel delle camere a muoni di CMS, e compararli con i valori di p_T assegnati dall'attuale sistema di trigger di Livello 1, il Barrel Muon Track Finder (BMTF).

Il Capitolo 1 fornisce una panoramica di LHC, con maggiore attenzione all'esperimento CMS.

Il Capitolo 2 fornisce un'introduzione al Muon System di CMS, con una particolare attenzione al Trigger di Livello 1 relativo alla regione del Barrel.

Il Capitolo 3 introduce alcuni concetti e termini fondamentali di Machine Learning, con uno sguardo al processo di creazione di un modello di Supervised Learning.

Il Capitolo 4 presenta i risultati originali di questo progetto: partendo dal problema fisico studiato, ai processi necessari per la preparazione dei dati e la creazione di una Neural Network per la stima del momento trasverso dei muoni. Infine, vengono discussi i risultati ottenuti.

Abstract

With the advent of the High-Luminosity phase of the LHC (HL-LHC), the instantaneous luminosity of the Large Hadron Collider at CERN will increase up to $7,5 \cdot 10^{34} cm^{-2} s^{-1}$. Therefore, new algorithmic techniques for data acquisition and processing will be necessary, in preparation for a high pile-up environment that would eventually make the current electronics and trigger devices obsolete.
Nowadays, Machine Learning techniques represent a promising alternative to this problem, as they make possible the selection of multiple information - collected by the detector - and build from them different models, able to predict with a certain efficiency fundamental physical quantities, including the transverse momentum p_T.
The analysis presented in this Master Thesis consists in the production of such models - with data obtained through Monte Carlo simulations - capable of predicting the transverse momentum of muons crossing the Barrel region of the CMS muon chambers, and compare the results with the p_T assigned by the current CMS Level 1 Barrel Muon Track Finder (BMTF) trigger system.

Chapter 1 provides a global view of CMS experiment at LHC.

Chapter 2 presents an introduction of the CMS Muon System, with a particular attention to the Level 1 Trigger system of the Barrel region.

Chapter 3 introduces Machine Learning concepts and terminology, offering an overview of a common workflow in the creation of a Supervised Learning algorithm.

Chapter 4 presents the original results of this project: a physical overview of the problem, the processes required for the data preparation and the creation of a Neural Network for the estimation of the particles momentum. In the end, a discussion of the results is given.

Chapter 1

High energy physics at the LHC

1.1 The Large Hadron Collider

The Large Hadron Collider (LHC) [1, 2] is part of the CERN accelerator complex (see Figure 1.1) [3], in Geneva, and it is the most powerful particle accelerator ever built. The particle beam is injected and accelerated by each element of a chain of accelerators, with a progressive increase of energy until the beam injection into LHC, where particles are accelerated up to 14 TeV (by LHC design, its nominal center-of-mass energy).
The LHC consists of a circular 27 km circumference ring, divided into eight independent sectors, designed to accelerate protons and heavy ions. Particles travel in two separated beams on opposite directions and in extreme vacuum conditions (see Section 1.1.1). Beams are controlled by superconductive electromagnets (see Section 1.1.2) , keeping them in their trajectory.
Some of the main LHC parameters are shown in Table 1.1.

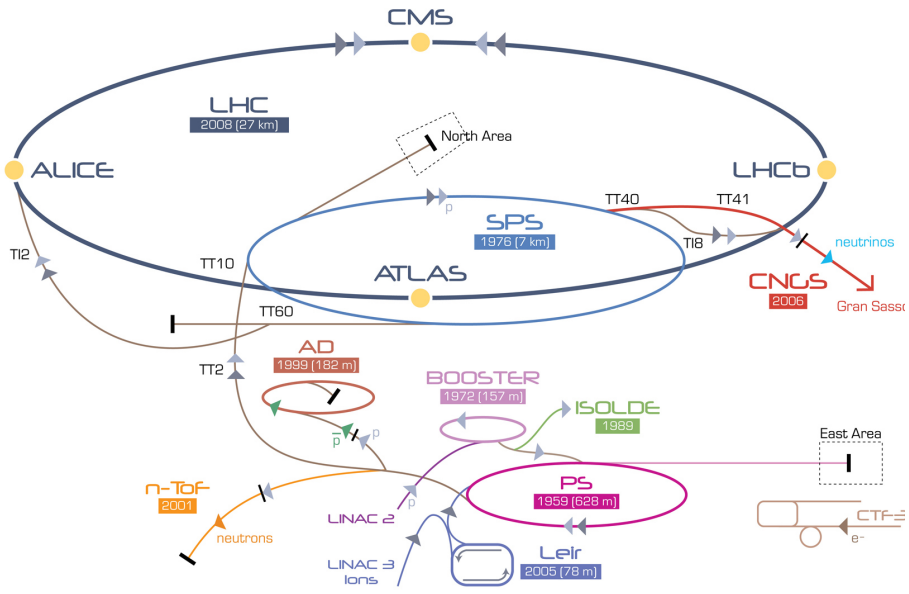


Figure 1.1: The accelerators chain at CERN.

Table 1.1: Main technical parameters of LHC.

Quantity	value
Circumference (m)	26 659
Magnets working temperature (K)	1.9
Number of magnets	9593
Number of principal dipoles	1232
Number of principal quadrupoles	392
Number of radio-frequency cavities per beam	16
Nominal energy, protons (TeV)	6.5
Nominal energy, ions (TeV/nucleon)	2.76
Magnetic field maximum intensity (T)	8.33
Project luminosity ($\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$)	2.06×10^{34}
Number of proton packages per beam	2808
Number of proton per package (outgoing)	1.1×10^{11}
Minimum distance between packages (m)	$\sim$7
Number of rotations per second	11 245
Number of collisions per crossing (nominal)	$\sim$20
Number of collisions per second (millions)	600

1.1.1 The vacuum system

The LHC vacuum system [4] is, with more than 104 kilometers of vacuum ducts, one of the most advanced in the world. It has two functions: the first is to avoid collisions between beam particles and air molecules inside the ducts, creating an extreme vacuum condition (10^{-13}atm), as empty as interstellar space; the other reason is to cancel heat exchange between components that need low temperatures in order to work properly and maximize the efficiency.
The vacuum system is made of three independent parts:

- an isolated vacuum system for cryomagnets;
- an isolated vacuum system for Helium distribution line;
- a vacuum system for beams.

1.1.2 Electromagnets

Electromagnets [5] are designed to guide beams along their path, modifying single particles trajectories as well as align them in order to increase collision probability. There are more than fifty different kind of magnets in LHC, totalling approximately 9600 magnets. Main dipoles generate a magnetic field with a maximum intensity of 8.3 T. In order to reach such an intense field, a current of 11 850 A is needed. To minimize power dissipation, superconducting magnets are employed. A system of liquid He distribution keeps the magnets at a temperature of about 1.9 K. At this incredibly low temperatures, below that required to operate in conditions of superconductivity, helium becomes also super-fluid: this means an high thermal conductivity, thus an efficient refrigeration system for magnets.

1.1.3 Radiofrequency cavities

Radiofrequecy cavities [1] [2] [6] are metallic chambers in which an electromagnetic field is applied. Their primary purpose is to separate protons in packages and to focus them at the collision point, in order to guarantee an high luminosity and thus a large number of collisions.
Particles, passing through the cavity, feel the overall force due to electromagnetic fields and are pushed forward along the accelerator. In this scenario, the ideally timed proton, with exactly the right energy, will see zero accelerating voltage when LHC is at nominal energy while protons with slightly different energies will be accelerated or decelerated sorting particle beams into "bunches". LHC has eight cavities per beam: each of which provides 2 MV at 400 MHz. The cavities work at 4.5 K and are grouped into four cryo-modules.
At regime conditions, each proton beam is divided into 2808 bunches, each containing about 10^{11} protons. Away from the collision point, the bunches are a few cm long and 1 mm wide, and are compressed down to 16 nm near the latter. At full luminosity, packages are separated in time by 25 ns, thus resulting in about 600 million collisions per second.

1.2 LHC detectors

Along the LHC circumference, the particles collide in four intersection points, where the main LHC experiments are located. Each experiment has its own detector, designed and built to gather the fragments of the large number of collisions and to reconstruct all physical processes that generated them.
In particular, the four major experiments installed at LHC are:

- A Large Ion Collider Experiment (ALICE)
- A Toroidal LHC ApparatuS (ATLAS)
- Compact Muon Solenoid (CMS)
- Large Hadron Collider beauty (LHCb)

In addition, there are secondary experiments, among which:

- Large Hadron Collider forward (LHCf)
- TOTal Elastic and diffractive cross section Measurement (TOTEM)

In the following sections, the LHC detectors are briefly introduced, with a major focus on the CMS experiment.

1.2.1 ALICE

ALICE [7, 8] is a detector specialized in heavy ions collisions. It is designed to study the physics of strongly interacting matter at extreme energy densities, where

a phase of matter called "quark-gluon plasma" forms. At these conditions, similar to those just after the Big Bang, quark confinement no longer applies: studying the quark-gluon plasma, as it expands and cools down, allows to gain insight on the origin of the Universe. Some ALICE specifications are illustrated in Table 1.2.
The collaboration counts more than 1000 scientists from over 100 physics institutes in 30 countries (updated to October 2014)

Table 1.2: ALICE detector specifications:

Dimensions	length: 26 m, height: 16 m, width: 16 m
Weight	10 000 tons
Design	central barrel plus single arm forward muon spectometer
Cost of materials	115 MCHF
Location	St. Genis-Pouilly, France

1.2.2 ATLAS

ATLAS [9, 10] is one of the two general-purpose detectors at LHC. Although its similarities with the CMS experiment regarding scientific goals, they have subdetectors based on different technology choices, and the design of the magnets is also different. Some specs are illustrated below in Table 1.3.
It is located in a cavern 100m underground near the main CERN site. About 3000 scientists from 182 institutes in 38 countries work on the ATLAS experiment. Around 1200 doctoral students are involved in detector development, data collection and analysis. The collaboration depends on the efforts of countless engineers, technicians and administrative staff.

Table 1.3: ATLAS detector specifications:

Dimensions	length: 46 m, height: 25 m, width: 25 m
Weight	7000 tons
Design	barrel plus andcaps
Cost of materials	540 MCHF
Location	Meyrin, Switzerland

1.2.3 CMS

CMS [11, 12], as well as ATLAS, is a general-purpose detector at LHC. Is built around a huge solenoid magnet with a cylindrical form able to reach a 3.8 T magnetic field. Its main characteristics are illustrated in Table 1.4. In the next chapter, CMS will be discussed more specifically.

Table 1.4: CMS detector specifications:

Dimensions	length: 21 m, height: 15 m, width: 15 m
weight	12 500 tons
Design	barrel plus end caps
Cost of materials	500 MCHF
Location	Cessy, France

1.2.4 LHCb

The LHCb [13, 14] experiment is specialized in investigating the slight differences between matter and antimatter by studying the quark bottom. Instead of ATLAS or CMS, LHCb uses a series of subdetectors to detect mainly forward particles: the first one is mounted near the collision point while the others are placed serially over a length of 20 meters.
Some specifications are illustrated below in Table 1.5. About 700 scientists from 66 different institutes and universities work on LHCb experiment (updated to October 2013).

Table 1.5: LHCb detector specifications:

Dimensions	length: 21 m, height: 10 m, width: 13 m
Height	5600 tons
Design	forward spectometer with planar detectors
Cost of materials	75 MCHF
Location	Ferney-Voltaire, France

1.2.5 Other LHC experiments

Aside from ALICE, ATLAS, CMS and LHCb, a few details on LHC smaller experiments, LHCf and TOTEM, are given in the following. LHCf [15, 16] is a small experiment which uses particles thrown forward by p-p collisions as a source to simulate high energy cosmic rays. LHCf is made up of two detectors which sit along the LHC beamline, at 140 m either side of ATLAS collision point. They only weights 40 kg and measures (30 x 80 x 10) cm.
LHCf experiment involves about 30 scientists from 9 institutes in 5 countries (updated to November 2012).

TOTEM [17, 18] experiment is designed to measure pp total elastic and diffractive cross section by measuring protons emerging at small angle with respect to the beam lines. Detectors are spread across half a kilometre around the CMS interaction point in special vacuum chambers called "roman pots" connected to beam ducts, in order to reveal particles produced during the collision.
TOTEM has almost 3000 kg of equipment and 26 "roman pot" detectors. It involves about 100 scientists from 16 institutes in 8 countries (updated to August 2014).

1.3 The CMS experiment

The CMS main purpose is to explore the p-p physics at the TeV scale, including precision measurements of the Standard Model, as well as search for new physics. Its cylindrical concept is built on several layers and each one of them is dedicated to the detection of a specific type of particle.
The CMS collaboration consists in over 4000 particle physicists, engineers, computer scientists, technicians and students from around 200 institutes and universities from more than 40 countries. Figure 1.2 shows an image of the CMS experiment.

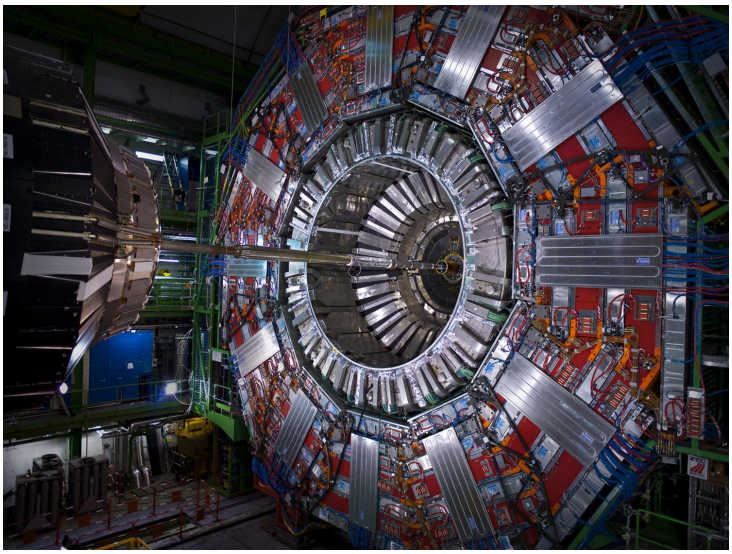


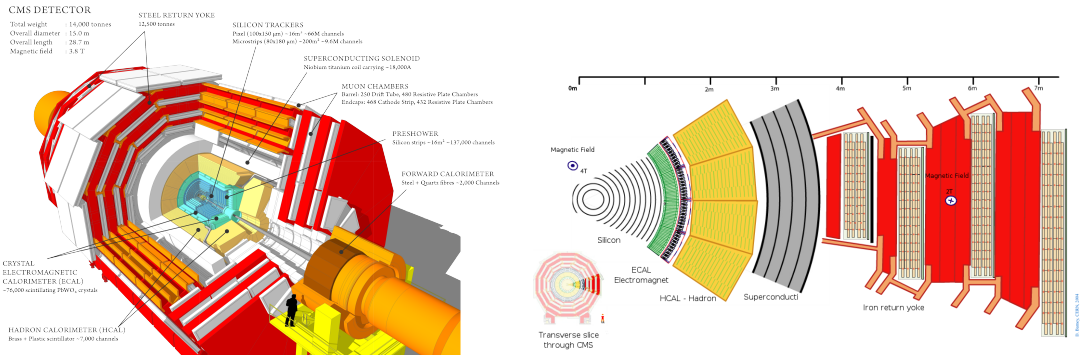
Figure 1.2: A view of the CMS experiment, with one of the endcaps removed.

1.3.1 The CMS detector: concept and structure

As mentioned above, the CMS detector is made up of different layers, as illustrated in Figure 1.3. Each of them is designed to trace and measure the physical properties and paths of different kinds of subatomic particles. Furthermore, this structure is surrounded by a huge solenoid based on superconductive technologies, operating at 4.4 K and generating a 3.8 T magnetic field.
In CMS a right handed coordinate system, centred at the nominal collision point, is definied: the *x-axis* points radially inward to the center of the accelerator ring, the *y-axis* points upward and the *z-axis* is parallel to the beam pipe (in direction of the Jura mountains). The polar angle θ is measured from the *z-axis* using a $0 \leq \theta \leq \pi$ range, while the azimuthal angle ϕ is measured in the x-y plane from the *x-axis* in a $0 \leq \phi \leq 2\pi$ range. The polar angle is usually replaced by another quantity called *pseudorapidity* η defined as:

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

The first and inner layer of the CMS detector is called Tracker [12, pp. 26-89]: made entirely of silicon, it is able to reconstruct the paths of high-energy muons,



(a) Section view.

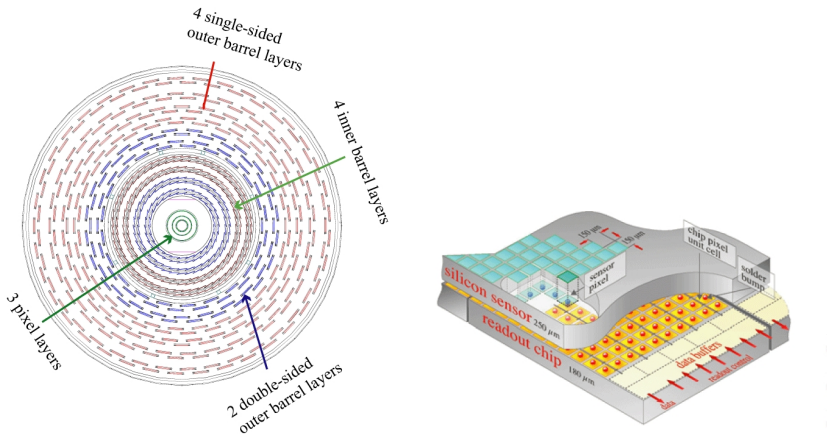
(b) View along the plane perpendicular to beam direction.

Figure 1.3: Compact Muon Solenoid.

electrons and hadrons as well as reconstruct secondary vertices from the decay of very short-lived particles with a resolution of 10 nm.
The second layer consists of two calorimeters, the Electromagnetic Calorimeter (ECAL) [12, pp. 90-121] and the Hadron Calorimeter (HCAL) [12, pp. 122-155] arranged serially. The first measures the energy deposited by photons and electrons, while the second measures the energy deposited by hadrons.
Unlike the Tracker, which interferes scarcely with passing particles, the calorimeters are designed to absorb them.
In the end, after the superconducting magnet, a series of muon detectors [12, pp. 162-246] are able to track muon particles, escaped from calorimeters. The lack of energy and momentum from collisions is assigned to the electrically neutral and weakly-interacting neutrinos.

Tracker

The Tracker is a crucial component in the CMS design as it measures particles momentum through their path: the greater is their curvature radius across the magnetic field, the greater is their momentum. As stated above, the Tracker is able to reconstruct muons, electrons and hadrons path as well as tracks produced by short-lived particles decay, such as quark beauty. It has a low degree of interference with particles and a high resistance to radiations. Located in the inner part of the detector, it is highly irradiated.
This detector (Figure 1.4a) is entirely made of silicon: internally there are three levels of pixel detectors, after that particles pass through 10 layer of strip detectors, until a 130 cm radius from the beam pipe.
The pixel detector, Figure 1.4b, contains about 120 millions of pixels, with three levels of respectively 4, 7 e 11 cm radius. The flux of particles at this radius is maximum: at 8 cm is about 10^6 particles/$cm^2 \cdot s$. Each level is divided into small units, each one containing a silicon sensor of 150 µm × 100 µm. When a charged particle goes through one of this units, the amount of energy releases an electron with the consequent creation of an hole. This signal is than received by a chip which amplifies it. It is possible, in the end, to reconstruct a 3-D image using bi-dimensional layers for each level.



(a) Tracker layers from a point of view perpendicular to beams.

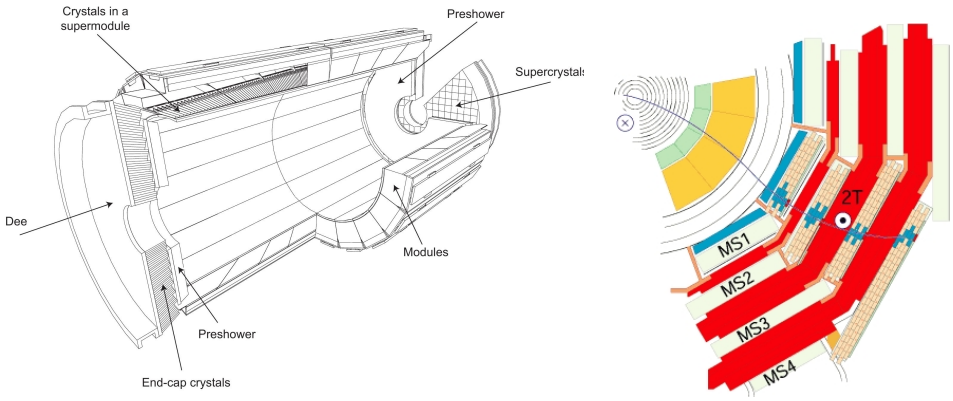
(b) CMS Silicon pixel detector.

Figure 1.4: Graphical depiction of the CMS Tracker subdetector.

The power absorption must remain at minimum, because each pixel absorbs about 50 µW with an amount of power not irrelevant; for this reason pixels are installed in low temperature pipes.
Strip detectors, instead, consist of ten layers. This section of the Tracker contains 10 million detector strips divided into 15 200 modules, scanned by 80 000 chips. Each module is made up of three elements: a set of sensors, a support structure and the electronics necessary to acquire data. Sensors have an high response and a good spatial resolution, allowing to receive many particles in a restricted space; they detect electrical currents generated by interacting particles and send collected data. Also this section of the detector is maintained at low temperature ($-20\,°C$), in order to "freeze" silicon structure damages from radiations and prevent it from perpetuating.

Calorimeters

In the CMS experiment there are two types of calorimeters that measure the energy of electrons, photons and hadrons.
Electrons and photons are detected and stopped by the electromagnetic calorimeter (ECAL). This measurement happens inside a strong magnetic field, with an high level of radiation and in 25 ns from one collision to another. Lead tungstate crystal ($PbWO_4$) is made primarily of metal and is heavier than stainless steel, but with a touch of oxygen in this crystalline form it is highly transparent and scintillates when electrons and photons pass through it. This means it produces light in proportion to the particle's energy. These high-density crystals produce light in fast, short, well-defined photon bursts that allow for a precise, fast and fairly compact detector. Photodetectors that have been especially designed to work within the high magnetic field, are also glued onto the back of each of the crystals to detect the scintillation light and convert it to an electrical signal that is amplified and sent for analysis. The ECAL is divided into a cylindrical body called "barrel" and the two ends called "endcaps" (see Figure 1.5a) creating a layer between the Tracker and the other calorimeter.



(a) ECAL layout showing crystal's modules.

(b) A muon, in the plane perpendicular to beams, leaving a curve trajectory into the four muon stations.

Figure 1.5: The *Electromagnetic CALorimeter* and the *muons detectors*.

Hadrons are instead detected by the hadron calorimeter (HCAL) specifically built for measuring strong-interacting particles. The hadron calorimeter is made up with a series of highly absorbent layers and each time a particle produced from various decays crosses a layer, a blue-violet light is emitted. This light is than absorbed by optic cables of about 1 mm diameter, shifting wavelength into the green region of the electromagnetic spectrum and finally converted to digital data by photodetectors. The optic sum of light produced along the path of the particle it's a measure of its own energy, and is performed by specifically designed photo-diodes.
The HCAL also provides tools for an indirect measurement of non-interacting particles like neutrinos. During hadrons' decay, new particles can be produced which may leave no sign to any detector at CMS. To avoid that, HCAL is hermetic i.e. it captures each particle coming from the collision point, allowing detection of "invisible" particles, through the violation of momentum and energy conservation. Unlike ECAL, HCAL has two more sections called "forward sections", located at the ends of CMS, designed to detect particles moving with a low scattering angle. These sections are built with different materials, with the aim of making them more resistant to radiations since they receive much of the beam energy.

