UNIVERSITÀ STATALE DI MILANO

FACOLTÀ DI SCIENZE E TECNOLOGIE

CORSO DI LAUREA TRIENNALE IN FISICA

On the usage of machine-learning techniques to speed-up amplitude evaluation in high-energy physics

Tesi di Laurea di Emanuele Ricci
Relatore: Dott. Marco Zaro

Milano, 14 Giugno 2022

Summary

The Large Hadron Collider (LHC) is the worlds largest and most powerful particle accelerator, capable to reach a peak of 13.6 TeV energy per collision.
Many physicists hope that the Large Hadron Collider will help answer some of the fundamental open questions in physics, which concern the basic laws governing the interactions and forces among the elementary objects, the deep structure of space and time, and in particular the interrelation between quantum mechanics and general relativity. Probably the most important result obtained in LHC is the discovery of the Higgs boson, on 4th July 2012, which is strong evidence that the Standard Model has the correct mechanism of giving mass to elementary particles.
With the beginning of the Run III, and in view of the future High-Luminosity run, a steep increase is foreseen for the computational load needed to match the very high statistics of experimental data. Therefore, new techniques to improve the speed and efficiency of computer simulations are pursued.
In this work the goal is to use a neural network to learn the structure of a matrix element instead of the matrix element itself and to assess possible gain in running time. A Neural Network must be trained on a large set of points, since Machine Learning needs them to train and build a solid NN.
All points are generated using MadGraph5_aMC@NLO, a tool that automates the generation of matrix elements for high energy physics processes. It uses Monte Carlo algorithms to predict cross sections and draw Feynman diagrams of strong interactions and decays between particles, up to LO or NLO.
Processes considerated are $gg \to t\bar{t}$, $gg \to hh$ and $gg \to ZZ$. We find that the gain in running time strongly depends on the type of processes. The fewer Feynman diagrams contribute to a specific process, the faster its matrix element is, hence reducing the gain when a NN is employed. On the other hand, the goodness of the approximation also depends on the shape of the matrix element across the phase-space.

Contents

Chapter 1

QCD at LHC

1.1 LHC

The Large Hadron Collider (LHC) is the worlds largest and most powerful particle accelerator, capable to reach a peak of 13.6 TeV energy per collision. Inside the accelerator, two high-energy particle beams travel at close to the speed of light before they are made to collide.
Many physicists hope that the Large Hadron Collider will help answer some of the fundamental open questions in physics, which concern the basic laws governing the interactions and forces among the elementary objects, the deep structure of space and time, and in particular the interrelation between quantum mechanics and general relativity.

Probably the most important result obtained in LHC is the discovery of the Higgs

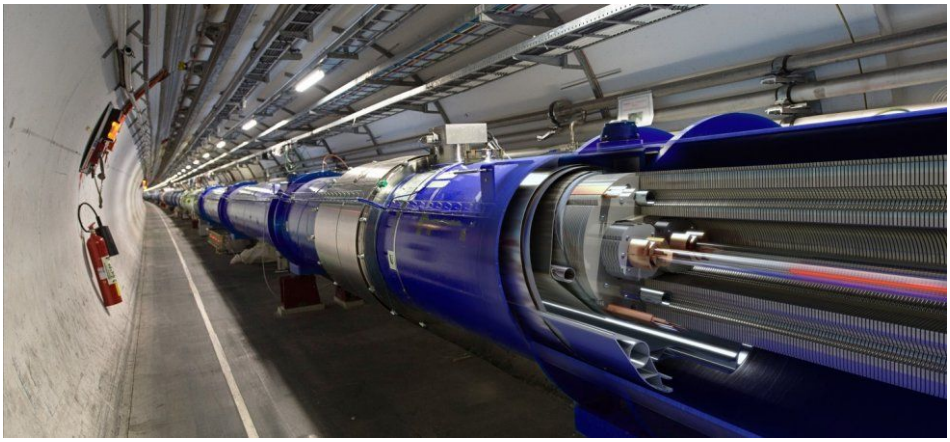

Figure 1.1:

boson, on 4th July 2012 [1] [2]. Eight detectors have been constructed at the LHC, located underground in large caverns excavated at the LHC's intersection points. Two of them, the ATLAS experiment and the Compact Muon Solenoid (CMS), are large general-purpose particle detectors.ALICE and LHCb have more specialized roles and the last four, TOTEM, MoEDAL, LHCf, and FASER are much smaller and are for very specialized research. The ATLAS and CMS experiments discovered the Higgs boson, which is strong evidence that the Standard Model has the correct mechanism of giv-

ing mass to elementary particles. All these detected data would be completely useless