Muon detectors

Muon detectors are placed outside the solenoid because muons are highly penetrant and are not stopped from the previous calorimeters. Their momentum is measured by recreating trajectories along four muons stations interspersed with the iron of the ferromagnets return yoke, shown in red in Figure 1.5b, and synchronizing data with those obtained from the Tracker. There are 1400 muons chambers: 250 "drift tubes" (DTs) and 540 "cathode strip chambers" (CSCs) tracing particles positions and acting at the same time as a trigger, while 610 "resistive plate chambers" (RPCs) form a further trigger system. The next chapter will provide more information about these detectors.

1.3.2 Trigger and Data Acquisition

When CMS is at regime, there are about a billion p-p interactions each second inside the detector. The time lapse from one collision to the next one is just 25 ns so data are stored in pipelines able to withhold and process information coming from simultaneous interactions. To avoid confusion, the detector is designed with an excellent temporal resolution and a signal synchronization from different channels (about 1 million) is achieved.
The trigger system [12, pp. 247-282] is organized in two levels. The first one, called Level-1 Trigger, is hardware-based, synchronous with the LHC bunch crossing time and takes a decision in less than 4 μs. It selects physical interesting data, e.g. an high value of energy or an unusual combination of interacting particles. The next layer, the High Level Trigger (HLT), is software-based, acting asynchronously than the signal reception phase and reduces acquired data up to a few hundreds of events per second.
A more detailed view of the Trigger system involving the muon detectors is provided in the Chapter 2.

1.4 The High-Luminosity LHC project

Since 2010, the LHC has been producing proton-proton collisions with a 7 and then 8 TeV centre-of-mass energy. After a two year long shutdown in spring 2015, the machine has been restarted delivering collisions between protons at a new record of 13 TeV centre-of-mass energy. In order to increase its discovery potential by mid-2020, the LHC would need an upgrade to increase the total number of collisions by a factor of 10. A more powerful LHC will allow even rarer events to be detected and will increase our understanding of the energy frontier.
How this should be done is at the heart of the *High-Luminosity LHC project* (HL-LHC)[19], which design study came to a close on October 2015 with the publication of a technical design report and a budget of 950 million CHF over a period of 10 years. The HL-LHC relies on a number of new technologies, including 11-12 Tesla superconducting magnets, compact superconducting crab cavities with a precise phase control for beam rotation, new beam collimators etc... raising the instantaneous luminosity from $\approx 2.01 \cdot 10^3 4\ cm^{-2}s^{-1}$ up to $7.5 \cdot 10^3 4\ cm^{-2}s^{-1}$.
Figure 1.6 shows the timeline for HL-LHC, currently scheduled to start its data taking in 2026 and to operate through 2030s.

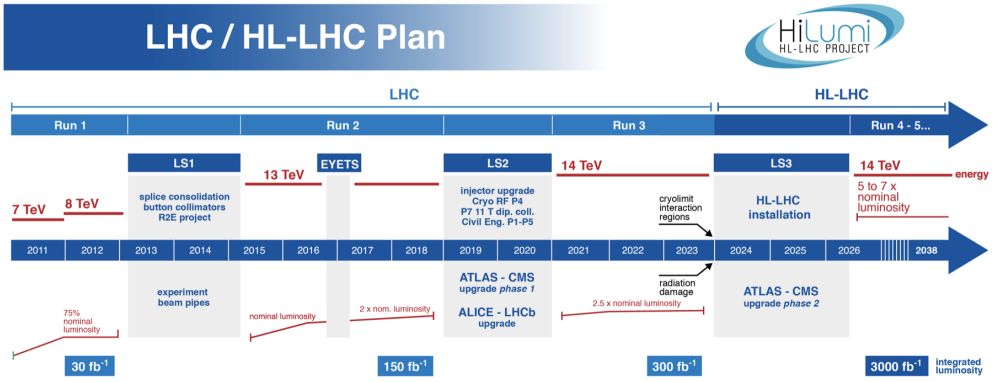


Figure 1.6: Timeline of operations at the LHC for the next years. After a Long Shutdown time period (LS3), the HL-LHC is supposed to start its life cycle.

Chapter 2

The CMS Muon System

The muon (μ) is an elementary particles classified as a lepton, with an electric charge of -1 (+1 for antimuons) and a spin of $\frac{1}{2}$ with an higher mass than the electron, about 105 MeV. During p-p collisions, high energy muons are produced and, crossing both electromagnetic and hadronic calorimeters, they are specifically detected in the far external group of subdetectors of the CMS experiment, the Muon System (Figure 2.1).

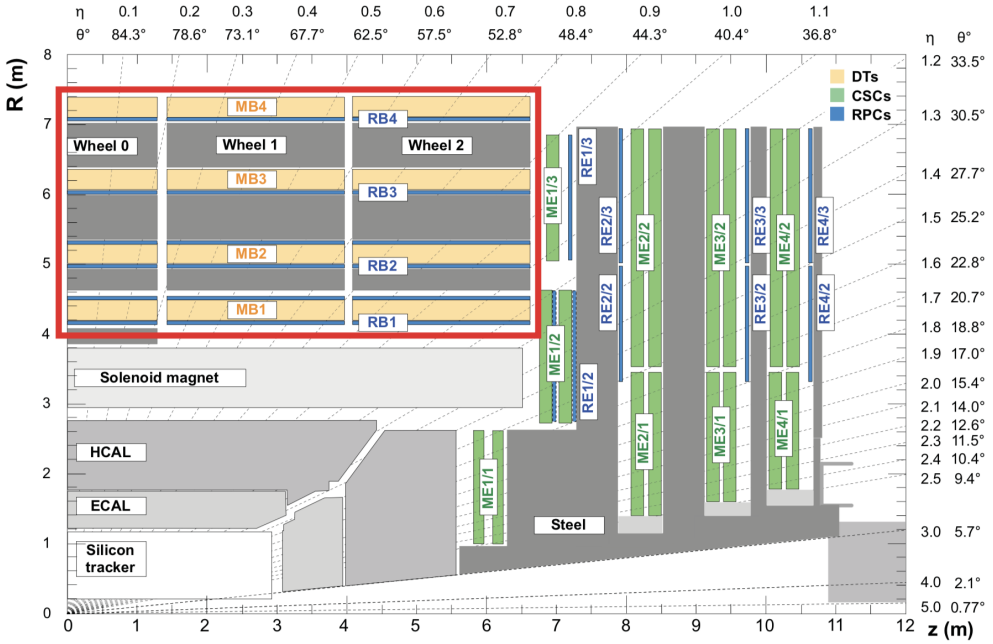


Figure 2.1: A schematic view of the CMS muon system.

2.1 The experimental apparatus

The experimental muon setup [20] consists in three different types of gaseous detectors with a different design, coping with the radiation environment and magnetic field at different values of η:

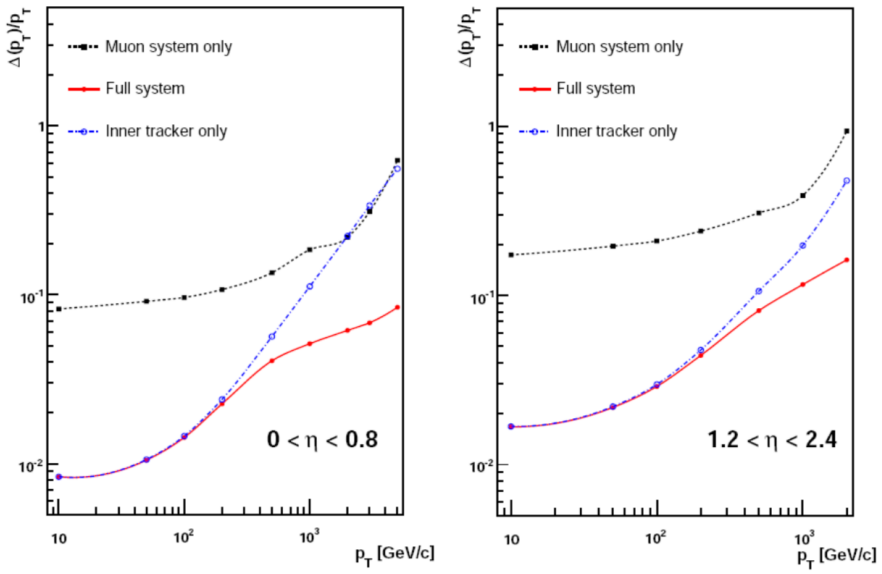


Figure 2.2: Muon transverse momentum resolution as a function of p_T for the inner tracker only (blue dashed line), the muon system only (black dashed line) and the two systems combined (red line). On the left the results for the barrel are shown and on the right for the endcap.

- 250 Drift Tube Chambers (DT), used in the barrel region (with $|\eta| < 1.2$) where a low residual magnetic field is present and track occupancy is low;
- The endcaps ($0.8 < |\eta| < 2.4$) are equipped with 540 Cathode Strip Chambers (CSC) with a faster and radiation resistant capability in order to cope with an higher particle flux and a non uniform magnetic field;
- To ensure redundancy and improve trigger performances, 610 Resistive Plate Chambers (RPC) complement the DT and CSC in both regions up to $|\eta| < 2.1$, due to their fast response and excellent time resolution but low spatial resolution, improving the precision in the muon trigger on the determination of the bunch crossing (BX) in which the muon has been created.

Figure 2.2 shows the total p_T resolution as a function of η, together with the curve obtained using only information coming from the tracker or the muon system only. For muons with $p_T < 200\ GeV$, the tracker precision is the most relevant because of the multiple scattering occurring when crossing the calorimeters and the iron yoke of the muon part, while at higher p_T the combination of the two systems improves the overall resolution.

2.1.1 The Drift Tube Chambers

The barrel region of the muon system is characterized by a low residual magnetic field, low occupancy and a large area to be covered. For this reason, Drift Tube chambers have been employed [21].
The detector management follows the yoke segmentation that consists of 5 iron

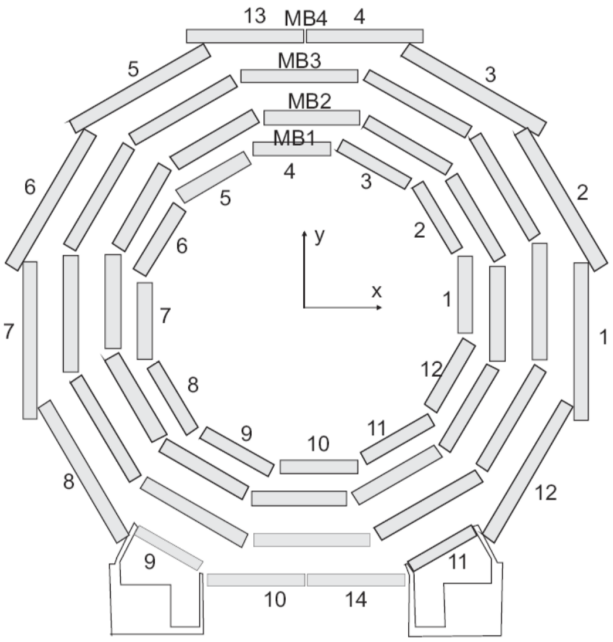
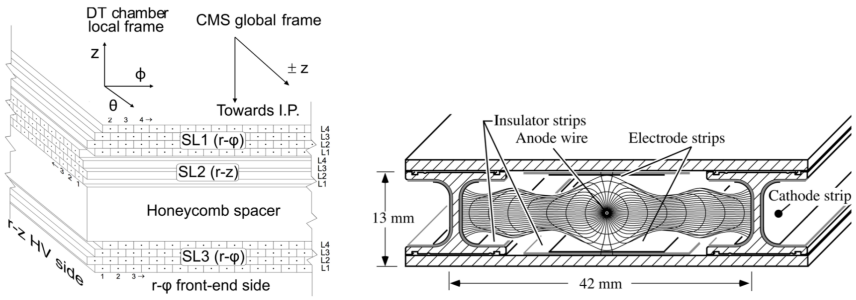


Figure 2.3: Transversal view of the CMS DT system. Station and sector numbers are shown in figure.

wheels composed of 12 *azimuthal sectors*, covering an angular region of $\approx 30^\circ$ each and labelled with numbers 1 to 12, starting from the *x-axis* (Figure 2.3). For each wheel, 4 concentric rings (called *stations*) of DT chambers are installed, named MB1÷MB4 (where MB stands for *Muon Barrel*) respectively from the inner to the outer part. Each station is therefore formed by 12 DT *chambers* with the exception of MB4 which consists of 14 of them.
The basic component of the DT system is the drift cell, shown in Figure 2.4. The cell has a transverse size of 4.2 cm × 1.3 cm and a length that varies from 2 to 4 m. Each cell is equipped with a 50 μm diameter gold-plated stainless steel wire at the center and makes use of 5 electrodes to shape the drift field: the anode wire, 2 cathode strips on the side walls of the tube and 2 strips above and below the wire on the ground planes between the layers, operating at +3600 V, -1200 V and +1800V, respectively, and generating an almost uniform electric field among the cell. The gas used is a mixture of 85%/15% Ar/CO_2 that allows electrons to drift with a saturated velocity of 55 $\mu m/ns$, corresponding to a maximum drift time around 385 ns.
Four staggered layers of parallel cells form a superlayer (SL). A DT chamber consists of 2 SL that measure the ϕ coordinate plus one measuring the z coordinate. The only exception to this rule applies to the MB4 stations where only the two ϕ superlayer are present.
DT cells have an efficiency of 99.8% and a spatial resolution around 200μm, leading to a resolution of almost 100μm for a high quality (i.e. 8 hits) ϕ reconstructed segment.



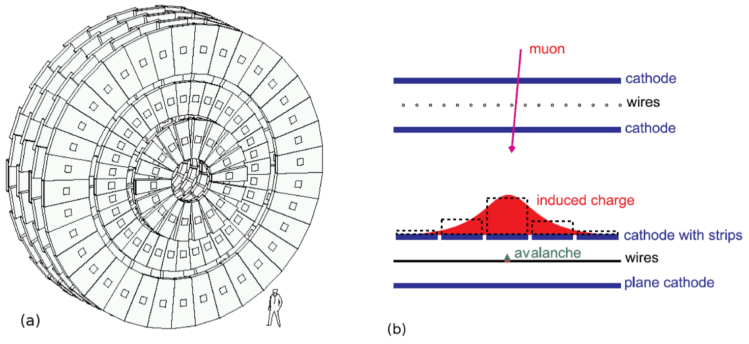
(a) Schematic view of a DT chamber.

(b) Section of a drift tube cell showing the drift lines.

Figure 2.4: DT chamber schematic in the Muon System and a drift tube cell.

2.1.2 The Cathode Strip Chambers

The high magnetic field and particle rate expected in the muon system endcaps does not allow to use drift tubes detectors to perform measurements at large η values. Therefore a solution based on Cathode Strip Chambers (CSC) has been adopted [22]. The CSC are gaseous trapezoidal Multiwire Proportional Chambers (MWPC), characterized by a short drift length which leads to fast signal collection. Information about the position of the incoming particle is collected both in the anode wire and on a group of finely segmented cathode strips. The CSC layout is shown in Figure 2.5. Those chambers are arranged to form four *disks* of concentric *rings* placed in between the endcap iron yokes.
Each chamber is composed by 6 layers of 9.5 mm thick arrays of anode wire enclosed between two planes of finely segmented cathode strips for the collection of the ionization signal produced in the 30%/50%/20% $Ar/CO_2/CF_4$ gas mixture. The wires give information about r coordinate and the strips are used to determine the polar angle. In the first disk, that operates in a region of high magnetic field, the anode wire are tilted by 20° to compensate for the Lorentz drift effect.
The position resolution measured with the strips varies from $\approx$ 70 μm for the innermost stations to $\approx$ 150 μm for the outermost ones, while r can be determined with a precision of $\approx$ 0.5 cm.



(a) Layout of the CSC subsystem.

(b) Schematic overview of a CSC chamber.

Figure 2.5: The Cathode Strip Chambers of the CMS endcap muon system.

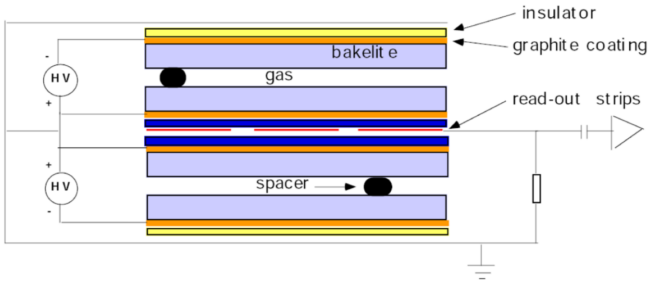


Figure 2.6: Schematic view of a CMS double gap RPC.

2.1.3 The Resistive Plates Chambers

Resistive Plates Chambers are used both in barrel and endcaps, complementing DT and CSC systems, in order to ensure robustness and redundancy to the muon spectrometer [23]. RPC are gaseous detectors characterized by a coarse spatial resolution, but may perform precise time measurements, comparable with the ones provided by scintillators. This ensures precise BX identification to the muon trigger system.
CMS uses double-gap RPC chambers composed by 4 bachelite planes forming two 2 mm gaps, as shown in Figure 2.6, where the crossing particles ionize the 90%/10% mixture of freon ($C_2H_2F_4$) and isobutane (C_4H_{10}), and the electrons are multiplied in avalanche mode. The electrodes, set at a potential of 9.5 kV, are constituted by a graphite coating and the central part of the chamber is equipped with insulated aluminum strips, used to collect the signal generated by the electronic avalanche caused by the crossing particles. The design choice of using double gap chambers is adopted to increase the signal induced on them. In the barrel the strips are rectangularly segmented (12.1 to 41 cm wide and 80 to 120 cm long) and run along the beam axis, whereas the endcaps are equipped with trapezoidal shaped strips covering approximatively the range $\Delta\phi = 5 - 6°$, $\Delta\eta = 0.1$. No measurement is possible in the η coordinate, apart from the constraint imposed by the strip length. In order to sustain higher rates, the detector operates in *avalanche* instead of using the most common *streamer* mode but due to the reduced gas multiplication, improved electronic multiplication is required.
In the barrel region the system, layout follows the DT segmentation and two RPC stations are attached to each side of the two innermost DT chambers of a sector, whereas one single RPC is attached to the inner side of the third and fourth DT chambers. This solution ensures to extend the low p_T range of the trigger system in the barrel, detecting even low p_T muons before they stop in the iron yoke.

2.2 Trigger and Data Acquisition

The LHC collisions have a rate of 40 MHz, which means that two proton bunches intersect in the collision points where the experiments are placed every 25 $ns = 1$ BX. Considering that in the two general purpose experiments ATLAS and CMS each event has a size of about 1 MB, an effective trigger strategy used to select interesting events among all physical interactions, had to be adopted. In order to

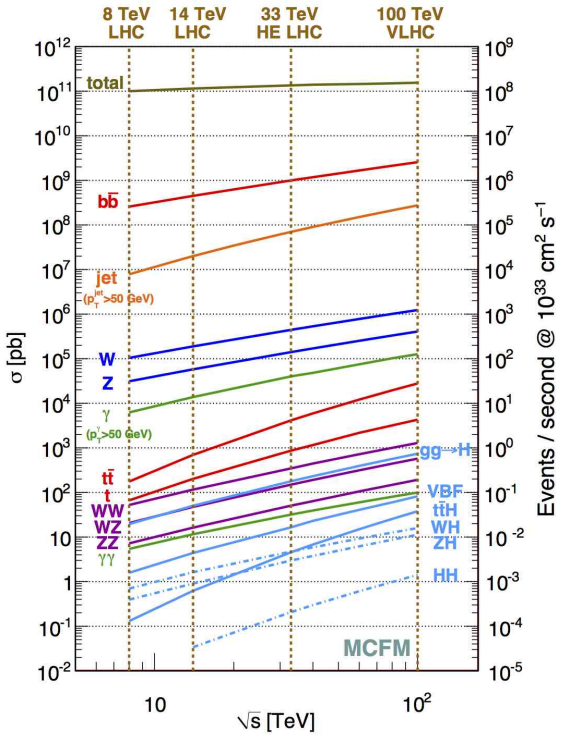


Figure 2.7: Plot showing the different cross sections for the main processes produced at the LHC.

achieve a rate of events up to $\approx$ 1 kHz, imposed by the final storage system, a total reduction rate of 10^7 has to be reached. Despite the high rejection factor, trigger algorithms have to be also quite sensitive to different physical processes and their probability: for instance, processes as $Z \to ll$ and $W \to l\nu$ have cross sections which are orders of magnitude lower than QCD processes, that would saturate selections based on simple high p_T lepton identification (Figure 2.7).
Furthermore, the BX frequency of 40 MHz requires the development of a system able to take a selection/rejection decision every 25 ns. Since this time is too short to collect all the information coming from all the subdetectors and process it in a single step, a pipelined trigger architecture based on different levels of increasing complexity, has been adopted. In CMS there are two steps, nominally the Level 1 Trigger (L1T) and the High Level Trigger (HLT):

- The L1T system [24], built using dedicated hardware electronics, operates a first selection reducing the rate by a factor of 10^4, elaborating signals from the muon detectors and the calorimeters;

- The remaining selection is performed by HLT algorithms [25], on the basis of the full information available from all the detectors using a software system running on a filter farm of a thousand commercial processors, reducing by another factor up to 10^3 and satisfying the storage system requirements.

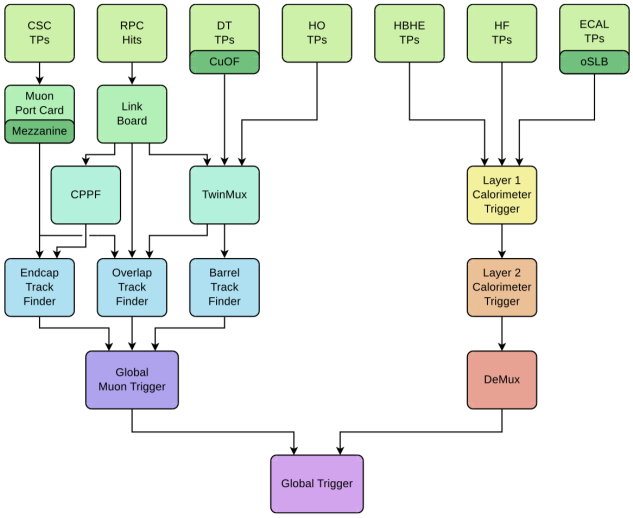


Figure 2.8: Schematic diagram of the CMS Level-1 trigger system.

2.2.1 The Level-1 Trigger System

The Level-1 trigger [24] is designed to accept or reject events at a 40 MHz rate (every BX) in a pipeline mode using custom developed programmable hardware. Field Programmable Gate Arrays (FPGA) are used where possible but also Application Specific Integrated Circuits (ASICs) and Programmable Lookup Tables (LUTs) are taken into account to complete each processing step in less than 25 ns. Due to the maximum length for the pipeline buffers fixed at 128 BX (3.2 μs), taking into account the signal propagation time and the subdetectors latencies, the effective decision operation time for the L1T is less than 2 μs.
The Level-1 Trigger at CMS detector can be further subdivided into three major subsystems: the Muon Trigger, the Calorimeter Trigger and the Global Trigger. The first two systems process informations coming, respectively, from the muon spectrometer and calorimeters and do not have to perform the task of rejecting/accepting events by themselves. On the other hand, they identify and perform sorting on various types of *trigger objects* (electron/photon, jets and muons) and then forward the four best "candidates" of each kind of trigger object to the Global Trigger where the final decision is made, as shown in Figure 2.8.

Level-1 Muon Trigger

The Level-1 Muon trigger is designed to reconstruct muon position and p_T and assign particle origin in terms of BX. In 2013-2014, before Run-II data taking, the Level-1 trigger was upgraded in order to cope with an increase of the total event rate of roughly a factor 6 [24]. Functional relations between the components are shown in Figure 2.9.
Each different region of the detector present its own trigger and each part of the apparatus described in Section 2.1 take part to the trigger process in a redundant and safe mechanism.
The first processing phase of the DT and CSC triggers is elaborated by on-board electronics, operating reconstruction processes on the detector signals. They are

charged of BX assignment, position and direction estimation as well as track quality tagging.
In the original system (Figure 2.9, left), information were then passed to the DT and CSC Track Finders (DTTF - CSCTF),whose job was to build full muon tracks on the basis of local reconstructed primitives, assign the charge sign and estimate their transverse momentum value. They also delivered quality bits related to the number of stations used for reconstruction. In the η region, where muons cross both DT and CSC chambers (the *overlap region*), an exchange between DTTF and CSCTF was performed improving the reconstruction of the final track. The four highest p_T muons coming from both DT and RPC in the barrel, and both RPC and CSC in the end-caps, were then forwarded to the Global Muon Trigger (GMT) for further processing. At last, the resulting trigger were forwarded to the Global Trigger along with the other sub-detectors.

In the upgraded L1 trigger (Figure 2.9, right), the architecture of the electronics devoted to muon tracking follows a geographical partitioning. Separate systems process local information produced in the barrel, endcap and overlap regions, corresponding to $|\eta| < 0.8$, $0.8 < |\eta| < 1.2$ and $1.2 < |\eta| < 2.4$, respectively, to identify muons and measure their coordinates and momentum (see Section 2.1). Following this rationale, the E*ndcap* M*uon* T*rack* F*inder* (EMTF) [26] reconstructs muon trigger candidates using CSC segments and RPC hits. The O*verlap* M*uon* T*rack* F*inder* (OMTF) [27] combines DT and CSC segments and RPC hits from the detector elements belonging to its acceptance region. In the barrel region, a slightly different approach is implemented. DT segments and RPC hits are collected and synchronized by the TwinMux system, which is in charge of the merging of the two, thus delivering improved trigger segments (also called "Superprimitives") to the B*arrel* M*uon* T*rack* F*inder* (BMTF) [28]. Superprimitives combine the very good spatial resolution of DT trigger segments with the superior timing properties of RPC hits, improving the efficiency and the quality of the information used in the BMTF for track identification and momentum measurement.
More details about the TwinMux and the BMTF processors, related to this work, will be given in Section 2.3.4 and 2.4.

2.2.2 The High Level Trigger and DAQ

The CMS High Level Trigger [25] has the task to further reduce the event rate from the L1 Trigger to $\approx 1\ kHz$ required from the storage system. In order to achieve this requirement, the HLT performs an analysis similar to off-line event reconstruction that relies on a farm of commercial processors.
As shown in Figure 2.10, data coming from the Level-1 Trigger are initially stored in a pipelined 40 MHz buffer by the Front End System (FES). Afterwards, data are moved by the Front End Drivers (FEDs) to the Front End Readout Links (FRLs) which are able to get information from two different FEDs. Event fragments coming from different FRLs are then sent to the *Event Builder* system in charge to build up the full event which has two different purposes: transport data from the underground to the CMS surface buildings and than begin the reconstruction phase. After the assembly phase is completed, the event is sent to the *Event Filter*

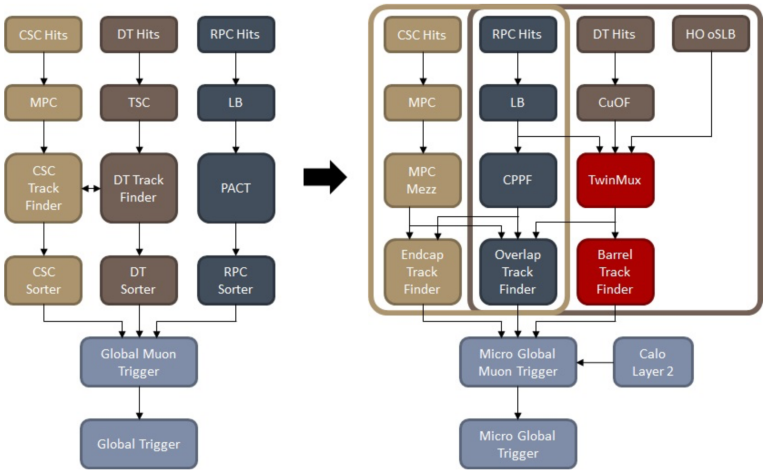


Figure 2.9: CMS Level-1 Muon Trigger data flow. On the left, the old Level-1 Trigger, while on the right, the upgraded Level-1 is shown.

where HLT algorithms, together with some Data Quality Monitoring operations, are performed. Filtered data are then separated into several on-line streams, whose content depend on trigger configuration, and are sent to a local storage system before being migrated to the CERN mass storage infrastructure.

The analysis performed in this thesis is focused entirely on the Barrel Muon Track Finder (BMTF), with segments coming from both DT and RPC.

2.3 The DT Local Muon Trigger

The DT Local Trigger System [29] (or *DT Trigger Primitive Generator* DT-TPG) goal is the detection of charged particles crossing the muon barrel chambers. It has to measure position and direction of the crossing particles as well as their BX origin. It also tags the reconstructed segments with a *quality* word based on the number of layers involved in the measurement and, finally, sends the *trigger primitives* to their matching Track Finder for further analysis.
As shown in Figure 2.11, the signal coming from individual wires are initially

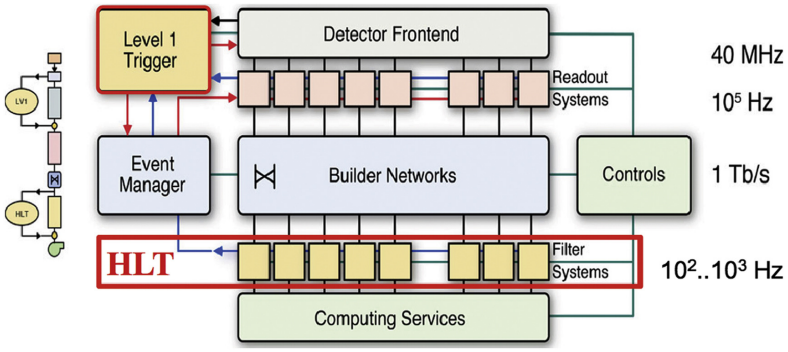


Figure 2.10: Schematic diagram of the CMS High Level Trigger System.

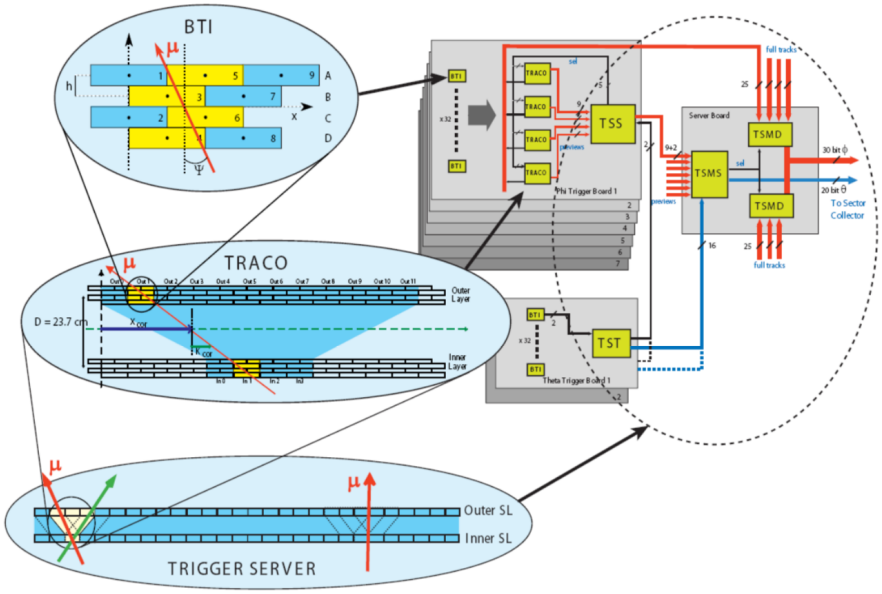


Figure 2.11: Schematic diagram of the CMS DT Local Trigger.

processed by the *Bunch and Track identifiers* (BTIs) that operate a rough track fitting within a single Super-Layer and perform BX assignment. Then, the *Track Correlator* (TRACO) is devoted to match the information coming from the two Super-Layers, improving the parameter measurements. The next step is performed by the *Trigger Server* (TS), whose job is the selection of candidates coming from different TRACO. Finally, the information is passed to the TwinMux card together with the RPC hits, that has to further refine the sorting, perform synchronization on data coming with different latencies and transfer them to the Trigger processors designed for both Barrel and Overlap track finding.

2.3.1 Bunch and Track Identifier

The Bunch and Track Identifier [24] is the implementation of a trigger device based on the *mean-timer technique*, aiming to the identification of the tracks giving signals in at least three out of the four measurement planes.
This method relies on the fact that the particle path is a straight line and the wire positions along the path are equidistant. Considering the drift times of any three adjacent planes of staggered tubes, Figure 2.12, the relation

$$T_{MAX} = \frac{T_A + 2T_B + T_C}{2}$$

where T_{MAX} is maximum drift time to the wire, holds independently of the track impact point (d) and angle of incidence (Ψ). Extending this method to four layers, the identification is possible even if the drift time of a tube is missing (due to inefficiency) or is wrong (due to noise generated by the emission of a δ ray). It also insensitive to uncorrelated single hits and it is therefore well-suited to a high radiation environment.
After the particle's crossing, a signal is induced on the detecting i_{th} wire at a

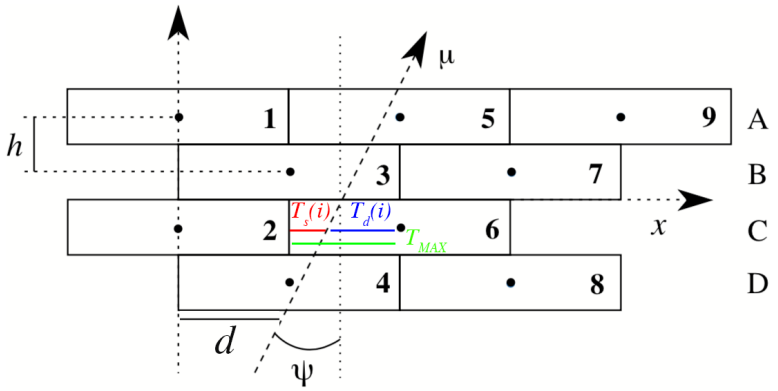


Figure 2.12: BTI geometric layout showing the channels allocations and the important parameters.

time $T_d(i)$ that depends on the drift velocity v_d as well as the distance between the particle trajectory and the wire. On every BTI clock count, the apparent $T_d(i) = T_{MAX} - T_s(i)$ is computed (since the BTI digitizes the time T_s), and a straight line is reconstruction is performed from different pairs of layers. Parameters like the position and the angular parameter $k = h \tan\Psi$ (where $h = 13\ mm$ correspond to the distance between the wire planes) are then evaluated for each pair of layers.
Hence at every clock cycle, a whole set of *k-equations* are computed and a BTI trigger is generated if at least three of the six possible combinations are in coincidence. This coincidence is possible only at a fixed latency T_{MAX} after the particle passage, allowing BX identification. In the case of a four hit alignment, an *High Quality Trigger* (HTRG) is raised, while in any other case, with a minimum of three hits a *Low Quality Trigger* (LTRG) is found. However there might be a possibility that the alignment of four hits at some clock step produces a LTRG just before or after the correct HTRG: a so called *ghost* trigger candidate. The noise reduction of ghosts is obtained issuing the LTRG signal only if at the neighbouring steps there is not a HTRG generated: this mechanisms is called *Low Trigger Suppression* (LTS).

2.3.2 The Track Correlator

The BTI is followed in the electronics chain by a Track Correlator (TRACO) [30]. It is required to associate portions of tracks in the same chamber relating groups of BTIs among them. The TRACO links together the two SLs of the ϕ view, receiving information from the BTI connected to it and trying to find the couple of BTI track segments that fits the best track (linking the inner layer candidates to the outer layer ones). The introduction of this device is necessary mainly for two reasons:

- The BTI is a relatively noisy device and requires a local pre-selection and a quality certification;
- The number of BTIs per channel is around few hundreds and it is not possible to connect together all the channels to perform any pre-selection at chamber

Table 2.1: TRACO quality code conversion table.

TRACO code	Quality label	meaning
6	HH	Correlated H+H trigger (outer and inner SL)
5	HL	Correlated H+L or L+H trigger
5	LL	Correlated L+L trigger
4	H_O	Uncorrelated High trigger (outer SL)
3	H_I	Uncorrelated High trigger (inner SL)
2	L_O	Uncorrelated Low trigger (outer SL)
1	L_I	Uncorrelated Low trigger (inner SL)
7	/	Null Track

level.

The current design connects four BTIs of the inner SL to twelve BTIs on the outer SL as shown in Figure 2.13. The total number of TRACOs per chamber depends on the size of the station (usually from 12 to 24 TRACOs are needed). In the θ view, since there is only one detecting SL per station, no TRACO operation is performed forwarding data directly to the Trigger Server (TS).
The algorithm of the Track Correlator starts by selecting, among all candidates in the inner and outer SL independently, the best track segment according to the trigger quality (High or Low) and to the track proximity to the radial direction compared to the vertex (i.e. its p_T). Then, it computes the k-parameter and the position of a correlated track candidate, using the relations:

$$\begin{cases} k_{COR} = \frac{D}{2} \tan\psi = x_{inner} - x_{outer} \\ x_{COR} = \frac{x_{inner}+x_{outer}}{2} \end{cases}$$

Due to the long arm between the two SLs, the angular resolution of a correlated track is 10 *mrad* which is significantly improved compared to the BTI resolution. The resolution on the position remains unchanged.
Using programmable LUTs, the computed parameters are then converted to the chamber reference system: position is transformed to radial angle ϕ and k-parameter to *bending angle* $\phi_b = \psi - \phi$ (as shown in Figure 2.14). The chosen track is forwarded to the Trigger Server, for further selection. If the correlation fails, the Track Correlator forwards an uncorrelated track following a preference list that includes both the parent SL (IN/OUT) and the quality bit (H/L) of the two tracks selected for correlation. When no correlation is found, since there is no candidate in one SL, the uncorrelated track is anyway forwarded.

2.3.3 The Trigger Server

The Trigger Server (TS) [24] is the last on-board component of the DT Local Trigger logic and has to select the best trigger candidates among the track selected by all TRACOs in a muon station and sends them to the Sector Collector, where they will be forwarded to the Regional Muon Trigger.

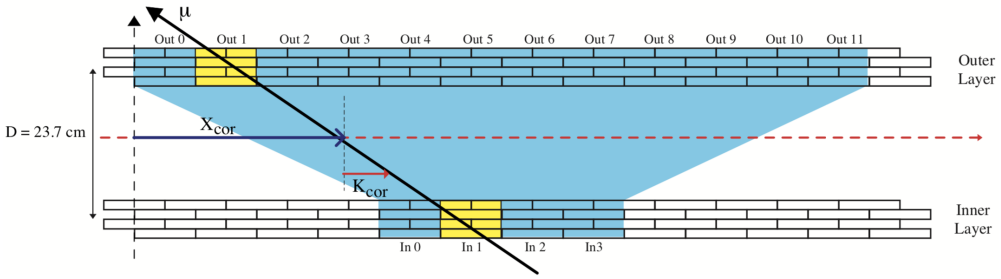


Figure 2.13: Track Correlator (TRACO) geometrical layout.

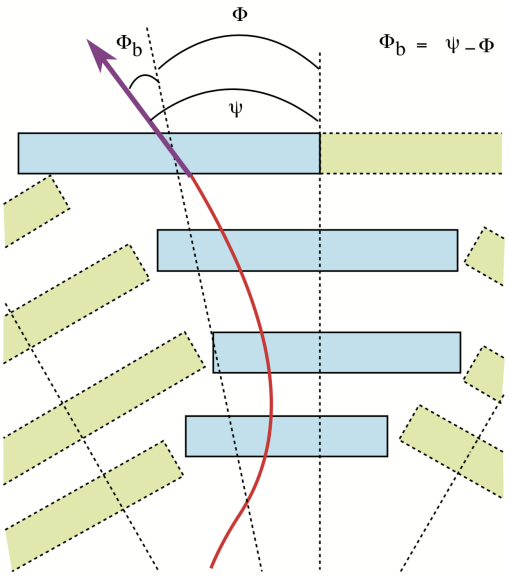


Figure 2.14: Definition of the angles transformed by the TRACO.

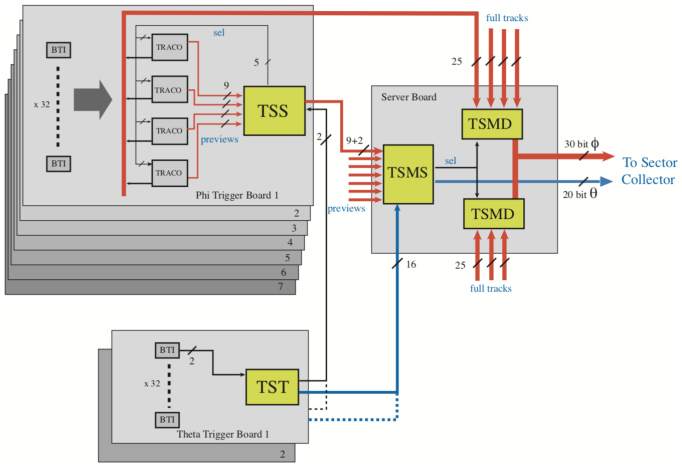


Figure 2.15: Internal architecture of the Trigger Server board.

Each muon station is equipped with only one TS and, for this reason, needs to be extremely robust and redundant. The TS has to fulfil the following requirements:

- High efficiency and purity of the selected segments: a lot of interesting physics has two close muons that can hit the same muon station therefore the selection should be based on both the bending angle and the quality of the track segment;
- Ghost rejection generated by TRACO using a configurable fake rejection algorithm;
- Processing time independent on the number of TRACOs in the station;
- Capability to treat several pile-up events;
- Requires a built-in redundancy since there is only one TS on each station, resulting in a bottleneck of the on-chamber trigger.

In order to fulfil these requirements (keeping at the same time the number of logic cells as low as possible) the TS is composed by two subsystems: one for the transverse view (TSϕ) and the other for the longitudinal view (TSθ). In particular, the TSϕ is implemented in a two level structure, as shown in Figure 2.15: the *Track Sorter Slave* (TSS) and the *Track Sorter Master* (TSM). Each TSS processes up to 4 data word per BX coming from multiple TRACOs then they are sent to the TSM for final processing. The θ-view of the TS, instead, is equipped with two identical units (TST): their task is to group together data coming from up to 8 BTIs, performing a logical OR on the information received in order to prepare the θ-view trigger for regional processing.

2.3.4 TwinMux

In the new muon trigger chain, the adaptive layer for the track finder in the barrel region is called *TwinMux*. Each TwinMux processor receives DT and RPC links

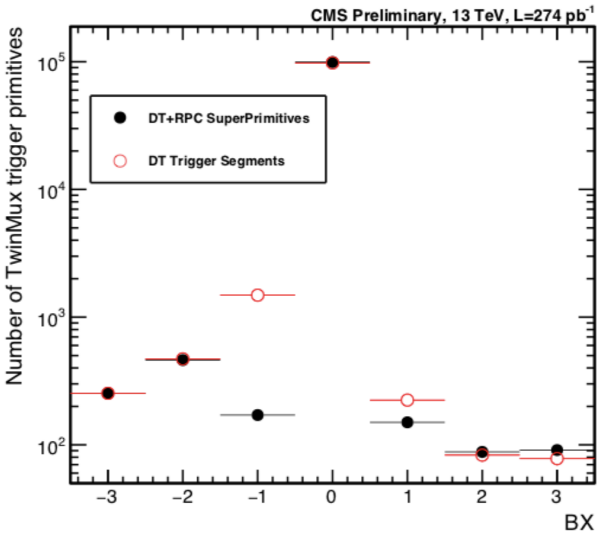


Figure 2.16: Bunch crossing distribution of the trigger primitives. In open red circles an "only DT" trigger segment scenario is shown; in solid black circles, the "DT+RPC" Superprimitive scenario is shown [31].

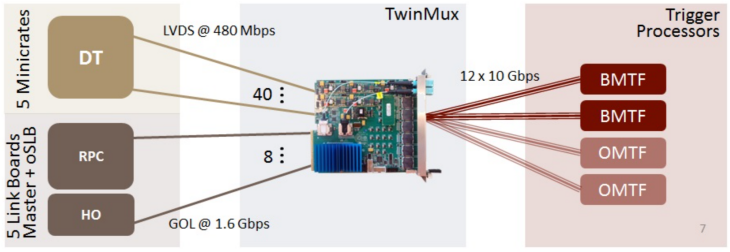


Figure 2.17: DT and RPC trigger chain. Notice that high speed links connect the TwinMux to the Trigger Processors.

from one sector of the barrel muon detector. It is responsible for bringing forward the merging of the DT, RPC and HO (HCAL Outer barrel) trigger primitives, unburdening the trigger processors. At first, input data are deserialised and synchronized, then a clustering algorithm is applied to RPC hits i.e. neighbouring hits are merged into one cluster position.
An important feature is the search of an RPC cluster in a time window of ±1BX around the DT Trigger segment [31]. This combined solution improves the number of Superprimitives in BX=0, as shown in Figure 2.16, with an average increase of the efficiency of about 1.4%.
It is then in charge of sending such combined primitives to the BMTF and, in addition, also the RPC and DT data are sent to the OMTF. Another task of the TwinMux is the duplication of the trigger primitives in order to reduce connections between trigger processors, increasing the reliability of the system.
In Figure 2.17 the trigger chain of the combined DT and RPC trigger data is shown.

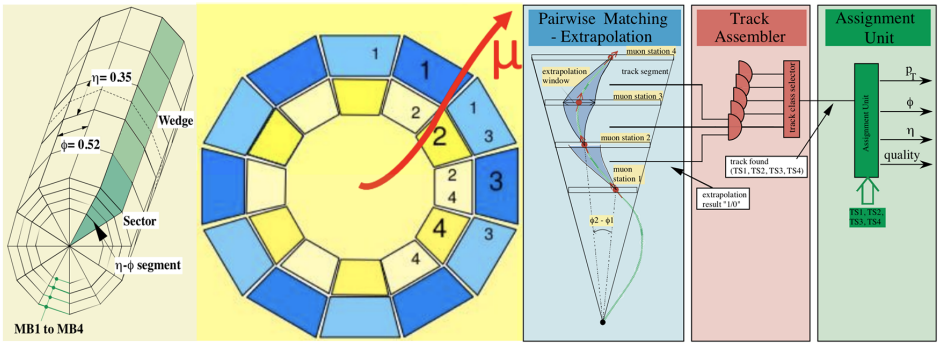


Figure 2.18: The BMTF algorithm.