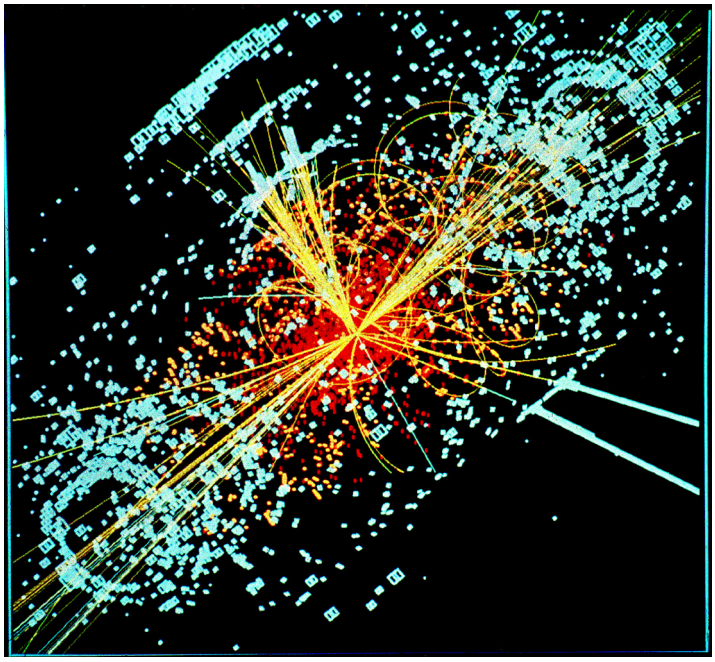

Figure 1.2: An example of simulated data modeled for the CMS particle detector on the Large Hadron Collider (LHC) at CERN. Here, following a collision of two protons, a Higgs boson is produced which decays into two jets of hadrons and two electrons. The lines represent the possible paths of particles produced by the proton-proton collision in the detector while the energy these particles deposit is shown in blue.

without a theory able to explain what's happening. This theory is the Standard Model.

1.2 Why QCD?

The Quantum Chromodynamics, QCD, is a part of the Standard Model. During an event where 2 particles interact with each other and at the end of the process there are usually more and/or different particles than we had before. We call all of this a "strong interaction" and the Quantum Chromodynamics (QCD) is the theory used to understand how it works.
Strong interactions are characterized at moderate energies by the presence of a single dimensionful scale of the order of few hundred MeV, usually indicated with Λ_S, and typical cross sections are of the order of 10 millibarns. Since in the experiments of LHC the interaction is made with protons, the QCD is exactly what we need.
Protons, indeed, are made of sub-atomic particles: gluons and quarks. One interesting thing of the QCD is that it is impossible to see these particles alone: they are seen only in triplet or doublet, this is caused by the fact that strength increases with distance. In all processes we will see later, there is never a particle alone.

1.3 The Improved Parton Model Formula

We will now turn to describe the application of perturbative QCD to processes in which hadrons are present also in the initial state, like Deep-Inelastic Scattering (DIS), or the production of some objects of high invariant mass in hadronic collisions [3]. It turns out that cross sections for these processes can be computed and related to each other. In general the cross section for the production of some final state with high invariant mass will be expressed by the so called improved parton model formula.

$$\sigma_{H_1,H_2} = \sum_{i,j} \int f_i^{(H_1)}(x_1,\mu) f_j^{(H_2)}(x_2,\mu) \hat{\sigma}_{ij}(x_1 p_1, x_2 p_2, \alpha_S(\mu), \mu)\ dx_1 dx_2 \tag{1.1}$$