2.4 The Barrel Muon Track Finder

The muon barrel architecture groups the barrel muon detectors in 12 wedges. Each wedge has five sectors and each sector four DT stations and three RPC planes. The front-end electronics generate the muon primitives and send the data to the TwinMux. The TwinMux combines DT and RPC data to create better quality candidates (*Superprimitives*) than the primitives of the DT. Then it fan-outs the Superprimitives to the Barrel Muon Track Finder (BMTF) [28] and Overlap Muon Track Finder (OMTF).
The BMTF trigger receives the muon Superprimitives from the Barrel area of CMS ($|\eta| < 0.85$) and uses this information to reconstruct muon tracks and calculate the physical parameters: transverse momentum (p_T), the ϕ and η angles, the quality of the candidate and the track addresses. The algorithm has three stages:

- The first stage is performed by the *Extrapolator Unit*, which combines the track stubs, bending angle and quality bits from different DT stations of one sector. Each combination extrapolates the muon candidate to the next station and checks whether it is within an acceptable window. The same algorithm is repeated between neighbour sectors;
- The second stage is performed by the *Track Assembler Unit* that links the segment pairs to the full track and assign it with a quality word related to the number and type of stations involved in the reconstruction;
- Finally, the *Assignment Unit* uses LUTs in order to assign the p_T physical parameter to the algorithm output.

In Chapter 4, the p_T produced by the Machine Model learning models, will be compared with the p_T assigned after the output produced by the Barrel Muon Track Finder (in the *Assignment Unit*).
In Figure 2.18 the entire process of track finding is shown. The algorithm run in parallel for all the sectors of one wedge in the same card. Each one of the twelve processing cards delivers the three best muon candidates to the μGMT.
The μGMT then processes the muons for the Barrel, Overlap and Endcap Muon Track Finder.

Chapter 3

An introduction to Machine Learning

The name *machine learning* was coined in 1959 by Arthur Samuel [32] and, over the past two decades, Machine Learning (ML) has become one of the pillars of information technology and quite adopted into daily technologies: users all around the globe, each day, use a learning algorithm without even knowing. Every time a user performs a Google search or watches a video on YouTube or Netflix, a complex structure of learning algorithms records all our choices and preferences in order to build a customised environment. Email providers also adopt spam filters that are the result of hidden ML algorithms and an increasing improvement of *pattern recognition* techniques allows social networks (Facebook, Instagram etc...) to identify different people in photos but also to create 3D images of human tissues and organs for medical diagnosis.
Currently, the rise in learning algorithms in many sectors is mainly related to an increasing quantity of data available combined with a technological progress in storage and computational power, together with lower maintenance and material costs.

3.1 Types of Machine Learning

There are several types of learning algorithms. The main two types are called *supervised* learning, *unsupervised* learning and *reinforcement* learning.

3.1.1 Supervised Learning

In this specific type of learning algorithm, as the word *supervised* suggests, the machine is learning a function that maps an input to an output based on provided input-output pairs. In other words, we provide along with the input *features* also a *target* variable which contains the information that the machine has to learn during its training.
In order to understand this concept, a brief example is shown [33]. Let's consider a dataset containing the US housing prices with a summary plot shown in Figure 3.1. In this example we have the price of the house on the vertical axis and the size

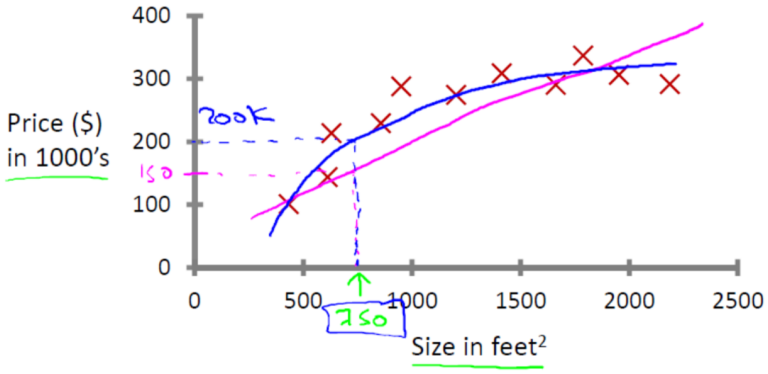


Figure 3.1: US housing prices example. The information is represented by the red crosses while the prediction curves are plotted with coloured lines.

of the house in the horizontal axis. In order to make a prediction for a 750 $feet^2$ house, a plausible learning algorithm might put a straight line through the data (pink line in Figure 3.1). In this case, the corresponding price would be $150k\$$. However, this is not the only algorithm we could use: instead of a straight line (linear regression), a n-order polynomial curve may be used (blue curve in Figure 3.1). Each possible curve with a different prediction on the housing price.
This example is supervised because the algorithm is provided with a dataset containing the "right" answers, i.e. the actual pricing for different sizes. This problem is also called "regression" because we are trying to predict a continuous output value (the housing prices in this particular example). On the other hand, if the prediction is discrete (between 0 and 1, or signal/background in a physical way), this becomes a so called "classification" problem, like an email spam filter which has to decide if a message is or is not fraudulent (Figure 3.2.
Visual recognition is an application domain that highly relies on supervised ML algorithms. For instance, a system might need to learn to identify pedestrians on a street in a automotive application for self-driving cars: to do so, it is trained with millions of short videos about street scenes, with some of the videos containing no pedestrians at all while others having up to dozens. The presence or not of pedestrian is known a-priori, hence the learning is supervised: a variety of learning algorithms are trained on such data, with each having access to the correct answer.
Sorting algorithms at large are also an example of a supervised ML application domain. Suppose that a robot needs to learn how to sort objects based on its material (metal, glass, wood) using visual identification on a conveyor belt. With each item labelled, the training allows the system to learn how to perform the selection a-priori and not via real time rewards (differently from a reinforcement learning system, discussed later on this Section): the more training data are available, the better the sorting algorithm will perform.
Many other decision support mechanisms that have at disposal large quantity of data on which to train a ML system, could be the base of a supervised ML approach: e.g. historical data on medical exams' output may drive a supervised ML system to learn and prompt the probability of suffering from a disease, etc.

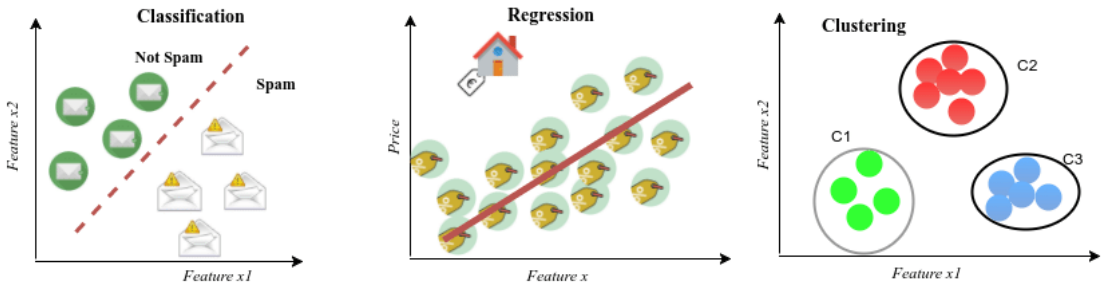


Figure 3.2: Examples of real-life problems in the context of supervised and unsupervised learning tasks: Spam filtering as a classification task and House price estimation as a regression task are part of supervised learning; Clustering is part of unsupervised learning in which customers are grouped into three different categories based on their purchasing behaviour (image from [34]).

3.1.2 Unsupervised Learning

With respect to the supervised case, in an unsupervised learning algorithm we provide the computer with an input dataset but no corresponding output values with the goal of underline the possible structure of data and therefore learn the possible outcomes. Given this dataset, an unsupervised learning algorithm may decide to perform a clustering selection, as shown in Figure 3.2.
Clustering algorithms like this one are used in many web-based application such as newspaper websites. Using thousands of stories as input data, the learning algorithm automatically cluster the information together based on topic or user preferences, organizing information in a coherent and customizable way.
Unsupervised learning is used also for a bunch of other applications. It is used to organize large computer cluster, deciding which machines tend to work together in order to put those machines together and increase the overall efficiency. Another application is on social network analysis identifying between hundreds of friends the closest one, based on private communications or recent social activities.

3.1.3 Reinforcement Learning

Reinforcement learning is an important type of Machine Learning, in which an agent learns how to behave in a environment by performing actions and deciding about the next actions based on the outcome of the previous ones. In recent years, a lot of improvements were observed in this area of research. Most popular examples include DeepMind and the Deep Q learning architecture (dated 2014), which culminated in the media when the champion of the game of Go was beaten by AlphaGo in 2016.
Reinforcement learning can be understood using the concepts of Agents, Actions, Environments, States, and Rewards (note that this description is intended to be short and provided just for completeness in describing possible ML types, and no deep discussion of its characteristics and implementation will be given, as it would go beyond the research scope of this thesis).

- An "Agent" is the component that takes actions, e.g. a video game character navigating in its virtual environment, as well as a drone making a delivery.

- An "Action" is one amongst the set of all possible moves/choices the agent can make. In a reinforcement learning environment, agents can choose only among a predefined list of possible actions. E.g. in video games, the list might include moving right or left, jumping or not, jumping high or low, or crouching, or standing still; in the stock markets, the list might include buying, selling or holding any title among a list of financial product.

- The "Environment" is the world through which the Agent moves. The environment takes as input the Agent's current State and its selected Action as input, and returns as output the Agent's Reward and next State. E.g. in a model in which the Agent would be myself, the Environment could well be the entirety of the physics laws, plus the rules of the society I live in and within which filter all my Actions would be processes and the consequences of them determined.

- The "State" is a concrete situation in which the Agent happens to put itself. E.g. it could be a specific moment in time and/or place in space, a local and instantaneous configuration that puts the Agent in contact and relation with its Environment (e.g. tools, obstacles, prices).

- The "Reward" is the feedback based on which you measure the success or failure of the Agent's choices. E.g. in a video game, a reward could indeed be the gain of a price when a special object is captured, and similar. Every time an Agent does something in the Environment that foresees a possible Reward, the Agents sends output in form of Actions to the Environment and the Environment returns the Agent's new state as well as the obtained Rewards (or lack of them).

In a nutshell, reinforcement learning judges actions by the results they produce. It is fully goal oriented, as its aim is just to learn sequences of actions that will eventually lead an Agent to achieve a predefined goal, in terms of maximizing its objective function. In the video game example, the final goal is to finish the game with the maximum score, so each additional point obtained throughout the game will affect the Agent's subsequent behaviour (i.e. an Agent might learn that making some combination of Actions will lead to a point to it will try to redo the same actions effectively in case a similar situation will be offered next in the Environment). In a robotics example, a robot might have as a goal to travel from A to B: every millimetre the robot gains that makes it closer to the spatial objective B would be counted as additional Reward, so the robot will learn the direction to go and eventually reach the final destination.
The implementation of reinforcement learning models requires a lot of data. For this reason, it has historically been associated with domains in which plenty of simulated data is available (e.g. video games and robotics, as in the examples above). One other aspect to mention is that - with respect to other ML types - it is far from easy to take results from academic research papers and implement them in real-world applications. Also, the results reproducibility is challenging, and this is a key issue: as ML gets deployed in mission-critical situations, the reproducibility and the ability to estimate errors become essential, and this aspect is particularly

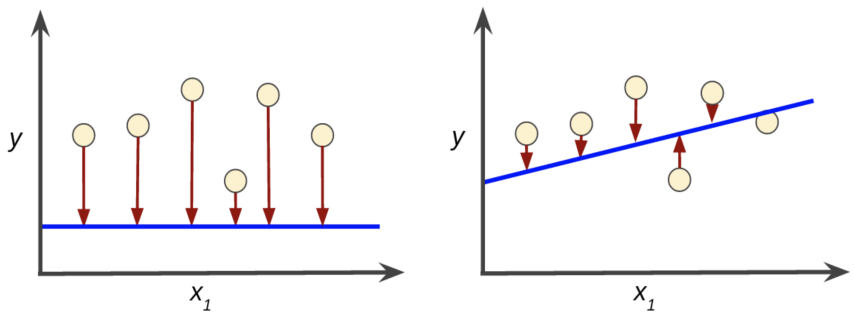
important for reinforcement learning.
Nowadays, applications of RL can be seen e.g. in high-dimensional control problems, like robotics: they have been the subject of research (in academia and industry) for many years, and now start-ups are beginning to use this ML type to build products for industrial robotics applications.

3.2 Supervised Learning in more detail

The Machine Learning algorithms used in this thesis belong to the category of *supervised* learning, therefore a more detailed description is required [35]. Using this particular approach, a model is *trained* on a predefined set of *training examples*, which allows future predictions using new and unseen data.
Suppose we have a set of known inputs (also called *features*) x_i each with a certain weight w_i. In order to infer a quantity y, we define a *predictor* $y' = h(x_i)$ defined as:

$$y' = b + w_i \cdot x_i$$

with b as a given bias. Training a model simply means learning (determining) good values for all the weights and the bias from labelled examples. In a supervised learning, the algorithm builds a model by examining many examples and attempting to find a model that minimizes loss.
Loss is the penalty for an incorrect prediction: is a number indicating how bad the model's prediction was an example in the training set. Hypothetically speaking, if the model's prediction is perfect, the corresponding loss would be zero. In Figure 3.3, two different models are shown: Figure 3.3a shows a model with high loss and Figure 3.3b one with low loss. The red arrows in the left plot are much longer than their counterparts in the right plot. In fact, the blue line in the right plot is a much better predictive model than the blue line in the left plot.
The next question now becomes the possibility to create a mathematical function - a loss function - that would aggregate the individual losses in a ordered way.



(a) Model example with high loss. **(b)** Model example with low loss.

Figure 3.3: Different models for a supervised Machine Learning algorithm. In the left model, an high loss is obtained while in the right model, a lower loss is achieved. Red arrows represent loss while the blue line represents predictions.

Squared loss

The linear regression models, use a loss function called **squared loss**. It is defined as follows:

$$(\text{observation} - \text{prediction}(x))^2 = (y - y')^2$$

An example is the *mean square error* (MSE): the averaged squared loss per example over the whole dataset. To calculate the MSE, sum up all the squared losses for individual examples and then divide by the number of examples:

$$\text{MSE} = \frac{1}{N} \sum_{(x,y) \in D} (y - \text{prediction}(x))^2$$

where:

- (x, y) is an example in which:
  - x is the set of features that the model uses to make predictions.
  - y is the example's label (the quantity to predict).
- prediction(x) is a function of the weights and bias in combination with the set of features x.
- D is a dataset containing many labelled examples, which are (x, y) pairs.

Mean squared error is not the only metric used in Machine Learning, but it is commonly-used especially in regression problems. For instance, the *mean absolute error* (MAE) is another loss function defined as follows:

$$MAE = \frac{1}{N} \sum_{(x,y) \in D} |y - \text{prediction}(x)|$$

In section 4.3.4, the MSE metric will be used as a scorer for the created models.

3.2.1 Training the model

The model takes one or more features as input and return one prediction (y') as output. Initially, all the weights are randomly set and, at each iteration, the value of the loss function generates new values for w_i. This learning procedure continues until the algorithm discovers the model parameters with the lower loss possible. When the loss stops changing (or changes really slow), the model has *converged*. Once the model has converged, the loss function reaches a minimum for a given weight value w_i. Therefore, this kind of regression problems will have a convex plot of loss vs. weights. However, calculating the loss function for every value of w_i is an inefficient way of finding the convergence point. Another technique - quite common in machine learning - is called *gradient descent*. After choosing a starting value for the weights, typically random, the gradient descent algorithm calculates the gradient of the loss curve at the starting point. Mathematically speaking, a vector of partial derivatives with respect to the weights, is computed and in case

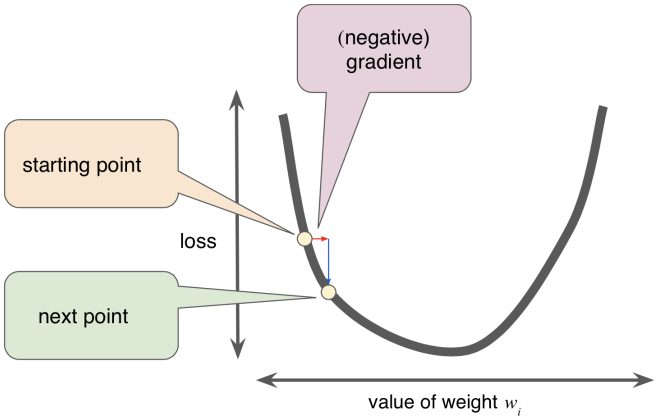


Figure 3.4: Schematic representation of the gradient descent mechanism (image from [35]).

of a negative gradient, the algorithm performs the next step (shown in Figure 3.4, edging closer to the minimum.
Gradient descent algorithms multiply the gradient by a scalar value known as *learning rate* (or *step size*, one of the model's *hyperparameters*) to determine the next point. Tuning the learning rate is a tricky problem: if a learning rate too small is selected, the learning procedure will take too long; conversely, if it is too large, the next point will exceed the minimum weight value and the correct value will never be reached.
In case of loss function minimisation with gradient descent, when large volumes of data come into play, other parameters in the gradient descent strategy become very relevant for a fast convergence of the minimisation algorithm towards finding the global minimum, e.g. the gradient descent *batch size*: it corresponds to the total number of examples you use to calculate the gradient in a single iteration. By default, the batch matches the entire dataset. However, datasets may often contain millions and millions of examples as well as a huge number of features. In these cases, a batch can be enormous and also the corresponding computing time. Smaller batch sizes are therefore suggested: in the *stochastic gradient descent* (SGD), for instance, the batch size is brought to one single example per iteration. Given enough iterations, SGD works but is very noisy.
An intermediate approach among batch GD and SGD is the so called mini-batch GD: the original training set is subdivided into smaller batches (usually of the order of 10, but can be much more depending on the size of the dataset) - these are chosen at random and allow parallelization of the entire process, which might be extremely useful in case of heterogeneity of hardware devices available for training - and will ultimately provide GD algorithm convergence in shorter time.

Overfitting

A common problem with Machine Learning is *overfitting*. This particular issue occurs when a trained model - e.g. based on high-order polynomial hypotheses - matches the training dataset in a very precise way, thus achieving a high accuracy on that dataset but a poor generalisation capability over new, previously unseen

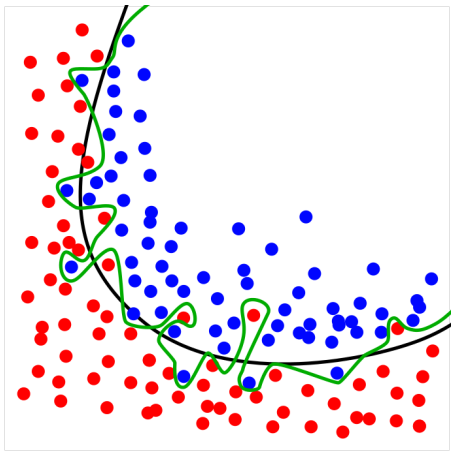


Figure 3.5: Example model that overfits data . In black a model for the data classification is shown. In green an overfitted model is shown.

data. In Figure 3.5, an example is shown. In this example, the red and blue data, from the positive and negative classes of a binary classification problem, are trained with two different models: the black line and the green line. The first model, is quite simple (with some misclassification, of course) and works quite well for new-untrained data. However, the lowest loss possible might be computed using a complex cut, as shown in the green line. In that case, the loss is the lowest possible but the ability of such model to predict data with an entire new dataset decreases and predictions are no longer reliable.
Therefore, the less complex a Machine Learning algorithm is, the more likely an empirical result is not caused by peculiarities of the sample.

3.2.2 Training and Test set

A Machine Learning model aims to make good predictions on new, previously unseen data. However, when the model is built starting from a specific dataset, there are no remaining examples with new unseen data. There are 4 different techniques in order to fix this common problem for a Machine Learning study:

1. Split into a Training and Test Set;
2. *k-fold* Cross-Validation;
3. Leave One Out Cross-Validation;
4. Repeated Random Test-Train Splits.

Training and Test split

This algorithm evaluation technique is very fast. It has the following pros and cons:

- Pros: It is ideal for large datasets (hundreds of thousands of records). Splitting a large dataset into large sub-datasets allows that, first, each split of the

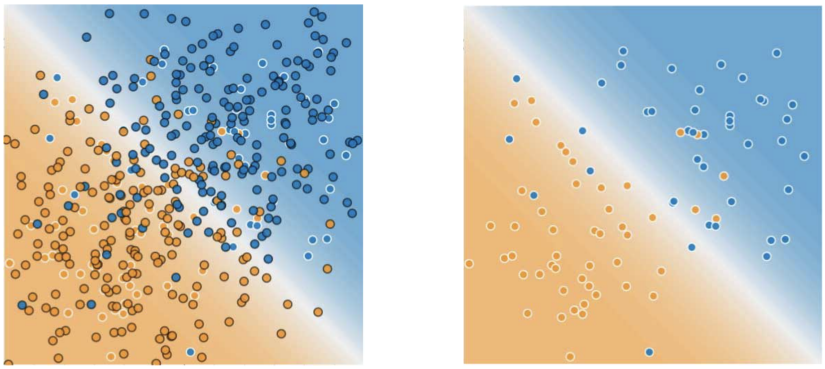


Figure 3.6: Example showing the splitting between training and test datasets. On the left side the training dataset while, on the right side, the test dataset used for verifying the trained model (image from [35]).

data is not too tiny, and second, both are representative of the underlying problem. Because of the high speed, it is useful to use this approach when the algorithm you are investigating is slow to train.

- Cons: A downside of this technique is that it can have a high variance. This means that differences in the training and test dataset can result in meaningful differences in the estimate of accuracy. Furthermore, in order for this split to work, the test set has to be representative of the dataset as a whole, i.e. the test set should not have different characteristics than the training test.

In Figure 3.6, the model learned from the training data is quite simple. This model is not perfect, since a few predictions are wrong. However, this model does about as well on the test data as it does on the training data. In a way, using this particular model, overfitting does not occur.

k-fold Cross-Validation

K-fold cross-validation is another approach that can be used to estimate the performance of a ML algorithm with less variance than a single train-test set split. The data set here is divided into k subsets (called "folds"): for each fold, one of the k subsets is used as the test set and the other $k-1$ subsets are put together to form the training set (see Figure 3.7). Then, the average error across all k trials is computed, together with the standard deviation.
This has the following pros and cons:

- Pros: This method is less sensitive to how the data division happens to be made: every data point gets to be in a test set exactly once, and gets to be in a training set k-1 times. Therefore, the variance of the resulting estimate is reduced as k is increased.

- Cons: The disadvantage of this method is that the training algorithm has to be rerun from scratch k times, which means it takes k times as much computation to make an evaluation, thus resulting evidently slower.

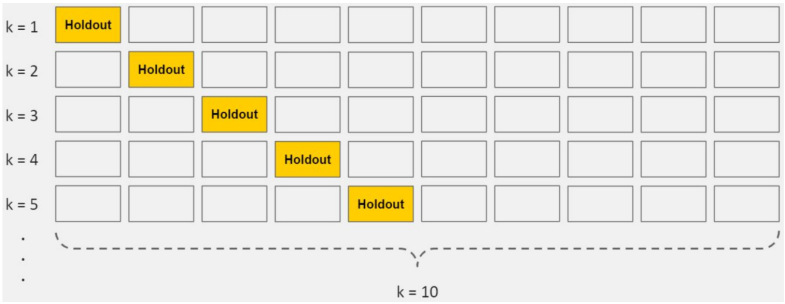


Figure 3.7: Pictorial representation of 10-Fold Cross Validation [36].