An incoming beam made of hadrons of type H is equivalent to a beam of constituents (also called partons), that is to say of quark and gluons, with a longitudinal momentum distribution characterized by the parton density functions (pdfs from now on) $f_i^{(H)}(x,\mu)$. More specifically, given the hadron H with momentum p, the probability to find in H the parton i with momentum between xp and $(x+dx)p$ is precisely $dx f_i^{(H)}(x,\mu)$. The pdfs are universal, that is to say, they do not depend upon the particular process considered.
The same is not true for $\hat{\sigma}_{ij}(x_1 p_1, x_2 p_2, \alpha_S(\mu), \mu)$, the short distance cross section $\hat{\sigma}$ has to be calculated for every single process as a perturbative expansion in α_S:

$$\hat{\sigma}_{ij}(x_1 p_1, x_2 p_2, \alpha_S(\mu), \mu) = \sum_l \hat{\sigma}_{ij}^l(x_1 p_1, x_2 p_2, \mu) \alpha_S^l(\mu) \tag{1.2}$$

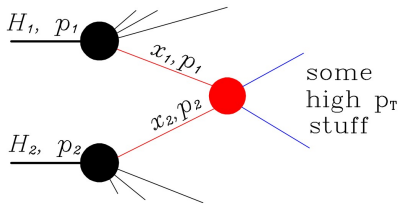

Figure 1.3: A graphic representation of the improved parton model formula

Since $\alpha_S(\mu) \approx \frac{1}{\log \frac{\mu}{\Lambda}}$, this means that by increasing the perturbative order at which the computation is performed, one adds corrections which are suppressed by one more inverse power of $\log \frac{\mu}{\Lambda}$. Those corrections are not included in this approach. Thus, for example, the pdfs describe the longitudinal momentum distribution of the partons. Since the partons are confined in a hadron, one knows that they must also have a transverse momentum of the order of the inverse of a typical hadron size, that is to say $\frac{1}{\Lambda}$. This transverse momentum is neglected, since it would give rise to power suppressed corrections.

1.4 Pertubative QCD

In the production of very massive particles, or in processes in which particles at high transverse momentum appear we can apply perturbative QCD. As seen before, particle

like a proton is composed of other smaller particles. At high energies it is impossible to consider only the proton without its component, so the tranverse momentum relative to the momentum of the proton can't be ignored.
Given the two colliding hadron beams, one defines the kinematical variables of any outgoing particles, according to the figure 1.4,

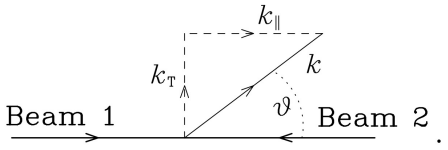

Figure 1.4: Transverse momentum at high energy. [3]

Thus, the transverse momentum k is the projection of the particle momentum into the transverse plane (the plane orthogonal to the collision axis).

1.5 What to compute

The inclusive cross section for the production of the final state X (for example a Drell-Yan lepton pair, or a Higgs boson) in the collision of hadron H_1 and H_2, is then given by the convolution

$$\sigma_{H_1H_2\to X} = \sum_{a,b} \int f_a(x_1,\mu_F) f_b(x_2,\mu_F) \hat{\sigma}_{ab\to X}(\hat{s},\mu_F,\mu_R)\ dx_1 dx_2 d\Phi_{FS} \tag{1.3}$$

where:
$f_a(x_1,\mu_F)$ and $f_b(x_2,\mu_F)$ are the parton density functions: they give the probability to find partons (quarks and gluons) in a hadron as a function of the fraction x of the proton's momentum carried by the parton.
$\hat{\sigma}_{ab\to X}(\hat{s},\mu_F,\mu_R)$ is the parton-level cross section: the cross section of the specific sub atomic particles considered
$dx_1 dx_2 d\Phi_{FS}$ is the phase-space integral: the integral over all possible states the system is represented

The parton-level cross section can be computed as a series in perturbation theory, using the coupling constant as an expansion parameter, schematically:

$$\hat{\sigma} = \sigma^{Born}\left(1 + \frac{\alpha_S}{2\pi}\sigma^{(1)} + \left(\frac{\alpha_S}{2\pi}\right)^2\sigma^{(2)} + \left(\frac{\alpha_S}{2\pi}\right)^3\sigma^{(3)} + ...\right) \tag{1.4}$$

Including higher corrections improves predictions and reduces theoretical uncertainties. Leading Order predictions can depend strongly on the renormalization and factorization scales. Including higher order corrections reduces the dependence on these scales. To compute the LO we use:

$$\sigma = \frac{1}{2s}\int |\mathcal{M}|^2 d\Phi(n) \tag{1.5}$$

1.6 Feynman Diagram example

The simplest way to calculate the cross section is using Feynman diagrams. Here an easy example is provided. Consider the process:

$$e^- e^+ \to \mu^+ \mu^- \tag{1.6}$$

where $\mathcal{M}$ is obtained calculating the diagram fig 1.5:

$$\mathcal{M} = \frac{e^2}{(p_1 + p_2)^2} [\overline{u}_3 \gamma_\mu v_4][\overline{v}_2 \gamma^\mu u_1] \tag{1.7}$$

using abbreviation $u_1 \equiv u(p_1, \sigma_1)$ If we knew momenta and polarizations of all

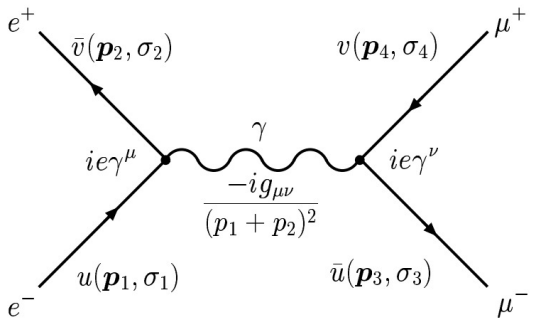

Figure 1.5: Feynmann diagram of process $e^+e^- \to \mu^+\mu^-$

external particles, we could calculate $\mathcal{M}$ explicitly. However, experiments are often done with unpolarized particles so we have to sum over the polarizations (spins) of the final particles and average over the polarizations (spins) of the initial ones:

$$|\mathcal{M}| \to |\overline{\mathcal{M}}|^2 = \frac{1}{2}\frac{1}{2} \sum_{\sigma_1\sigma_2} \sum_{\sigma_3\sigma_4} |\mathcal{M}|^2 \tag{1.8}$$

Thus,

$$|\mathcal{M}|^2 = \frac{e^4}{(p_1 + p_2)^4} \sum_{\sigma_{1,2,3,4}} [\overline{v}_3 \gamma_\mu u_4][\overline{u}_1 \gamma^\mu v_2][\overline{u}_3 \gamma_\upsilon v_4][\overline{v}_2 \gamma^\upsilon u_1] \tag{1.9}$$

The number of Feynmann diagram increases rapidly, for example we can consider another process:

$$gg \to t\bar{t}(+g) \tag{1.10}$$

we can see how quick is the growth of the total number of Feynman diagrams when adding 0,1,2 or 3 gluons in table 1.1. Matrix-element evaluation time is taken using Madraph, a tool which uses Monte Carlo algorithms for simulations of strong interactions. This topic is widely treated in the next chapter, for now let's notice that time increases quickly as well.

	0 gluon	1 gluons	2 gluons	3 gluons	4 gluons
N diagrams	3	16	123	1240	15945
time[s]	0.010	0.050	0.231	2.605	39.099

Table 1.1: Increase of times for generating processes with 0,1,2 or 3 gluons. CPU: Intel Core i7-7820HQ 2.90GHz

1.7 Computing Power and Resources

Calculating the cross section has a certain computational cost [4]. In fig 1.6 we can see that the futures needs in terms of resources will overcome what could be possibly achieved in the next few years, until 2034. Therefore, it is of utmost importance to divide new, more efficient computational techniques and algorithms that could reduce the load needed to carry all the complex stages of the LHC simulations.