Leave One Out Cross-Validation

The *Leave One Out Cross-Validation* is a simplified version of the k-fold where the size of the fold is 1 (i.e. k=n). The result is a large number of performance measures that can be summarized in an effort to give a more reasonable estimate of the accuracy of your model on unseen data. An evident downside of this procedure is the computational power which is much more expensive than k-fold CV.

Repeated Random Test-Train Splits

Another variation on k-fold cross-validation is to create a random split of the data like the train/test split described above, but repeating multiple times the process of splitting and evaluation of the algorithm , like a cross-validation. This has pros and cons:

- Pros: it has the speed of using a train/test split and the reduction in variance in the estimated performance of k-fold cross-validation. It is also possible to repeat the process many more times as needed to improve the accuracy.
- Cons: repetitions may include much of the same data in the train or the test split from run to run, introducing redundancy into the evaluation.

In the scope of this thesis, a training a validation split sets was used due to the large dataset available for analysis (see Section 4.3.2).

3.2.3 Validate the model

In the previous topic, the concepts of training and validation datasets are introduced in order to perform a check on the trained model. This workflow is repeated many times: each time a parameter is changed i.e. the learning rate or the addition of an extra feature. At the end, the best model is used for the test set. Another way to do this, reducing the chances of overfitting, is the partition of the dataset into three subsets: the *training set*, the *validation set* and the *test set*. With this data splitting, the validation set is used to evaluate results coming from the training set. Then, the test set performs a double-check on the evaluation after the model has passed the validation set. In Figure 3.8, an example of workflow is shown: the training set is firstly checked by the validation set, which provides the model's optimal parameters for another cycle of training; then, the best model is tested again using the testing subset to obtain a reliable model.

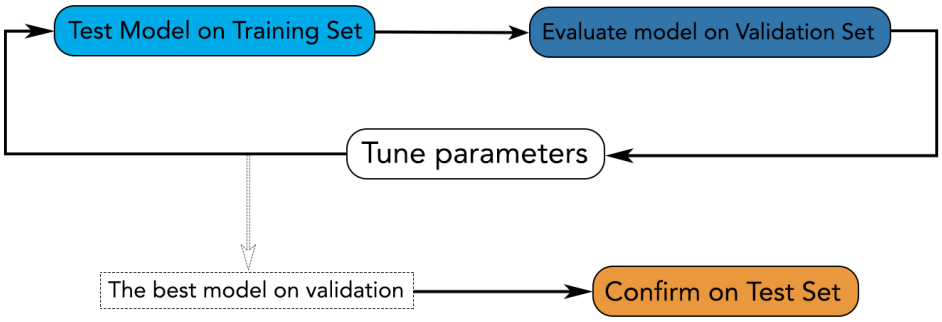


Figure 3.8: Workflow using a validation and test subset.

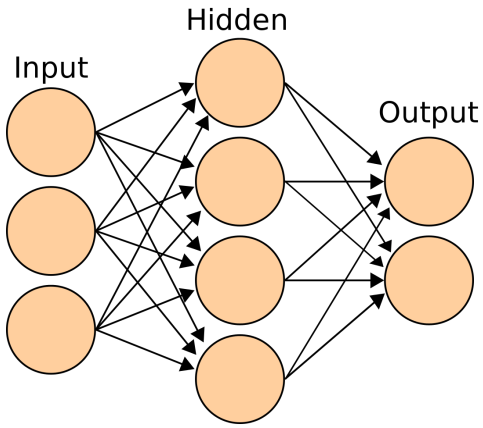


Figure 3.9: Neural Network basic working principle.

3.3 Artificial Neural Network

An artificial neural network (ANN) is a computing system vaguely inspired by the biological neural connections that constitute a human brain, specificately designed to non-linear learning problems. Currently, ANNs are one of the main tools used in Machine Learning [37]. The schematic working principle is described in Figure 3.9.
Each input feature is represented as a *neuron*, creating an *input layer*: the output(s) are then computed by a weighted sum of the inputs, similarly to other learning algorithms. The main difference between Neural Networks and other algorithms relies in the addition of other layers of neurons between them, called *hidden layers*.

Hidden Layers

In the model shown in Figure 3.9, the hidden layer is located between the input and the output layers. Each neuron of the hidden layer is a weighted sum of the input values and the output is a weighted sum of the hidden neurons as well. Despite this complex network, the model is still a linear combination of its inputs, independently on the number of hidden layers added to the model.

Activation Function

In order to achieve a non-linearity to the model, a non-linear function of the layer is required. This function is also called *activation function* and differs from model to model. Examples of common activation functions are:

1. **Sigmoid**: Function that converts the weighted sum to a value between 0 and 1:
$$F(x) = \frac{1}{1 + e^{-x}}$$
2. **Rectified linear unit** or (ReLU): usually works better than a smooth function like the sigmoid, and also easier to compute:
$$F(x) = max(0, x)$$

In fact, any mathematical function can serve as an activation function.

3.4 Machine Learning Frameworks

In these recent years, Machine and Deep Learning have become an important part of users' web experience but also a difficult task for developers creating new models. In order to simplify this process, several Machine Learning Frameworks have been created allowing developers to build ML models without getting into the complex underlying algorithms.
In this section, a few of these frameworks will be listed with an increasing attention to those involved in this thesis.

3.4.1 TensorFlow

TensorFlow is an open-source software library for dataflow programming across a range of tasks. It is a symbolic math library for numerical computation that uses data flow graphs and is also used for machine learning applications, such as neural networks. It was developed by the Google Brain team [38] for internal use and released under the Apache 2.0 open-source licence on November 2015 [39]. Currently it is one of the most utilised frameworks, due to its completeness and reliable libraries.

Figure 3.10: TensorFlow logo.

The architecture is flexible so to allow users to deploy computation to one or more CPUs (or GPUs) in a desktop, server, or even mobile device with a single API. In a not exhaustive list of features, one could mention [40]:

- TF runs on Windows, Linux, and macOS, and also on mobile devices, including both iOS and Android;

- TF provides a very simple Python API, which is called TF.Learn (*tensorflow.contrib.learn*), compatible with Scikit-Learn;
- TF provides another simple API called TF-slim (*tensorflow.contrib.slim*) to simplify building, training, and evaluating NNs;
- TF's main Python API offers much more flexibility (at the cost of higher complexity) to create all sorts of computations, including any NN architecture you can think of;
- TF includes highly efficient C++ implementations of many ML operations, particularly those needed to build NNs. There is also a C++ API to define your own high-performance operations;
- TF provides several advanced optimization nodes to search for the parameters that minimize a cost function: TF automatically takes care of computing the gradients of the functions one defines, i.e. implements automatic differentiating (or autodiff);
- TF also comes with a great visualization tool called TensorBoard that allows you to browse through the computation graph, view learning curves, and more;
- Once a model is done with TF, computations can be deployed to one or more CPUs or GPUs, as needed.

An example code usage for the creation of a TensorFlow Gradient Descent model (see Section 3.2.1) is listed above:

Listing 3.1: TensorFlow example code.

```
import tensorflow as tf
import numpy as np

n_epochs = 10000
learning_rate = 0.01
X = tf.constant(<MY_FEATURES>, dtype=tf.float32, name="X")
y = tf.constant(<MY_TARGET>, dtype=tf.float32, name="y")

theta = tf.Variable(tf.random_uniform([n+1,1],
-1.0,1.0, seed=42), name="theta")

y_pred = tf.matmul(X, theta, name="predictions")
error = y_pred - y
mse = tf.reduce_mean(tf.square(error), name="mse")
gradients = 2/m * tf.matmul(tf.transpose(X), error)

training_op = tf.assign(
theta, theta - learning_rate * gradients)

init = tf.global_variables_initializer()

with tf.Session() as sess:
    sess.run(init)
```

```
    for epoch in range(n_epochs):
        if epoch % 100 == 0:
            print("Epoch", epoch, "MSE =", mse.eval())
        sess.run(training_op)

best_theta = theta.eval()
```

3.4.2 Scikit-learn

Scikit-learn was initially developed by David Cournapeau as a Google summer of code project in 2007. It provides a range of supervised and unsupervised algorithms via a consistent interface in Python and is distributed under many Linux distributions, for academic and commercial use [41].

Figure 3.11: Scikit-learn logo.

With few lines of code, scikit-learn provides lots of algorithms for classification and regression problem e.g. SVM, Linear Regressor, BDT, Neural Networks, etc... The following example shows how to load a dummy dataset containing Iris flowers characteristics:

Listing 3.2: Scikit-learn example code.

```
# Sample Decision Tree Classifier
from sklearn import datasets
from sklearn import metrics
from sklearn.tree import DecisionTreeClassifier
# load the iris datasets
dataset = datasets.load_iris()
# fit a CART model to the data
model = DecisionTreeClassifier()
model.fit(dataset.data, dataset.target)
print(model)
# make predictions
expected = dataset.target
predicted = model.predict(dataset.data)
# summarize the fit of the model
print(metrics.classification_report(expected, predicted))
print(metrics.confusion_matrix(expected, predicted))
```

In the example shown above, the model is created in Line 8 and trained in the next line. At last, predictions are made in Line 13 and the scorers of such predictions in the last two lines.

3.4.3 Keras

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano [42]. It was developed with a

focus on enabling fast experimentation. It is a framework with a simple interface, written in python, minimizing the number of user actions required for common use cases and providing clear and actionable feedback upon user error. Keras is one

Figure 3.12: Keras logo.

of the most favourite frameworks among developers, as shown in the number of mentions in scientific papers uploaded to arXiv.org (Figure 3.13). The core data structure of Keras is a model, a way to organize layers. The simplest type of model is the *Sequential* model, a linear stack of layers.
Here is an example of sequential model:

Listing 3.3: Create a sequential model.

```
from keras.models import Sequential

model = Sequential()
```

Stacking NN layers is easily achieved by:

Listing 3.4: Stack layers in Neural Network.

```
from keras.layers import Dense

model.add(Dense(units=64, activation="relu", input_dim=100))
model.add(Dense(units=10, activation="softmax"))
```

In Line 3 a hidden layer is created: *units* corresponds to the number of nodes (or neuron) for that specific layers, *activation* is the activation function (explained in Section 3.3) required. The first hidden layer requires also the dimension of input layer which is given in the *input_dim* parameter.
Once the model looks good, the configuration of its learning process is done by:

Listing 3.5: Configure learning.

```
model.compile(loss='mean_squared_error', optimizer='sgd')
```

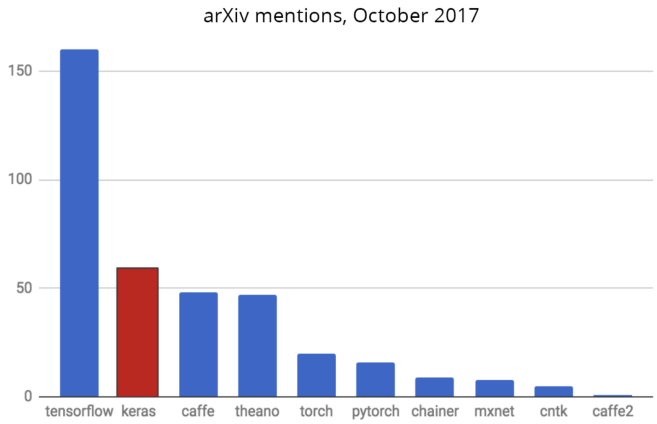


Figure 3.13: Keras popularity in arXiv mentions.

This method takes as input the *loss* function (mean squared for this example but many more can be used) and the configuration of the *optimizer* (in this case the Stochastic Gradient Descent, introduced in Section 3.2).
Training the model in batches is than performed:

Listing 3.6: Training iteration in batches.

```
model.fit(x_train, y_train, epochs=5, batch_size=32)
```

At last, the predictions on new data with a simple line:

Listing 3.7: Predictions on new data.

```
classes = model.predict(x_test, batch_size=128)
```

This basic example shows the simplicity in the construction of a primitive Neural Network. Using a more complex structure, the analysis presented in this thesis is mainly done in Keras (using TensorFlow engine as backend).

Chapter 4

Muon momentum assignment with Machine Learning

In Chapter 3, an overview on Machine Learning techniques was presented and various aspects of the application of learning algorithms have been discussed. In particular, it has been underlined that many algorithms are adequate in very different domains, scientific or not.
The main purpose of this thesis concerns the application of Machine Learning algorithm using data coming from local trigger segments (trigger primitives) from the Drift Tubes chambers (explained in more detail in Chapter 2) at the CMS experiment for the *muon p_T assignment predictions*.
After a brief description of the physical problem analysed, this chapter will be articulated into three main sections:

- The first section will provide an overview of the data analysed and the pre-processing required in order to make such data ready for a Machine Learning model;
- The second section will show the creation of the models, training and test, as well as the predictions for new unseen data;
- The third section will compare the predictions coming from such models with the actual Level-1 trigger algorithm running in CMS.

All the code written for this thesis, the output data and plots are available in this GitHub repository [43].

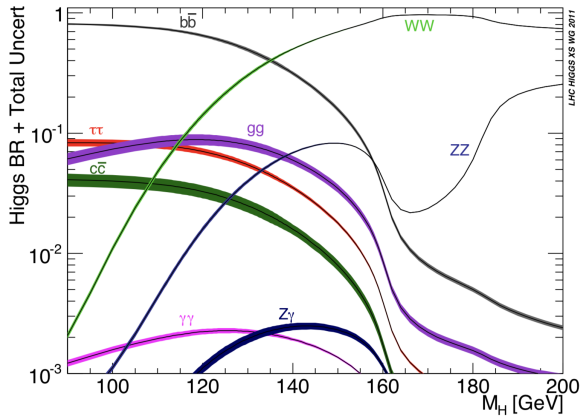
4.1 The muon p_T assignment

Detection of muons is very important in the CMS experiment. Muons are expected to be produced in a large set of physics signatures studied as part of the CMS physics programme; for instance, a *golden* decay channel for the study of the Higgs Boson (Figure 4.1a), is the one into four muons [44], as shown in Figure 4.1b. Since muons can penetrate several metres of iron with little energy loss, unlike most particles they are not stopped by any layer of CMS calorimetry systems. CMS muon detectors (described in more detail on Chapter 2) are therefore the outermost

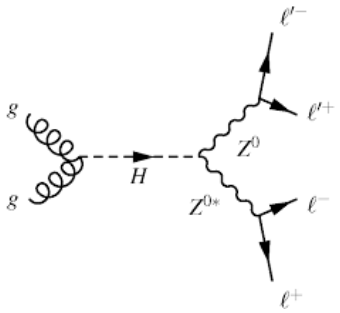
ones, far from the interaction point and shielded from most prompt background sources.
A muon is tracked through the four muon stations of the barrel system in two steps. First, individual hits with single chambers are combined by local trigger electronics (see Section 2.3), resulting in track segments (trigger primitives) encoding local position and direction. Then, a Track Finder system combines segments from different stations into muon track candidates. The present Barrel Muon Track Finder (see Section 2.4) estimates the muon transverse momentum in a relatively simple way, using Look Up Tables addressed by the difference in the azimuthal position of the trigger primitives from the innermost stations participating in the track.
Improving the transverse momentum measurement performed by the trigger, namely the momentum resolution, is very important to achieve a reduction of trigger rates. In fact, due to the exponential shape of the inclusive prompt muon p_T spectrum, even a relatively small reduction of the resolution can provide a significant decrease of the trigger rate at a given p_T threshold, by reducing the number of low momentum muon candidates misidentified as high momentum. This becomes particularly pressing in the context of future upgrades of CMS, in view of High-Luminosity LHC (see Section 1.4), to avoid using higher momentum thresholds as luminosity increases, with the consequence of loosing physics acceptance.
Besides, the current algorithm for Level-1 muon p_T assignment is based on the use of a fraction of all the available information from the trigger primitives: as a matter of fact, firstly it assumes that the muon comes from the vertex, and secondly it uses just the ϕ_B of the most internal station (as long as it is not MB3) and the difference of the ϕ angles among the two most internal stations. A ML approach, especially in the implementation of an artificial neural network, allows to ingest all the angle differences, all ϕ_b, the position and direction within a chamber of all trigger primitives (up to 4 of them), the station/sectior/wheel coordinates of the chambers generating the primitives (more information compared to the Level-1 trigger), and tries to explore though the use of more hidden layers in the network architecture many possibile non-linear relations among the input data to perform a supervised regression problem, thus spanning a much wider parameter space.
In this work, we use ML predictions to provide a momentum measurement based on the trigger primitives (at chamber level) associated to muon track candidates built by the present Track Finder system, which will also provide the reference performance in terms of momentum assignment capability.

4.2 Data Preparation

The first important step prior to model creation is the data preparation. In a standard Machine Learning problem, this is a delicate phase that deals with aspects such as data cleansing, data scaling (normalization and binning),feature inspection and eventually feature engineering.
In this particular case, the input data format is also an issue. Every input data is given in a ROOT [45] TTree data format. This particular format is so widely used by the HEP community to be considered as a "standard" in relation to



(a) Higgs branching ratios and their uncertainties for the full mass range.



(b) Higgs decay into 4 leptons ($l = e, \mu, \tau$).

Figure 4.1: The image on the left shows the branching ratios (defined as the ratio between a certain decay channel over the sum of all channels). The image on the right shows the $ZZ \to llll(\mu^+\mu^-\mu^+\mu^-)$ golden channel.

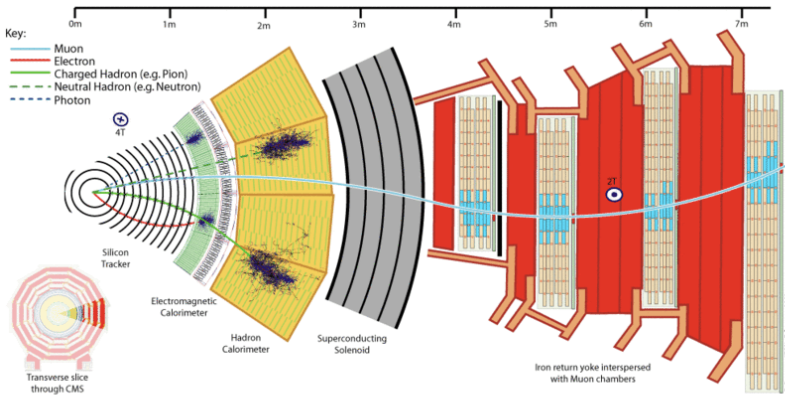


Figure 4.2: Path of a muon in the CMS detector (blue curve). Notice how the other particles (muons, photons, hadrons...) are stopped way before the magnet. Crossing the muon chambers, they produce a signal that allow the reconstruction of the particle path.

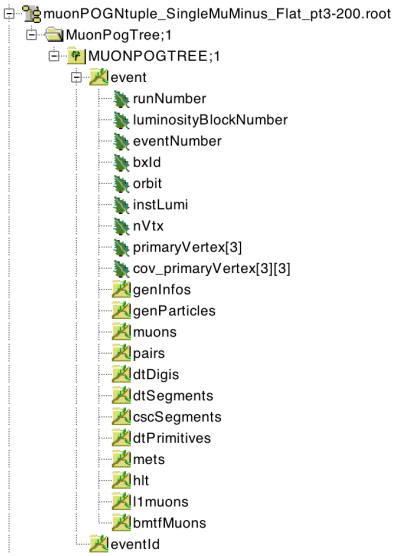


Figure 4.3: The structure of the ROOT Tuple used for analysis.

physics data analysis. ROOT files are also known for their ability to store almost everything: from data Tuples (TTrees) to histograms and N-dimensional plots. However, it has to be highlighted that this format may not be suitable for a direct ML learning application, if the choice is to be able to use and compare world-class algorithms in standard ML frameworks (see Section A) and by the overall Data Science community, not only the packages available inside the ROOT framework. For this reason a transition from the ROOT input file to any other format, more suited for any existing ML framework (see Section A for more information), is required.
The first option evaluated for this task was a toolkit named *uproot* [46], a reader of the ROOT file format using only Python and Numpy (a package for scientific computing in Python) [47] for a performant and scalable solution, able to deal with a large set of ROOT file structures. This solution, however, was soon excluded for a specific reason: the TTrees used for this study are produced by the CMS Muon POG (Physics Object Group) using a class-customized structure, as shown in Figure 4.3. The *uproot* package, instead, has the capability to deal only with ROOT TTree with with a flat structure.
In order to deal with such a structure, the best option available was to create a series of custom-made python macros able to correctly extract such information from the Muon POG TTree structure and store it in a plain CSV (C*omma* S*eparated* V*alues*) file i.e. a format in which tabular data (numbers and text) are stored as a sequence of records each of which consisting of one or more fields, separated by a comma.

4.2.1 From ROOT files to flat CSV

The python macro written for this specific scenario, however, was not part of any pre-made existing package (like *uproot*) and was developed specifically for this study.

This macro was written using a python extension module called *pyROOT*: it allows interactions with any ROOT class (familiar to the HEP community, written mostly in C++) directly from the Python interpreter. Furthermore, using a parsing I/O interface, a single configuration line is required for the entire conversion process. In the code listed above, an example of its usage its shown:

Listing 4.1: Conversion macro usage.

```
./root_to_csv.py --help
Usage: PROG [-h] [--fin INPUTFILE] [--branch TTREE]
[--branches BRANCHES] [--fout FOUT] [--branch-list] [--cut CUT]
```

In this line, some arguments need to be set for an optimal usage:

- -h, –help: shows the help message (with the same instructions written in this bullet list);
- –fin INPUTFILE: the input ROOT file path name that requires conversion;
- –branch BRANCH: the ROOT specific TTree to be read (in Figure 4.3, the branch that has to be converted is 'event');
- –branches BRANCHES: list of comma-separated variables, stored in the in the ROOT tree selected with the above parameter, that will be extracted and converted. If nothing is inserted, all variables will be included;
- –fout FOUT: the output CSV file containing the variables listed above;
- –branch-list: this optional boolean argument may be useful in case there is no familiarity with the ROOT file. If enabled, a list of the ROOT file structure will be provided;
- –cut CUT: list of possible cuts that may be requested for the ROOT file e.g. a given threshold in a listed variable.

Muon primitives into CSV file

Concerning the actual data used for analysis, the input ROOT TTree is generated out of a simulated sample of muons with different energy and direction crossing the CMS detector. Each generated muon, passing through the barrel muon DT chambers, results into several track segments, called *trigger primitives*. The number of segments associated to each muon depends on the number of chambers crossed by its trajectory. Therefore, inside the *TTree* structure (see Figure 4.3) all the necessary information are stored, starting from the physical attributes of each simulated muon and the respective primitive hits in the detector, up to the Level-1 trigger information.
An example of the specific I/O settings used for this analysis is listed below:

Listing 4.2: Muon ROOT file conversion parameters.

```
./root_to_csv.py
--fin .../muonPOGNtuple_SingleMuMinus_Flat_pt3-200.root
--branch MuonPogTree/MUONPOGTREE
```

```
4 --branches "dtPrimitive.id_r,dtPrimitive.id_eta,
  dtPrimitive.id_phi,dtPrimitive.phiGlb(),dtPrimitive.phiB,
6 dtPrimitive.quality,l1Muon.eta,l1Muon.phi,l1Muon.charge,
  genParticle.status,l1Muon.quality,l1Muon.pt,genParticle.pdgId,
8 genParticle.eta,genParticle.phi,genParticle.pt"
  --fout output_muons.csv
10 --cut "dtPrimitive.bx==0"
```

where the input filename and the list of branches are shown.
The variables extracted from the ROOT file and their physical meanings are the follows:

1. *dtPrimitives* contains the variables of muon local segments from the DT chambers (Section 2.1.1):
   - dtPrimitive.id_r → an integer with the identifier of the trigger primitive station (ranging from 1 to 4);
   - dtPrimitive.id_phi → an integer with the identifier of the trigger primitive sector (ranging from 1 to 12);
   - dtPrimitive.id_eta → an integer with the identifier of the trigger primitive barrel wheel (ranging from -2 to 2);
   - dtPrimitive.phiGlb() → the ϕ angle of the particle expressed in global coordinates (depending on the active station);
   - dtPrimitive.phiB → the corresponding bending angle (see Figure 2.14 for reference);
   - dtPrimitive.quality → an integer with a value representing the quality of the hit (a map of such values for the quality of the primitive is shown in Table 2.1).
2. *l1Muons* contains the variables of muons exiting the Level-1 Trigger (specifically the BMTF, explained in Section 2.4):
   - l1Muon.eta and l1Muon.phi → the value of the η and ϕ angles of the muon track estimated by the Level-1 trigger;
   - l1Muon.charge → an integer representing the charge of the muon (-1 or +1);
   - l1Muon.quality → an integer with a value of quality given by the Level-1 trigger depending on the type of Track Finder involved (in this case Barrel TF). Ranges from 0 to 15, an higher value implies, in general, the use of more trigger primitives and/or the use of primitives from the innermost stations.
   - **l1Muon.pt** → the estimation of the transverse momentum p_T made by the Level-1 Trigger system. This variable is quite important as it is used to compare the performance of the p_T assignment by the model;
3. *genParticles* contains the variables of the generated muons. These are the "true" physical values since the original interacting muon is generated and not coming from real data:

```
Event,dtPrimitive.id_r,dtPrimitive.id_eta,dtPrimitive.id_phi,dtPrimitive.phiGlb(),dtPrimitive.phiB,dtPrimitive.quality,l1Muon.eta,l1Muon.phi,l1Muon.charge,
genParticle.status,l1Muon.quality,l1Muon.pt,genParticle.pdgId,genParticle.eta,genParticle.phi,genParticle.pt
1,1,1,7,3.3085849285125732,-9.0,2,0.8047500252723694,-2.9780611991882324,-1,1,12,111.0,13,0.779859185218811,-3.016859292984009,55.022857666015625
1,2,2,7,3.3127353191375732,9.0,6,0.8047500252723694,-2.9780611991882324,-1,1,12,111.0,13,0.779859185218811,-3.016859292984009,55.022857666015625
1,3,2,7,3.3149325847625732,2.0,6,0.8047500252723694,-2.9780611991882324,-1,1,12,111.0,13,0.779859185218811,-3.016859292984009,55.022857666015625
1,4,2,7,3.3156650066375732,4.0,6,0.8047500252723694,-2.9780611991882324,-1,1,12,111.0,13,0.779859185218811,-3.016859292984009,55.022857666015625
2,2,1,12,5.557437896728516,2.0,6,0.380625009536743l6,-0.731013298034668,-1,1,12,139.5,13,0.3656277060508728,-0.7421414852142334,163.72418212890625
2,3,1,12,5.558170318603516,2.0,5,0.380625009536743l6,-0.731013298034668,-1,1,12,139.5,13,0.3656277060508728,-0.7421414852142334,163.72418212890625
2,4,1,12,5.558658599853516,-24.0,2,0.380625009536743l6,-0.731013298034668,-1,1,12,139.5,13,0.3656277060508728,-0.7421414852142334,163.72418212890625
3,1,1,2,0.4276515245437622,4.0,6,0.6851249933242798,0.42541199922561646,-1,1,12,139.5,13,0.6750427484512329,0.41554176807403564,186.16143798828125
3,2,1,2,0.4283839464187622,-28.0,2,0.6851249933242798,0.42541199922561646,-1,1,12,139.5,13,0.6750427484512329,0.41554176807403564,186.16143798828125
```

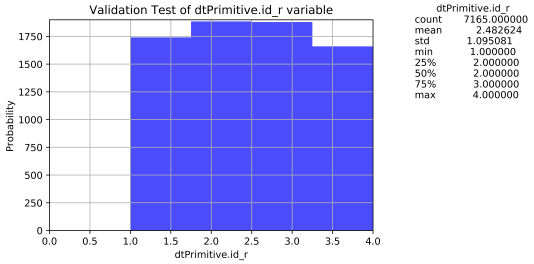
Figure 4.4: Output produced by the conversion macro. Each field is separated by a comma and each record by a breakline.

- genParticle.status → an integer flag assuming different values depending on the generation process. Particles with status flag 1 are the ones propagated from the generator to the GEANT [48] simulation and are the ones used later in this study;
- genParticle.pdgId → Pdg code of the generated particle (-13 for muons and +13 for anti-muons);
- genParticle.eta and genParticle.phi → 'real' value of the η and ϕ angles of the muon;
- **genParticle.pt** → transverse momentum of the generated muon. This variable will be used as the target from the Machine Learning algorithm i.e. the quantity that the model has to learn.

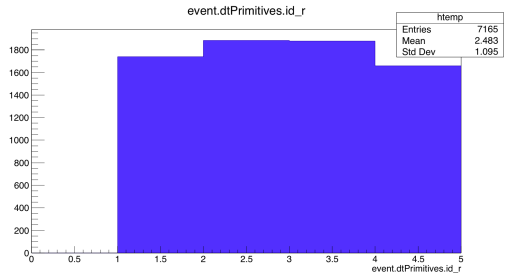
Figure 4.4 shows the output produced after the execution of the conversion macro, stored in a plain CSV file.

4.2.2 Data validation

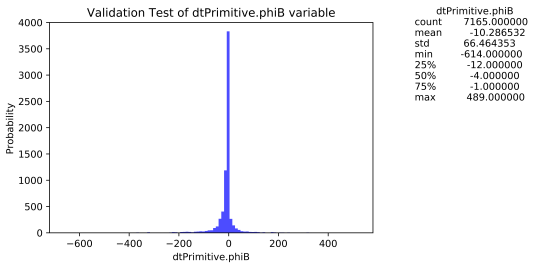
This particular process of data extraction and conversion, however, may be dangerous if done wrong. For instance, there might be a lot of missing attribute values for some data entries or an error during conversion (bug in the code or in the original file). In order to prevent this kind of issues, a data cross-check validation is strongly suggested before entering the phase of model building and training.
The purpose of this step is not the control over the correctness of the physics information, which we assume to be correct a priori, as from the CMS Barrel data quality checks, but the consistence of the new file with respect to the original ROOT format.
This step was performed with a python-based pipeline designed on a Jupyter Notebook [49] for an efficient and interactive plot making and content checking. The number of branches and the number of items inside each have been compared to the original information from ROOT. Each column from the CSV file was then plotted on the Jupyter notebook (using Matplotlib module [50]) and checked with the original plot traditionally done in a ROOT *TCanvas*.
Some examples are shown in Figure 4.5: Figure 4.5a shows the *dtPrimitive.id_r* variable, that contains integers, to be compared with Figure 4.5b from the original ROOT file; Figure 4.5c shows the distribution of the ϕ angle, a floating point variable, compared with Figure 4.5d.



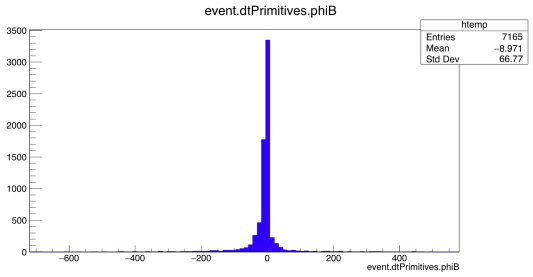
(a) Variable *dtPrimitive.id_r* plotted from CSV file with Matplotlib.



(b) Variable *dtPrimitive.id_r* plotted from the original ROOT file.



(c) Variable *dtPrimitive.phi* plotted from CSV file with Matplotlib.



(d) Variable *dtPrimitive.phi* plotted from the original ROOT file.

Figure 4.5: Data validation step. The left plots show the histograms of two variables after the conversion from the new CSV file; the right plots, instead, show the original variables plotted using ROOT TBrowser.

4.2.3 Data pre-processing

This step concludes the data preparation phase and has the purpose of making the CSV file, just created, ready for a Machine Learning model building and training. The entire pre-processing was handled and controlled by another macro, written in Python using a Jupyter Notebook that made possible a step-by-step execution.
In Figure 4.6 an extract from the CSV input file is shown. Each record of this file contains the physical information of one primitive: a single muon, however, does not produce only one primitive but a finite range (not exceeding four primitives i.e. the number of stations in the muon chambers). The column *Event*, instead, is an integer number associated with the actual interacting muon. For instance, in Figure 4.6, the heading four records of the CSV file are labelled as *Event=1*: this means that the corresponding muon has produced four primitives. The next event is then displayed and so on, until the last muon of the simulation is reached.
This particular configuration of data, is quite straightforward but is not optimal in view of a Machine Learning training process: in fact, looking at the *genParticle.pt* column (containing the transverse momentum of the generated muon), it is possible to notice that, for each event, the record is repeated. This is not surprising considering that is the same interacting muon. Since the measure of the muon momentum is actually performed combining information from all the primitives produced along the particle's trajectory, data must be re-arranged so that the link between a muon and the full set of primitives, generated in the different chambers,

	Event	dtPrimitive.id_r	dtPrimitive.id_eta	dtPrimitive.id_phi	dtPrimitive.phiGlb()	dtPrimitive.phiB	dtPrimitive.quality	l1Muon.eta	l1Muon.phi	...	genParticle.pt
0	1	1	1	7	3.308585	-9.0	2	0.804750	-2.978061	...	55.022858
1	1	2	2	7	3.312735	9.0	6	0.804750	-2.978061		55.022858
2	1	3	2	7	3.314933	2.0	6	0.804750	-2.978061		55.022858
3	1	4	2	7	3.315665	4.0	6	0.804750	-2.978061		55.022858
4	2	2	1	12	5.557438	2.0	6	0.380625	-0.731013	...	163.724182
5	2	3	1	12	5.558170	2.0	5	0.380625	-0.731013		163.724182
6	2	4	1	12	5.558659	-24.0	2	0.380625	-0.731013		163.724182
7	3	1	1	2	0.427652	4.0	6	0.685125	0.425412		186.161438
8	3	2	1	2	0.428384	-28.0	2	0.685125	0.425412	...	186.161438
9	3	3	2	2	0.429361	0.0	6	0.685125	0.425412		186.161438
10	3	4	2	2	0.429361	0.0	5	0.685125	0.425412		186.161438
11	4	4	-2	8	3.973297	16.0	2	-0.761250	-2.312673		176.452087
12	4	2	-2	9	3.972482	6.0	5	-0.761250	-2.312673	...	176.452087
13	4	1	-1	8	3.971100	-7.0	2	-0.761250	-2.312673		176.452087

Figure 4.6: Input CSV file example before the pre-process (extracted from the actual data).

is made explicit in the format in input to ML. The pre-processing procedure fix this issue, changing the main configuration of the input file. Instead of having a primitive for each row, the new file has an event (i.e. a muon) for each row and the primitives were all shifted as individual columns. The same extract with the new configuration is shown in Figure 4.7. It is possible to notice that each row has now its specific and unique transverse momentum, with the addition of a *n_Primitive* column with an integer value corresponding to the number of primitives contained in that record. Then, each column of the first data file has been multiplied n-times, proportionally to the maximum number of primitives possible. When an event has a number of primitives lower than the maximum, a place-holder is inserted (0).

Other important additions to the dataset structure are then the *delta_phi* columns. These columns are filled with the difference, in module, of the ϕ angles: more specifically the *dtPrimitive.phiGlb()* columns containing the ϕ angle expressed in global detector coordinates. Physically, the p_T of a particle, crossing the detector, is inversely proportional to the bending angle caused by the effects of the strong magnetic filed produced by the solenoidal magnet. Therefore, as a feature engineering problem, for each combination of station pairs (MB2-MB1, MB3-MB1, etc...), a new variable is created. In the eventuality of no primitive in a certain section, no difference can be computed and a place-holder (0) is inserted.
After this changes, the final dataset is store into another CSV file which is ready for the creation of the Machine Learning models.

4.3 Creation of the model

In the previous section, it was discussed the preparation of data for Machine Learning i.e. converting a ROOT file into a flat CSV (see Section 4.2.1) and how to validate the data (see Section 4.2.2), concluding with the pre-processing phase with feature optimizations (see Section 4.2.3). After these steps, a Machine Learning model for our specific task can start to be built.

	Event	n_Primitive	1dtPrimitive.id_r	2dtPrimitive.id_r	3dtPrimitive.id_r	4dtPrimitive.id_r	1dtPrimitive.id_eta	2dtPrimitive.id_eta	3dtPrimitive.id_eta	4dtPrimitive.id_eta	...
0	1.0	4.0	1.0	2.0	3.0	4.0	1.0	2.0	2.0	2.0	
1	2.0	3.0	2.0	3.0	4.0	0.0	1.0	1.0	1.0	0.0	...
2	3.0	4.0	1.0	2.0	3.0	4.0	1.0	1.0	2.0	2.0	
3	6.0	4.0	1.0	2.0	3.0	4.0	0.0	0.0	0.0	0.0	...
4	7.0	3.0	1.0	2.0	3.0	0.0	0.0	0.0	0.0	0.0	
5	12.0	3.0	1.0	3.0	4.0	0.0	0.0	1.0	1.0	0.0	...
6	17.0	2.0	2.0	3.0	0.0	0.0	-1.0	-1.0	0.0	0.0	

...	genParticle.pdgId	genParticle.eta	genParticle.phi	delta_phi12	delta_phi13	delta_phi14	delta_phi23	delta_phi24	delta_phi34	genParticle.pt
	13.0	0.779859	-3.016859	0.004150	0.006348	0.007080	0.002197	0.002930	0.000732	55.022858
...	13.0	0.365628	-0.742141	0.000732	0.001221	0.000000	0.000488	0.000000	0.000000	163.724182
	13.0	0.675043	0.415542	0.000732	0.001709	0.001709	0.000977	0.000977	0.000000	186.161438
...	13.0	0.108610	-2.497962	0.001709	0.003174	0.003906	0.001465	0.002197	0.000732	117.951797
	13.0	-0.079830	-1.993825	0.004395	0.007813	0.000000	0.003418	0.000000	0.000000	45.763668
...	13.0	0.243388	1.946428	0.003662	0.004150	0.000000	0.000488	0.000000	0.000000	112.617790
	13.0	-0.584353	-1.273355	0.002197	0.000000	0.000000	0.000000	0.000000	0.000000	49.043083

Figure 4.7: Input CSV file example after the pre-process (extracted from the actual data).

4.3.1 Algorithm selection

The selection of a Machine Learning algorithm that fits this particular problem is the first step. It is not possible to know a priori the best algorithm: therefore, the only way is to try a number of different algorithms and focus with those resulting to be the most promising. In this step, spot-checking is a way of discovering which algorithms perform well on a specific ML problem.
This step is quite basic and requires an easy framework with an high number of pre-loaded algorithms to compare. Scikit-learn framework (explored in more detail in Section A or [41]) provides an easy solution for this problem, comparing several algorithms at the time with a low CPU usage and an user-friendly programming interface.
In the following, several algorithms will be listed by category. Starting from the *linear* algorithms:

- **Linear Regression**: linear regression is a basic but powerful Machine Learning algorithm. Widely used in regression problems, the function used to fit the problem is:

  $$Y = \beta_0 + \beta_1 X$$

  where X is the vector of features and β_0, β_1 are the coefficients we wish to learn.

- **Ridge Regression**: is an extension for linear regression. It is a regularized model, which is a common mechanism for avoiding overfit by adding another element to the loss function that should be minimized as well. This term sums over squared β values and multiplies it by another parameter λ, punishing the loss function for high values of the β coefficients. Therefore, during training, such coefficients are enforced to be lower as possible.

- **Lasso**: another extension built on regularized linear regression, but with a small difference. The regularization term is in absolute value. This overcomes the disadvantage of Ridge regression by not only punishing high values of the β coefficients but actually setting them to zero if they are not relevant. Using

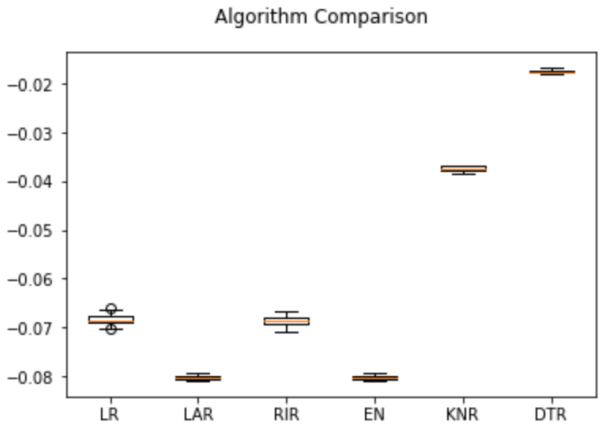


Figure 4.8: Comparison of several algorithms using Scikit-learn framework. The x-axis shows the different algorithms taken into account while the y-axis shows the RMSE scorer (negative). An higher number means a better model.

this algorithm, it is possible to end up with fewer features included in the model than the initial one.

- **Elastic Net**: it linearly combines the two methods above in a two-stage procedure: first it finds the ridge regression coefficients and then does a Lasso type shrinkage.

In addition to the algorithms above, also few *non-linear* ML algorithms have been considered:

- **k-Nearest Neighbors** (kNN): It uses a distance metric to find the k most similar instances in the training data for a new instance and takes the mean outcome of the neighbors as the prediction.

- **Decision Tree Regressor**: along with the Classification Tree, together named CART, or just decision trees, they construct a binary tree from the training data. Split points are chosen greedily by evaluating each attribute and each value of each attribute in the training data in order to minimize a cost function.

The algorithms mentioned above have been tested as potential good models for the physical problem under exam, using default values for all their internal parameters. They have been tried on the training sample, and the results have been compared one with the other in order to estimate their "scores". The scorer adopted for the evaluation of such model is the RMSE, describe in more detail in Section 3.2. The results obtained with the two best algorithms after the spot-checking step, as shown in Figure 4.8, show that KNR and DTR offer the best results in terms of RMSE without specific optimizations. Before proceeding further, e.g. via hyper-parameter tuning on these algorithms, in the meantime an alternative approach based on ANN was attempted, giving comparable results if not better prior to optimization. Given the larger flexibility of Neural Network learning algorithm, it was decided to continue pursuing this latter path, as explained in the following sections.

4.3.2 Artificial Neural Network with Keras and TensorFlow

Scikit-learn was the first attempt to find an optimal algorithm for the creation of models. However, results started to become interesting with the usage of Artificial Neural Network algorithms (described in detail in Section 3.3).
The neural networks, compared to other ML models, are much more flexible: instead of following a set of predefined instruction to solve a problem, they process information by a large number of interconnected processing elements working in parallel; the resulting model may also be easily loaded in other machines and tested with new datasets.
The following sections will provide a detailed description of the process needed to create such networks, using a python macro with the Keras framework environment (described in detail in Section A).

Input Dataset and feature selection

For this specific analysis, the entire dataset used contains about 300000 generated muons with a range in p_T from 3 to 200 GeV/c, in an uniform distribution. In Section 4.2.1, a description of the CSV file used for analysis was provided along with a list of the variables extracted from the ROOT file. Feature selection is the first step in the creation of the neural network model: from the above list, only some of the variable were effectively used as input feature. The other variables left will be used in the last part of this thesis, which is the comparison of the predictions with the Level-1 information.
The list of the features used is the following: *n_Primitive*, *dtPrimitive.id_r*, *dtPrimitive.id._eta*, *dtPrimitive.id_phi*, *dtPrimitive.phiB*, *dtPrimitive.quality*, *delta_phi* (one for each combination of stations). *dtPrimitive.id_r*, *dtPrimitive.id._eta*, *dtPrimitive.id_phi* were adopted for the location of the primitive in the detector; *delta_phi* and *dtPrimitive.phiB* for their correlation to the muon p_T; *dtPrimitive.quality* for the correlation between the number of hits used for the construction of the primitive, which has an impact on the resolution of ϕ_B (see Section 2.3).
As already stated in the section above, each *dtPrimitive* variable refers to one primitive only; each muon, however, has more than one primitive (*n_Primitive* indicates the exact number of primitives for a certain muon record). For this reason, every variable has an additional integer number at the beginning of their name indicating the primitive number. For instance, *dtPrimitive.id_r* actually means: *1dtPrimitive.id_r*, *2dtPrimitive.id_r*, *3dtPrimitive.id_r* and so on, for the other variables as well.

Target variable

The target variable is *genParticle.pt*, which contains the transverse momentum of the simulated muon, distributed from 3 to 200 GeV/c and normalized between 0 and 1 since, in a Neural Network environment (but it is extended to all Machine Learning algorithms), the target variable is supposed to be normalized in order to be easily learned.

4.3.3 Neural Network structure

The architecture of the Neural Network was implemented using Keras as follows:

Listing 4.3: Neural Network architecture.

```
model = Sequential()
model.add(Dense(1000, input_dim=27,
      kernel_initializer='random_normal', activation='sigmoid'))
model.add(Dropout(rate=0.1))
model.add(Dense(50, activation='sigmoid'))
model.add(Dropout(rate=0.1))
model.add(Dense(1, activation='sigmoid'))
```

The first hidden layer is described in Line 2: it has 1000 neurons and receives the information directly from the input layer of 27 features; the sigmoid function selected as activation function (see Section 3.3) and a gaussian random number as kernel initializer (for the initial randomized weight assignment). Then a dropout is inserted: it consists in a random selection of a fraction of input units to 0 at each update during training time, preventing overfitting.
The second layer, in Line 5, is identical to the first one but contains only 50 neurons. In the end, the output layer (with only one node) closes the Network.

4.3.4 Training the model

The training of the Neural Network was implemented with Keras as follows:

Listing 4.4: Train the Neural Network.

```
model.compile(loss='mean_squared_error', optimizer=optimizer)
    model.fit(x_train, y_train,
          batch_size=size,
          epochs=epochs,
          verbose=1,
          validation_data=(X_validation, y_validation))
```

The loss function used is the *Root Mean Squared Error* (described in Section 3.2). For the optimizer, several options were analysed [51]:

- **AdaGrad**: AdaGrad (Ad*aptive* Grad*ient* Algorithm) is an extension of the Stochastic Gradient Descent (Section 3.2.1) that, instead of maintaining a single learning rate for all weight updates during training, it adapts the learning rate to the parameters, performing smaller update. Frequently occurring features will have small parameters updates while infrequent features will have larger updates;

- **RMSProp**: RMSProp (Root Mean Square Propagation), another SGD extension similar to AdaGrad, have been originally developed in order to resolve Adagrad's diminishing learning rate issue. This optimizer, in fact, divides the learning rate by an exponentially decaying average of squared gradients;

- **Adam**: Adam (Ad*aptive* *M*oment Estimation) is another method which realizes the benefit of both AdaGrad and RMSProp. Instead of adapting

#	Optimizer	Training RMSE	Validation RMSE
1	AdaGrad	0.170369	0.169735
2	RMSProp	0.109598	0.110485
3	Adam	0.105400	0.107358
4	AdaMax	0.108883	0.110199

Table 4.1: Scores for the Neural Network training and validation phases.

the parameter learning rates based on the average first moment (the mean), Adam also makes use of the average of the second moments of the gradients (the uncentered variance);

- **AdaMax**: Similar to Adam, AdaMax is based on the infinity norm for stable parameters adjustments [52].