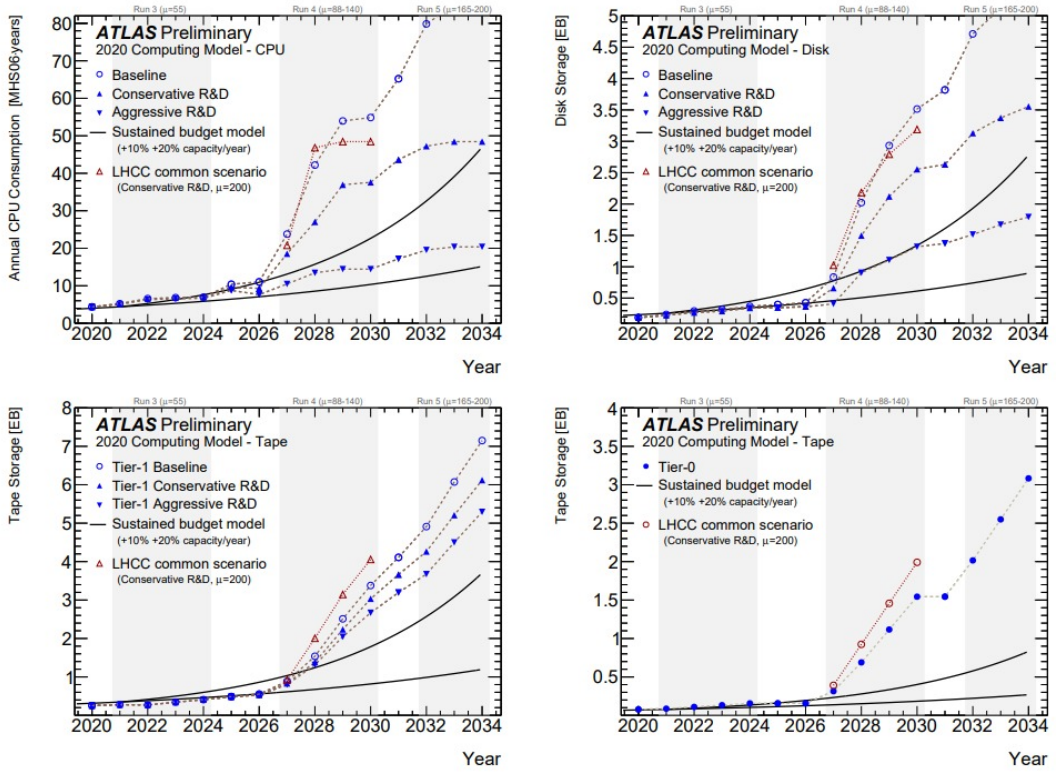

Figure 1.6: Estimated CPU, disk and tape (at the Tier-1 and Tier-0) resources needed for the years 2020 to 2034 under the different scenarios described in the text. The solid lines indicate annual improvements of 10% and 20% in the capacity of new hardware for a given cost, assuming a sustained level of annual investment. The blue dots with the brown dashed lines represent the three ATLAS scenarios following the current LHC schedule. The red open triangles indicate the Conservative R&D scenario under an assumption of the LHC reaching $\langle\mu\rangle = 200$ in 2028

In this work we will focus on the orange slice of the graph in fig 1.7. In fact Monte Carlo algorithms are implemented for the event generation, a step which, like the cross

section computation, requires several evaluations of the matrix element. The aim of this work is to try to decrease this time using Machine Learning technique.

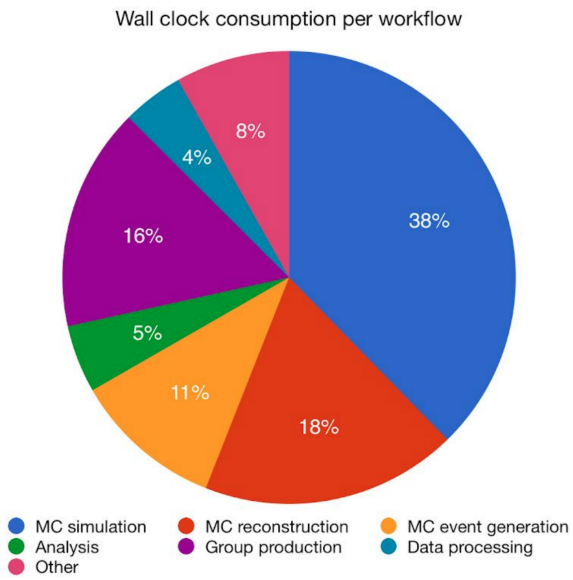

Figure 1.7: ATLAS CPU hours used by various activities in 2018

Chapter 2

Monte Carlo and Machine Learning

2.1 MadGraph5_a@NLO

MadGraph5_aMC@NLO [5] is a tool that automates the generation of matrix elements for high energy physics processes.
Mainly written in Python an Fortran, It uses Monte Carlo algorithms to predict cross sections and draw Feynman diagrams of strong interactions and decays between particles, up to LO or NLO.
After downloading it, it can be launched via command by typing in its directory:

```
./bin/mg5_aMC
```