In order to validate the model, a train/test split of 80/20% was performed. Using a batch size of 300 records and 260 epochs (total number of iterations) for each configuration, Table 4.1 shows the results for both the training and validation phases. Figures from 4.9a to 4.9d show the loss value trend over the iterations (epochs) during training and validation for each optimizer adopted (using the RMSE function): for each iteration, the minimization of loss is noticeable from the constantly decreasing curve.

4.3.5 Testing the model

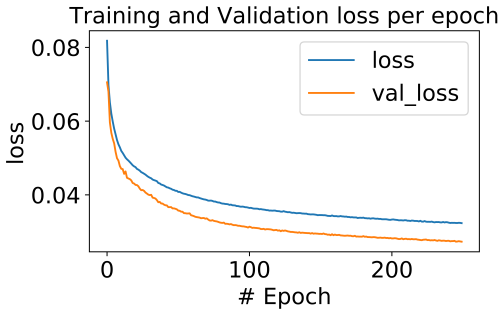
The final step for the model creation is the test of such models in a separate dataset. The Keras models trained in Section 4.3.4 were initially saved into a HDF5 file [53], containing:

- The architecture of the model, in order to be recreated with other datasets;
- The weights of the model;
- The training configuration (loss, optimizer);
- The state of the optimizer, allowing to resume training exactly where it was left.

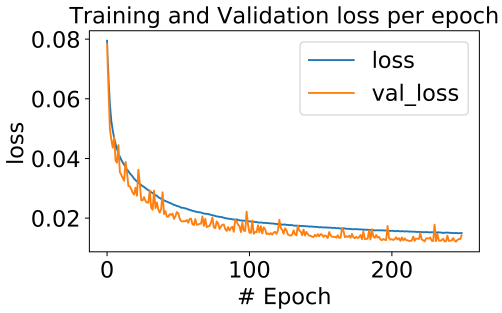
Then, using another CSV file with other simulated muons (about 15k muons), created and processed in the same way as the training sample (see Section 4.2), it was possible to load the models and predict the muon p_T by adding the following code:

Listing 4.5: Test the NN models.

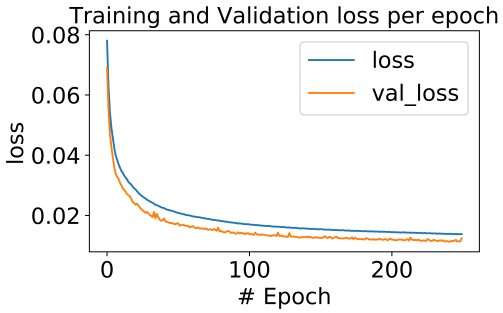
```
model = load_model('<model_name>.h5')
predictions = model.predict(X_test)
```



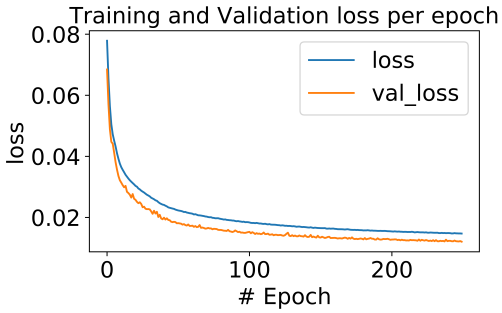
(a) Adagrad optimizer loss values.



(b) RMSProp optimizer loss values.



(c) Adam optimizer loss values.



(d) AdaMax optimizer loss values.

Figure 4.9: These plots show the trend of the loss function for each iteration (epoch). The blue line indicates the training phase while the orange line the validation phase. Notice that, for each epoch, the loss function is always minimized.

#	Optimizer	Test RMSE
1	AdaGrad	0.170537
2	RMSProp	0.111046
3	Adam	0.107109
4	AdaMax	0.110991

Table 4.2: Scores for the Neural Network test phase.

In Figure 4.10, scatter plots of the predicted value of p_T versus the "real" values of the p_T (coming from the generated muon variables) are shown. Each plot corresponds to a different loaded model used for the predictions. Table 4.2 shows the loss value for each test model, using the RMSE function. From this plots, it is important to notice the progressive decrease in quality for high values of momentum. This effect is probably physical: at high p_T, the bending angle of particles becomes very small, becoming comparable to the angular resolution of the trigger primitives.

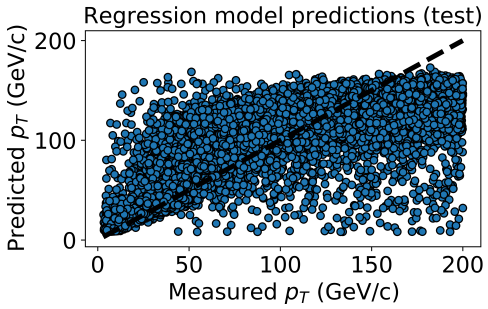
4.4 Results: comparison with Level-1 Trigger

Once the models have been trained and tested, the best one was chosen for the comparison between the p_T predicted by Machine Learning and the p_T estimated by the p_T assignment unit of the Barrel Muon Track Finder (Section 2.4). Tables 4.1 and 4.2 indicate that the best score was provided by the *Adam* optimizer model (see Section 4.3.4).
The analysis code performs this steps:

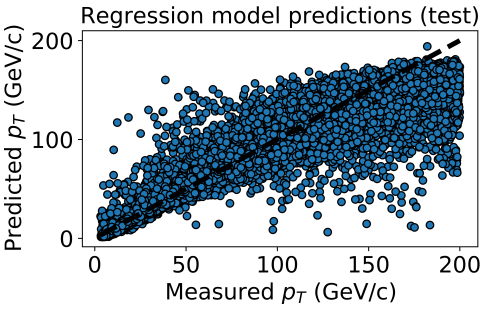
- Load the model, in this case *Adam* optimizer, which performed the best scores;
- For each generated muon, a $dR = \sqrt{d\phi^2 + d\eta^2}$ matching with the Level-1 Trigger tracks performed: in this way only muons with corresponding Level-1 track are considered, factoring the Level-1 reconstruction efficiency out of the study;
- If dR is below a certain threshold, the track is selected and histograms start to be filled.

The input dataset for the analysis contained about 14000 muon entries. There are two different types of plots produced as output:

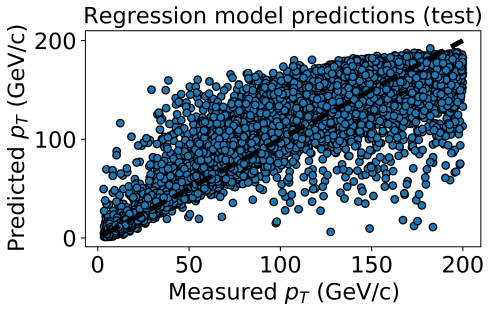
1. p_T resolution histograms;
2. Efficiency curves computed as function of the generated muon p_T ("Turn-on curves").



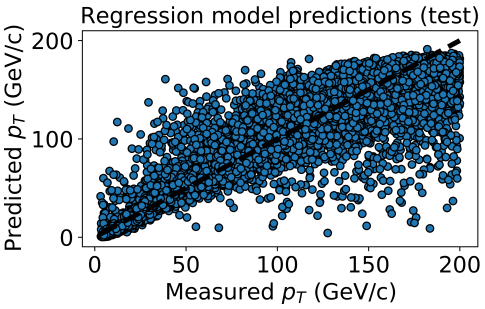
(a) Adagrad optimizer.



(b) RMSProp optimizer.



(c) Adam optimizer.



(d) AdaMax optimizer.

Figure 4.10: Scatter plot of the model p_T predictions versus "real" p_T from the simulation. The dashed line corresponds to an ideal correspondence of the predicted values. At high p_T values, predictions are less accurate than low p_T ones.

4.4.1 p_T resolution histograms

The p_T resolution histograms, for each generated muon, are filled using the following relation:

$$\frac{\Delta p_T}{p_T} = \frac{p_{T_{est}} - p_{T_{gen}}}{p_{T_{gen}}}$$

where $p_{T_{est}}$ is the estimation of the transverse momentum, given by the model prediction or the Level-1 trigger p_T *assignment unit*. Figure 4.11 shows the p_T resolution for the entire range analysed (from 3 to 200 GeV/c). The red line indicates the resolution of the Level-1 trigger system while the blue line indicates the resolution of the predictions made by the Neural Network model. In particular, it is possible to notice a less broad distribution for the Machine Learning resolution, resulting in a overall improvement (yet small) with respect to the Level-1 trigger system. Another noticeable detail, from the Level-1 trigger curve, is the small peak over the value of -1: this happens when the p_T assigned by the Level-1 Trigger is significantly underestimated with respect to the generated muon p_T. The Machine Learning momentum assignment is less prone to large p_T under estimation. As will be discussed in the next section, this difference has implications in the efficiency to trigger on muons when a minimal p_T cut is applied on the Level-1 trigger track.
Figures from 4.12 to 4.16 show the same p_T resolution histograms but in specific momentum ranges. Even though the overall resolution shows an improvement, Figure 4.12 highlights a significantly broader resolution plot in the $3 - 20$ GeV/c range of the Learning model which is instead rather in agreement in the other regions of p_T. In general, an additional feature is visible in all the p_T resolution plots is the scale: the peak of the ML-based distributions close to zero if compared with the Level-1 trigger one. This is due to a different calibration of the p_T scale in the two cases. The Level-1 Trigger p_T assignment is in fact defined as the point for which the efficiency turn on curve reaches the 90% value. The ML instead attempts to estimate the generated muon p_T and is therefore characterized by a smaller scale bias.

4.4.2 Efficiency turn-ons

The efficiency turn-ons, as the word states, represent an efficiency ϵ defined as follows:

$$\epsilon = \frac{\text{number of muons that passes a defined threshold in} p_T}{\text{total number of muons}}$$

After the definition of a minimal p_T threshold cut, the efficiency numerator is filled if the p_T value from the ML or the Level-1 p_T assignment is above threshold. Then the ratio with the denominator, filled with muons resulting in a Level-1 trigger track, is performed. Results are plotted as function of the generated p_T.
In Figure 4.17 a turn-on efficiency curve is shown. A threshold of 22 GeV/c was selected, as it is the threshold of lowest p_T-cut unprescaled single muon triggers. Such a momentum cut, in fact, has nearly full acceptance for signatures of events at the electroweak scale ($Z \to \mu\mu$, $W \to \mu\nu$) and above.
Figures 4.18, 4.19 show the same curves with different cuts, respectively 27 and

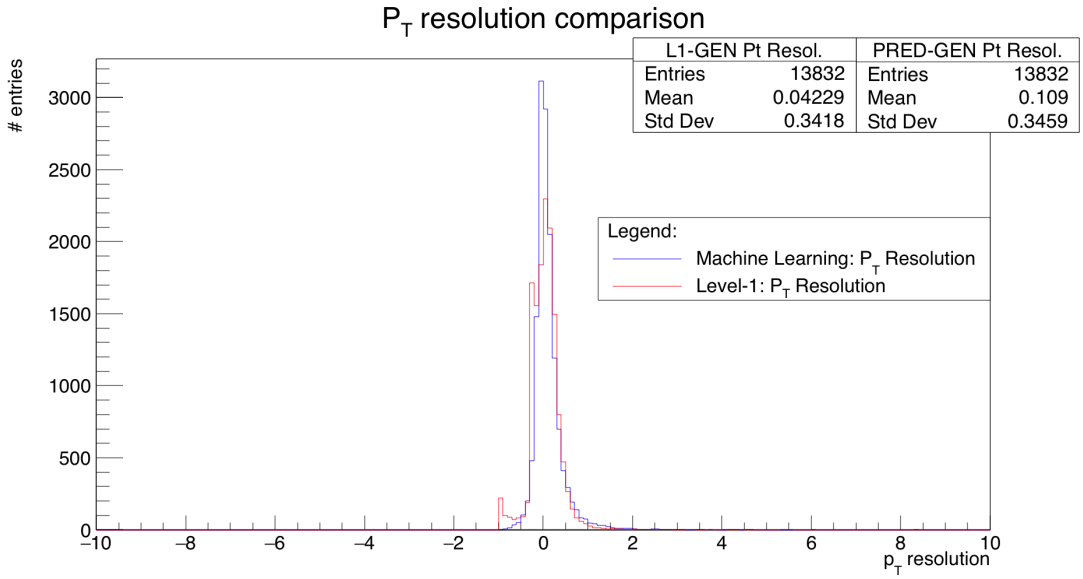


Figure 4.11: Transverse momentum resolution histograms computed for the ML-based (blue) and L1-trigger based momentum assignment, computed for muons generated in the $3 - 200$ GeV p_T range.

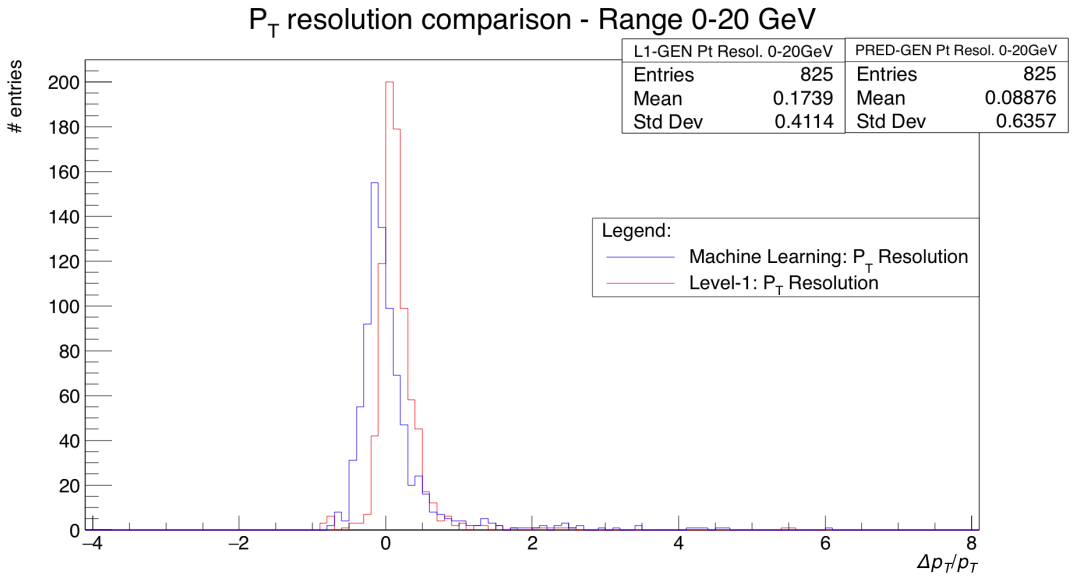


Figure 4.12: Transverse momentum resolution histograms computed for the ML-based (blue) and L1-trigger based momentum assignment, computed for muons generated in the $3 - 20$ GeV p_T range.

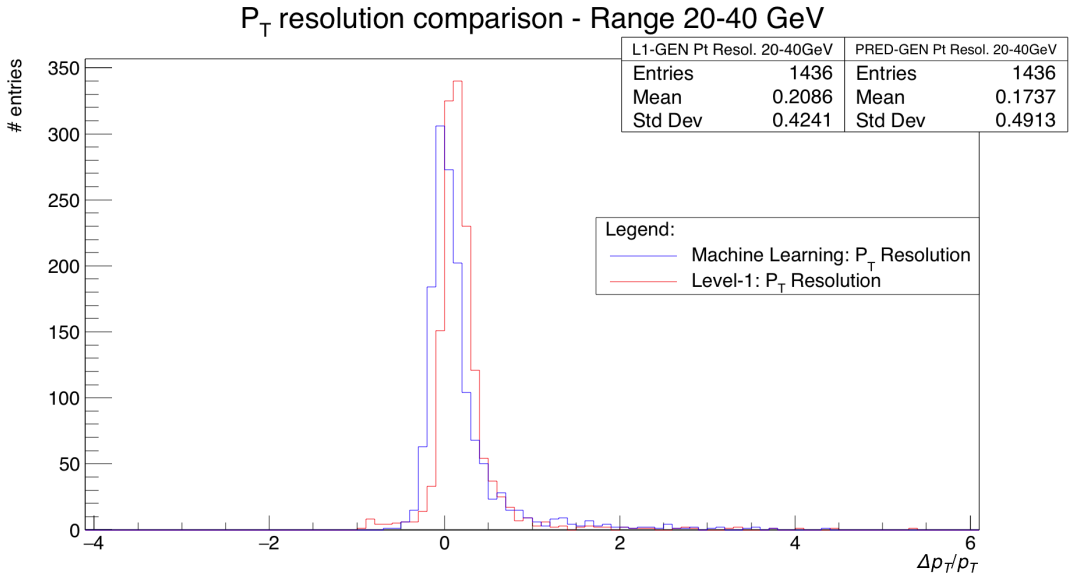


Figure 4.13: Transverse momentum resolution histograms computed for the ML-based (blue) and L1-trigger based momentum assignment, computed for muons generated in the $20 - 40$ GeV p_T range.

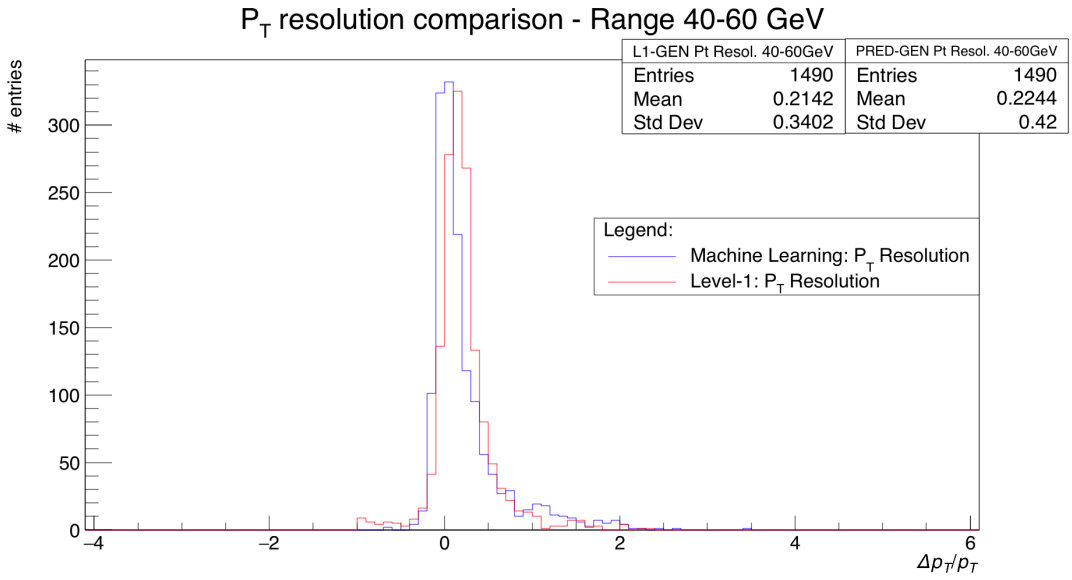


Figure 4.14: Transverse momentum resolution histograms computed for the ML-based (blue) and L1-trigger based momentum assignment, computed for muons generated in the $40 - 60$ GeV p_T range.

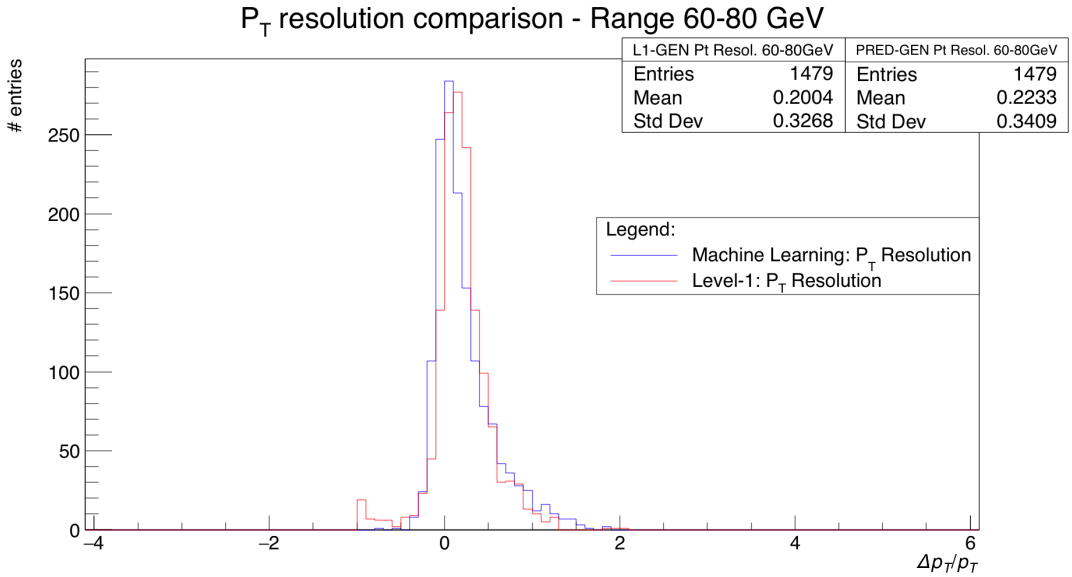


Figure 4.15: Transverse momentum resolution histograms computed for the ML-based (blue) and L1-trigger based momentum assignment, computed for muons generated in the $60 - 80$ GeV p_T range.

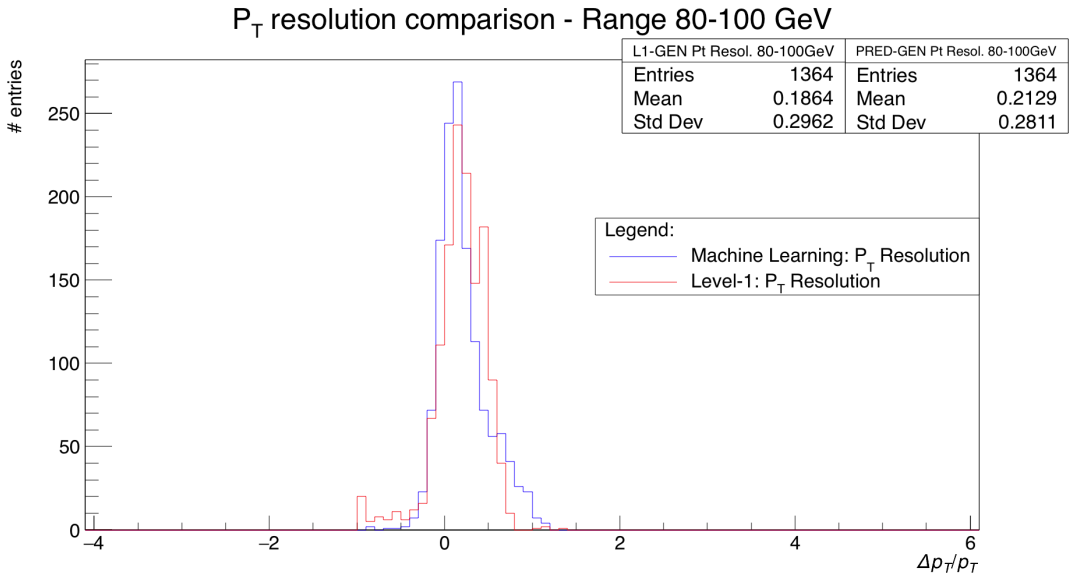


Figure 4.16: Transverse momentum resolution histograms computed for the ML-based (blue) and L1-trigger based momentum assignment, computed for muons generated in the $80 - 100$ GeV p_T range.