To create a process and export the code to compute the relevant cross section in a given directory, commands are:

```
generate g g > t t~
output name_of_directory
launch
```

The process generated is the scattering of two gluons into a top-antitop. Then it is possible to modify the files in the directory, written in fortran, to change a specific functionality of the code, such as change the number of matrix-element evaluations.
Problem arises for events that include loop calculations. Due to steep increase in complexity that affects loops, when one tries to evaluate the matrix element for a large number of points the time needed become excessive. An example of this kind of process is:

$$gg \to ZZ(+g) \tag{2.1}$$

where there is the Z boson. This time we use

```
generate g g > z z [virt=QCD]
output name_of_directory
launch
```

Even in this case, let's print the number of diagrams and time, adding 0,1,2 or 3 gluons, table 2.1. The time grows a lot compared with the other process. We will try to overcome the problem using Machine Learning following the work done in [6].

	0 gluon	1 gluons	2 gluons	3 gluons
N diagrams	38	264	2366	26400
time[s]	0.674	5.745	71.117	1214.921

Table 2.1: Increase of times for generating processes with 0,1,2 or 3 gluons. CPU: Intel Core i7-7820HQ 2.90GHz

2.2 Machine Learning

Machine learning (ML) is a field of inquiry devoted to understanding and building methods that 'learn', that is, methods that leverage data to improve performance on some set of tasks. Machine learning algorithms build a model based on sample data, known as training data, in order to make predictions or decisions without being explicitly programmed to do so.
In this work the goal is to use a neural network to learn the structure of a matrix element instead of the matrix element itself and see if this procedure needs less time.
To do all of this Jupiter Notebook has been used [7]. The Jupyter Notebook is a web application for creating and sharing computational documents, that uses python. Here for machine learning work, Tensorflow was the choice [8], which is an end-to-end open source platform for machine learning.

2.3 How to proceed

ML performance and accuracy were tested versus MadGraph5_aMC@NLO's for the processes

$$gg \to t\bar{t} \tag{2.2}$$

$$gg \to hh \tag{2.3}$$

$$gg \to ZZ \tag{2.4}$$

where h is the Higgs boson and the last two processes include loops. We'll try to see if using ML is possible to achieve the same results as what we found with MadGraph5_aMC@NLO, but with a significant increase of speed and a decent accuracy.

First, NN must be trained on a large set of points, since ML needs them to train and build a solid neural network. In order to achieve this part of the code was modified for the event generation, to print 6 Million of matrix elements. As one can imagine, printing this great number of points for loop processes would have taken a huge amount of time: one computer wasn't enough. My university's cluster were used, LCM (Laboratorio di Calcolo e Multimedia), and Condor tool. Condor is a software system that creates a High Throughput Computing (HTC) environment by effectively harnessing the power of a cluster of UNIX workstations on a network [9]. Instead of running a CPU-intensive job in the background on their own workstation, users submit their job to Condor. Condor will then find an available machine on the network and begin running the job on that machine. This allowed to launch the calculation and waiting for the end without keeping PC turned on.

```
var = 0
universe = vanilla
executable = ./check
arguments = $$([$(Process)+$(var)])
output = check.out.$$([$(Process)+$(var)]).txt
error = check.err.$$([$(Process)+$(var)]).txt
queue 20
```

Here one can send the executable file *check* plus arguments from 0 to 19. Arguments are seeds for the random generation of points. The processes are numbered from 0 to 19, because queue is 20. The output of the program, and eventually also errors, are printed in files *.txt*. Any of the executable prints 100k points, so one needs 60 different executions of check, with 60 different seeds.