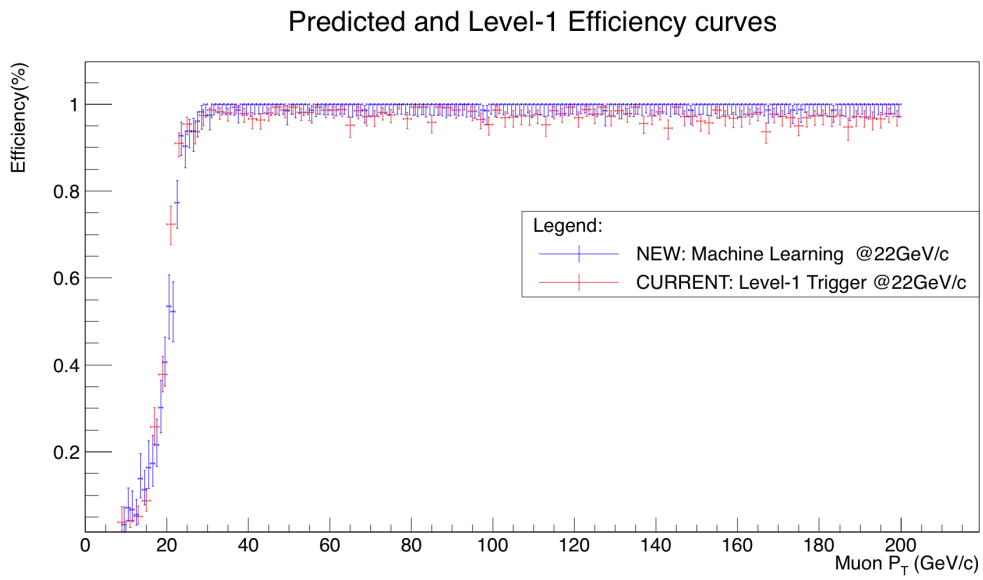


Figure 4.17: Efficiency turn-on given a p_T threshold cut of 22 GeV . The blue dots show the efficiency for the Machine Learning based momentum assignment, while the red dots show the efficiency for the Level-1 trigger p_T assignment.

32 GeV/c. As for the p_T resolution plots, turn-ons have a different calibration for Machine Learning and Level-1 trigger: however, this difference has been accounted for and the resulting plots are calibrated. From these plots, a very high efficiency in the plateau region of the Machine Learning turn-on plot can be observed: the lower efficiency of the Level-1 curve in the high-pt region is mainly caused by the underestimation of the muon Pt in some events, which is visible as the small peak at -1 in the pt resolution plots. The Machine Learning model, instead, is not affected by this issue and the efficiency at high values of p_T is nearly 100%. On the other hand, Figures 4.17 and 4.18 show a small increase of efficiency in the $10-20$ GeV/c region which is comparable with the worse resolution in the low p_T range (Figure 4.12). This is not optimal since it is located below the threshold thus resulting in a higher fraction of muons with low momentum misidentified as high momentum, and consequently in a potential increase of overall rate.

4.5 Next steps

Some aspects of this work may be analysed in more detail with a further study. Machine Learning models can be tuned and perfected over time: other algorithms may be explored and tested with further attention. Optimization of such models, including *ensemble* methods, has not been yet performed, but it is a future goal to undertake.
The p_T assignment produced by the Machine Learning model was comparable with the Level-1 Trigger p_T *assignment unit* with some future improvements: in Section 4.4.2, an increased efficiency below the threshold very likely causes an higher rate than the current Level-1 Trigger system. In the High-Luminosity LHC phase of operations, this rate should be as low as possible since the instantaneous luminosity

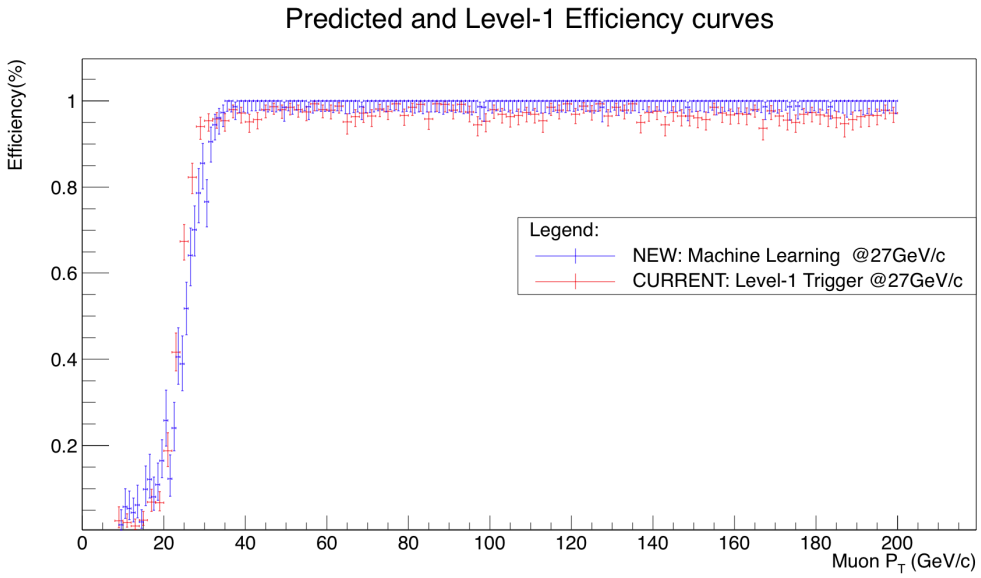


Figure 4.18: Efficiency turn-on given a p_T threshold cut of 27 GeV . The blue dots show the efficiency for the Machine Learning based momentum assignment, while the red dots show the efficiency for the Level-1 trigger p_T assignment.

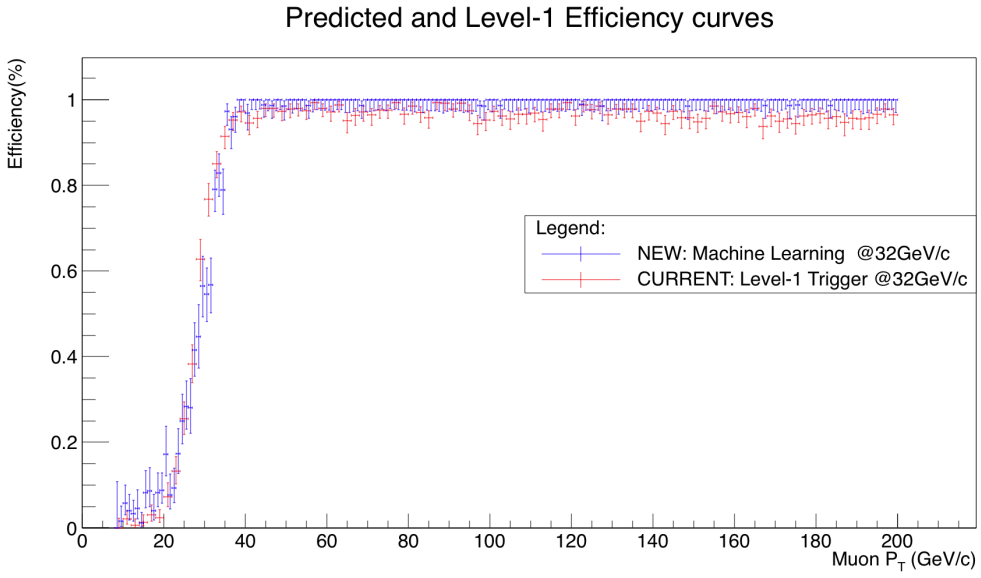


Figure 4.19: Efficiency turn-on given a p_T threshold cut of 32 GeV . The blue dots show the efficiency for the Machine Learning based momentum assignment, while the red dots show the efficiency for the Level-1 trigger p_T assignment.

will increase compared to the present run. For this reason, future studies in trigger rates should be performed. Moreover, it would be interesting to test the model model in other regions of phase space, for example studying the efficiency to select muons with transverse momentum of the order of several hundreds of GeV/c.

Conclusions

This thesis aims at creating a Machine Learning model for the assignment of transverse momentum of Level-1 Trigger muons crossing the Barrel region of the CMS experiment. A precise evaluation of such variable is crucial: a better p_T assignment, in fact, allows the trigger rate to be reduced, a very important improvement in view of High-Luminosity LHC.

After evaluating various linear and non-linear learning algorithms, a model based on an Artificial Neural Network approach was selected as the one offering the highest scores even without major investments in hyper-parameters tuning, thus still having additional margins of improvement. After preliminary investigations with the Scikit-learn python library, the main Machine Learning framework used to build and train the models was Keras, using TensorFlow as backend.
The training dataset for this specific analysis contained about 300000 generated muons with a range in p_T from 3 to 200 GeV/c, following a uniform distribution. The NN architecture built for this analysis consisted of 3 layers: an input layer with 27 features and 1000 neural units, an hidden layer of 50 neural units and the output unit, all with sigmoid as activation function. Among all the different choices in the model optimization, the best score was reached by the *Adam* optimizer with a RMSE of 0.105 on the training set, yielding 0.107 on the validation set.

Exploiting this model, we can implement regression on the target p_T observable which is presented in this thesis as p_T resolution histograms and "turn-on" curves i.e. efficiency curves computed as function of the generated muon p_T, discussed separately in the following.
Machine Learning p_T prediction shows a comparable overall resolution with respect to the Level-1 trigger system, slightly better at higher p_T values, while a worse assignment is observed in the $3 - 20$ GeV/c range. The steepness of the turn-on curves indicates, in general, a good agreement between the traditional Level-1 trigger system and the ML-based new solution. In the plateau region of the ML turn-on curve an efficiency very close to 100% is observed, significantly better than in the present Level-1. Thus, investigating the behaviour of the ML p_T assignment for very high momentum muons (order of TeV and above), crucial to several new physics searches, could be an interesting continuation of this work. On the other hand, an increase of efficiency in the low p_T ($10 - 20$ GeV/c) region is observed: this can probably cause a higher rate of low momentum muons misidentified as high momentum, although this was not directly investigated in this thesis, and should be addressed in further developments.

On possible further improvement on the model creation and optimization side, *ensemble* methods should be evaluated as potentially promising alternatives towards higher scores, as well as more advanced hyper-parameters tuning e.g. via grid search approaches. Concerning the performance in the training phase, which might result decisive to allow a faster turn around in implementing and evaluating different configurations of the model, a natural next step would be to explore Keras, using TensorFlow_GPU as backend and setup the code base used in this thesis on a GPU-equipped computing resource.

This work was presented at the Sixth Annual International Conference on Large Hadron Collider Physics (LHCP) in Bologna, June 2018.

Appendix A

Machine Learning Glossary

In the previous Sections, a brief explanation of some Machine Learning concepts and techniques was made. In this Appendix, a short list of commonly-used terms is offered [54], in order to facilitate the reading of this thesis:

- **Activation Function**: Is a mathematical function e.g. ReLU or sigmoid, that takes in the weighted sum of all the inputs and generates an output value (non-linear) to the next layer;
- **Classification**: A type of supervised machine learning model for distinguishing among two or more discrete classes. For example between 'Yes' or 'No', 'Male' or 'Female';
- **Clustering**: Grouping related examples, particularly used on Unsupervised Learning. Once all the examples are grouped, a human can optionally supply meaning to each cluster. An example, the *k-means* algorithm clusters examples based on their proximity to a centroid;
- **Examples**: An example is an instance of data e.g. a single row in a dataset. It is *labelled* if contains both features and the label, while it is *unlabelled* if contains the features but not the label. In a Machine Learning typical workflow, the labelled set of examples is used for training and the unlabelled subset for the validation/test subset;
- **Feature**: A feature is an input variable, x in a simple regression problem. A simple Machine Learning model has one feature, while more sophisticated algorithms could use dozens or hundreds of features:

$$\{x_1, x_2, \cdots x_n\}$$

- **Gradient Descent**: A technique to minimize loss by computing the gradients of loss with respect to the model's parameters, conditioned on training data. Informally, gradient descent iteratively adjusts parameters, gradually finding the best combination of weights and bias to minimize loss;
- **Hidden Layer**: In a neural network, a hidden layer is a collection of neurons located between the input layer (features) and the output layer (predictions);

- **Label**: the label is the thing we are predicting - the y variable in a simple regression problem. It can be a housing prize value or, in this specific work, a physical quantity of a fundamental particle;
- **Learning Rate**: A scalar used to train a model via gradient descent. During each iteration, the gradient descent algorithm multiplies the learning rate by the gradient. The resulting product is called the gradient step;
- **Loss**: A measure of how far a model's predictions are from its label. To determine this value, a model must define a loss function. For example, linear regression models typically use mean squared error for a loss function, while logistic regression models use Log Loss;
- **Model**: A model defines the relationships between features and label. There are two main phases that a model has to perform: *training* the model (see **Training**) and *inference* i.e. use the trained model to unlabelled examples;
- **Neural Network**: A model that, inspired by the structure of the human brain, is composed of layers (at least one of which is hidden) consisting of simple connected units or neurons;
- **Neuron**: Also called *node*, is the fundamental unit of a neural network. It calculates the output value by applying an activation function (non-linear) to a weighted sum of input values;
- **Overfitting**: Creating a model that matches the training data so closely that the model fails to make correct predictions on new data;
- **Regression**: A type of model that outputs continuous (typically, floating-point) values. To compare with **Classification** models;
- **Supervised Learning**: Training a model from input data and its corresponding labels e.g. a student that learns a subject by studying a set of questions and their answers, then answer to new and unseen questions. Compare with **Unsupervised learning**;
- **Test set**: The subset of the data set that you use to test your model after the model has gone through initial vetting by the validation set. Also see **Training set** and **Validation Set**;
- **Training set**: The subset of the data set used to train a model;
- **Unsupervised Learning**: Training a model to find patterns in a data set, typically an unlabelled data set. Usually, unsupervised learning is performed by clustering data into groups of similar examples. Compare with **Supervised learning**;
- **Validation set**: A subset of a dataset, disjunct from the training set, used for adjusting the parameters of the model. Also see **Training set** and **Test set**.

Bibliography

[1] Oliver Sim Brüning et al. *LHC Design Report*. Ed. by CERN library copies. Vol. 1, 2, 3. 2012. URL: http://ab-div.web.cern.ch/ab-div/Publications/LHC-DesignReport.html (cit. on pp. 1, 3).

[2] Lyndon Evans and Philip Bryant. "LHC Machine". In: *Journal of Instrumentation* 3.08 (2008). Ed. by IOPscience, S08001. URL: http://iopscience.iop.org/1748-0221/3/08/S08001 (cit. on pp. 1, 3).

[3] *CERN*. URL: http://www.cern.ch (cit. on p. 1).

[4] *CERN Engineering*. URL: http://home.web.cern.ch/about/engineering/vacuum-empty-interstellar-space (cit. on p. 2).

[5] *CERN Engineering*. URL: http://home.web.cern.ch/about/engineering/pulling-together-superconducting-electromagnets (cit. on p. 2).

[6] *CERN Engineering*. URL: http://home.web.cern.ch/about/engineering/radiofrequency-cavities (cit. on p. 3).

[7] *The ALICE experiment*. URL: http://home.web.cern.ch/about/experiments/alice (cit. on p. 3).

[8] The ALICE Collaboration et al. "The ALICE experiment at the CERN LHC". In: *Journal of Instrumentation* 3.08 (2008). Ed. by IOPscience, S08002. URL: http://iopscience.iop.org/1748-0221/3/08/S08002 (cit. on p. 3).

[9] *The Atlas Experiment*. URL: http://home.web.cern.ch/about/experiments/atlas (cit. on p. 4).

[10] The ATLAS Collaboration et al. "The ATLAS Experiment at the CERN Large Hadron Collider". In: *Journal of Instrumentation* 3.08 (2008). Ed. by IOPscience, S08003. URL: http://iopscience.iop.org/1748-0221/3/08/S08003 (cit. on p. 4).

[11] *The CMS Experiment*. URL: http://home.web.cern.ch/about/experiments/cms (cit. on p. 4).

[12] The CMS Collaboration et al. "The CMS experiment at the CERN LHC". In: *Journal of Instrumentation* 3.08 (2008), S08004. URL: http://stacks.iop.org/1748-0221/3/i=08/a=S08004 (cit. on pp. 4, 6, 7, 10).

[13] *The LHCb Experiment*. URL: http://home.web.cern.ch/about/experiments/lhcb (cit. on p. 5).

[14] The LHCb Collaboration et al. "The LHCb Detector at the LHC". In: *Journal of Instrumentation* 3.08 (2008). Ed. by IOPscience, S08005. URL: http://iopscience.iop.org/1748-0221/3/08/S08005 (cit. on p. 5).

[15] *The LHCf experiment*. URL: http://home.web.cern.ch/about/experiments/lhcf (cit. on p. 5).

[16] The LHCf Collaboration et al. "The LHCf detector at the CERN Large Hadron Collider". In: *Journal of Instrumentation* 3.S08006 (2008). Ed. by IOPscience. URL: http://iopscience.iop.org/1748-0221/3/08/S08006 (cit. on p. 5).

[17] *The TOTEM Experiment*. URL: http://home.web.cern.ch/about/experiments/totem (cit. on p. 5).

[18] The TOTEM Collaboration et al. "The TOTEM Experiment at the CERN Large Hadron Collider". In: *Journal of Instrumentation* 3.S08007 (2008). Ed. by IOPscience. URL: http://iopscience.iop.org/1748-0221/3/08/S08006 (cit. on p. 5).

[19] *The HL-LHC project*. URL: http://hilumilhc.web.cern.ch (cit. on p. 10).

[20] CERN. Geneva. LHC Experiments Committee. "The CMS muon project". In: (1997) (cit. on p. 13).

[21] M. Aguilar-Benitez et al. "Construction and test of the final CMS Barrel Drift Tube Muon Chamber prototype". In: *Nucl. Instrum. Meth.* A480 (2002), pp. 658–669. DOI: 10.1016/S0168-9002(01)01227-X (cit. on p. 14).

[22] J. Hauser. "Cathode strip chambers for the CMS endcap muon system". In: *Nucl. Instrum. Meth.* A384 (1996), pp. 207–210. DOI: 10.1016/S0168-9002(96)00905-9 (cit. on p. 16).

[23] G. Wrochna. "The RPC system for the CMS experiment at LHC". In: *Resistive plate chambers and related detectors. Proceedings, 3rd International Workshop, Pavia, Italy, October 11-12, 1995*. 1995, pp. 63–77 (cit. on p. 17).

[24] A. Tapper and Darin Acosta. "CMS Technical Design Report for the Level-1 Trigger Upgrade". In: (2013) (cit. on pp. 18, 19, 22, 24).

[25] P. Sphicas. "CMS: The TriDAS project. Technical design report, Vol. 2: Data acquisition and high-level trigger". In: (2002) (cit. on pp. 18, 20).

[26] K. Ecklund et al. "Commissioning of the upgraded CSC Endcap Muon Port Cards at CMS". In: *Journal of Instrumentation* 11.01 (2016), p. C01031. URL: http://stacks.iop.org/1748-0221/11/i=01/a=C01031 (cit. on p. 20).

[27] K. Zawistowski G. Żarnecki J. Dobosz P. Miętki. "Development and validation of the Overlap Muon Track Finder for the CMS experiment". In: vol. 10031. 2016, pp. 10031 –10031 –6. DOI: 10.1117/12.2250287. URL: https://doi.org/10.1117/12.2250287 (cit. on p. 20).

[28] J. Ero et al. "The CMS Level-1 Trigger Barrel Track Finder". In: *Journal of Instrumentation* 11.03 (2016), p. C03038. URL: http://stacks.iop.org/1748-0221/11/i=03/a=C03038 (cit. on pp. 20, 28).

[29] P. Paolucci. "The CMS muon system". In: *Astroparticle, particle and space physics, detectors and medical physics applications. Proceedings, 9th ICATPP Conference, Como, Italy, October 17-21, 2005*. 2006, pp. 605–615. DOI: 10.1142/9789812773678_0096. URL: http://weblib.cern.ch/abstract?CERN-CMS-CR-2006-006 (cit. on p. 21).

[30] Roberto Martinelli, A. J. Ponte-Sancho, and Pierluigi Zotto. "Design of the Track Correlator for the DTBX Trigger". In: (1999) (cit. on p. 23).

[31] "Performance of the CMS TwinMux Algorithm in late 2016 pp collision runs". In: (2016). URL: http://cds.cern.ch/record/2239285 (cit. on p. 27).

[32] A. Samuel. *Some Studies in Machine Learning Using the Game of Checkers*. Ed. by New York Springer (cit. on p. 29).

[33] *Machine Learning tutorial*. URL: https://class.coursera.org/ml-005/lecture (cit. on p. 29).

[34] Sinan Kaplan. "DEEP GENERATIVE MODELS FOR SYNTHETIC RETINAL IMAGE GENERATION". In: (July 2017) (cit. on p. 31).

[35] *Machine Learning Crash Course - Google Developers*. URL: https://developers.google.com/machine-learning/crash-course (cit. on pp. 33, 35, 37).

[36] *Overfitting in Machine Learning: What It Is and How to Prevent It*. URL: https://elitedatascience.com/overfitting-in-machine-learning (cit. on p. 38).

[37] S. Samarasinghe. *Neural Networks for Applied Sciences and Engineering: From Fundamentals to Complex Pattern Recognition*. CRC Press, 2016. ISBN: 9781420013061. URL: https://books.google.it/books?id=EyFeUiJibooC (cit. on p. 39).

[38] *Google Brain Team*. URL: https://research.google.com/teams/brain (cit. on p. 40).

[39] *TensorFlow Documentation*. URL: https://www.tensorflow.org (cit. on p. 40).

[40] Luca Giommi. "Prototype of machine learning ?as a service? for CMS physics in signal vs background discrimination". PhD thesis. URL: http://amslaurea.unibo.it/15803/ (cit. on p. 40).

[41] *Scikit-learn Documentation*. URL: http://scikit-learn.org (cit. on pp. 42, 54).

[42] *Keras Documentation*. URL: https://keras.io (cit. on p. 42).

[43] *Tommaso Diotalevi's GitHub project repository*. URL: https://github.com/Tommaso93/MuonPOGAnalysisTemplate.git (cit. on p. 45).

[44] A. Denner et al. "Standard Model Higgs-Boson Branching Ratios with Uncertainties". In: *Eur. Phys. J.* C71 (2011), p. 1753. DOI: 10.1140/epjc/s10052-011-1753-8. arXiv: 1107.5909 [hep-ph] (cit. on p. 45).

[45] R. Brun and F. Rademakers. "ROOT: An object oriented data analysis framework". In: *Nucl. Instrum. Meth.* A389 (1997), pp. 81–86. DOI: 10.1016/S0168-9002(97)00048-X (cit. on p. 46).

[46] *Uproot project repository*. URL: https://github.com/scikit-hep/uproot (cit. on p. 48).

[47] Oliphant Travis E. "A guide to NumPy". In: (2006) (cit. on p. 48).

[48] *GEANT4 website*. URL: https://geant4.web.cern.ch/support (cit. on p. 51).

[49] *Jupyter Notebook Documentation*. URL: http://jupyter.org/index.html (cit. on p. 51).

[50] *Matplotlib documentation*. URL: https://matplotlib.org (cit. on p. 51).

[51] *An overview of gradient descent optimization algorithms*. URL: http://ruder.io/optimizing-gradient-descent/index.html (cit. on p. 57).

[52] Jimmy Lei Ba Diederik P.Kingma. "Adam: A method for Stochastic Optimization". In: *3rd International Conference for Learning Representations, San Diego, 2015* (2014). arXiv: 1412.6980 (cit. on p. 58).

[53] *The HDF Group*. URL: https://support.hdfgroup.org/HDF5 (cit. on p. 58).

[54] *Machine Learning Crash Course - Glossary*. URL: https://developers.google.com/machine-learning/crash-course/glossary (cit. on p. 71).