2.4 Cinematics

In a system with 2 incident particles coming toward each other along z-axis, e.g. 2.1, and still 2 after the process, due to relativistic momentum conservation this event is completely described using just the total initial energy $\sqrt{s}$ of the incoming particles and the angles of scattering θ relative to the z-axis. The azimuth angle ϕ is not interesting because the system is invariant for rotations around z-axis. A neural network needs to

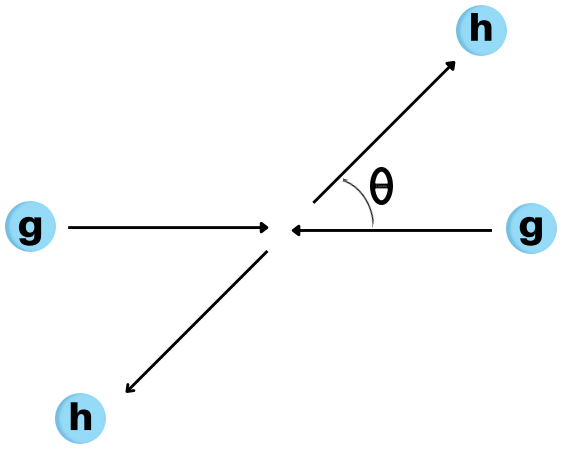

Figure 2.1: Scattering process $gg \to hh$

know the dimension of the input. In this case 2 dimensions is enough and we don't need, for example, to have 8 dimensions, which would be Energy and momentum along x, y, z, for both the particles. It's good because increasing the numbers input dimensions is more useless work for the neural network. It was always used a range of energy where the minimum was the total mass of the found particles, of course, and the maximum was arbitrary set to 3 TeV. Angles considered were only between 10 and 170 degrees to avoid possible issues related to numerical stability.

2.5 Neural Network components

A neural network is made of various components, see fig 2.2.

- A dataset $D = (X; Y)$ where X is a set of independent variables and Y is a set of dependent variables.
- A model $f(X; p)$, which is a function $f : x \to y$ of the parameters p. That is, f is a function used to predict an output from a vector of input variables
- The function $C[y; f(X; p)]$ that allows us to judge how well the model performs on the observations y. The model is fit by finding the value of p that minimizes the cost function. For example, one commonly used cost function is the squared error. Minimizing the squared error cost function is known as the method of least square

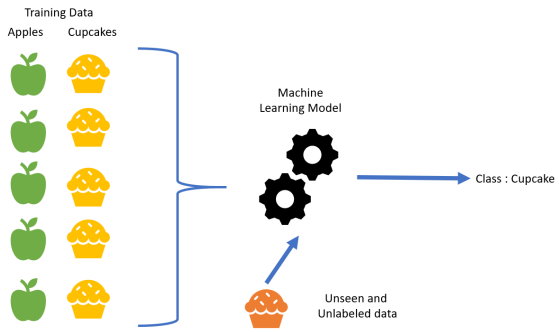

Figure 2.2: Graphic representation of how ML works

Usually a standard procedure is followed to obtain models that are useful for prediction problems.

- The first step in the analysis is to randomly divide the dataset D into two mutually exclusive groups D_{train} and $D_{test/validation}$ called the training and test/validation sets. Performing some analysis (such as using the data to select important variables) before partitioning the data is a common pitfall that can lead to incorrect conclusions.
- Typically, the majority of the data are partitioned into the training set with the remainder going into the test set, arbitrary divided 70% and 30%. The model is fit by minimizing the cost function using only the data in the training set $P = argmin_p C[Y_{train}; f(X_{train}; p)]$
- Finally, the performance of the model is evaluated by computing the cost function using the test set $C[Y_{test}; f(X_{test}; p)]$
- The value of the cost function for the best fit model on the training set is called the in-sample (or training) error $E_{in} = C[Y_{train}; f(X_{train}; p)]$ and the value of the cost function on the test set is called the out-of-sample (or generalization) error $E_{out} = C[Y_{test}; f(X_{test}; p)]$

Splitting the data into mutually exclusive training and test sets provides an unbiased estimate for the predictive performance of the model; this is known as cross-validation in the ML and statistics literature. Problems in ML typically involve inference about

complex systems where we do not know the exact form of the mathematical model that describes the system.
The basic idea of all neural networks is to layer neurons in a hierarchical way (e.g fig 2.3): The leftmost layer in this network is called the input layer, and the neurons within the layer are called input neurons. The rightmost or output layer contains the output neurons, or, as in this case, a single output neuron. The middle layer is called a hidden layer.

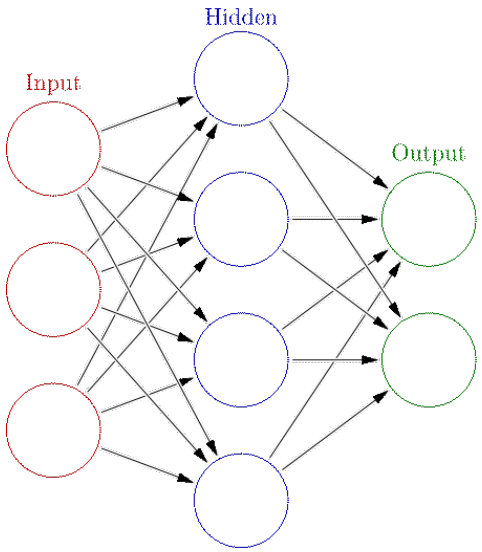

Figure 2.3: An artificial neural network is an interconnected group of nodes, akin to the vast network of neurons in a brain. Here, each circular node represents an artificial neuron and an arrow represents a connection from the output of one artificial neuron to the input of another.

2.5.1 Scaling

The scale of inputs and outputs used to train the model are an important factor. Unscaled input variables can result in a slow or unstable learning process, whereas unscaled target variables on regression problems can result in learning process to fail. Data preparation involves using techniques such as the normalization and standardization to rescale input and output variables prior to training a neural network model. Two methods are used: *Normalization* and *Standardization*. *Normalization* scales the range to $[0, 1]$, is used to transform features to be on a similar scale.

$$X_{new} = (X - X_{min})/(X_{max} - X_{min}) \tag{2.5}$$

*S*tandadization is the transformation of features by subtracting from mean and dividing by standard deviation and is not bounded to a certain range.

$$X_{new} = (X - mean)/Std \tag{2.6}$$

2.6 Neural Network used

For every interaction mentioned above, a great number of networks were made, always changing layers, activation functions, optimazer, scaling method, batch size and epoch. Not all of these above were explained. After many attempts, only the 2 best NN were kept. Both uses the *Normalization* method.

2.6.1 First NN

This is the code used to create the first. It is a function that, given the list of layers (*layers* is the argument), build a NN.

```
def baseline_model(layers, lr=0.001, activation='tanh', loss='mean_squared_error'):
      'define and compile model with a fixed dataset but random weights'
      # create model
      # at some point can use new Keras tuning feature for optimising this model
      model = Sequential()
      model.add(Dense(layers[0], input_dim=(2)))
      if activation == 'tanh':
          model.add(Activation(activations.tanh))
      elif activation == 'relu':
          model.add(Activation(activations.relu))
      else:
          raise ValueError('activation supported are either'
          'tanh or relu, you have used {}'.format(activation))

      for i in range(1, len(layers)):
          model.add(Dense(layers[i]))
          if activation == 'tanh':
              model.add(Activation(activations.tanh))
          elif activation == 'relu':
              model.add(Activation(activations.relu))

      model.add(Dense(1))
      # Compile model
      model.compile(optimizer = Adam(learning_rate=lr,
      beta_1=0.9, beta_2=0.999, amsgrad=False), loss = loss)

      return model
```

The layer used is

```
layers = [20,40,20]
```

Total parameters: 1741

2.6.2 Second NN

This is the second.

```
model = tf.keras.Sequential()
model.add(Dense(15, input_shape=(2,),activation='relu'))
model.add(Dense(40, input_shape=(2,),activation='tanh'))
model.add(Dense(45, input_shape=(2,),activation='relu'))
model.add(Dense(30, input_shape=(2,),activation='tanh'))
model.add(Dense(15, input_shape=(2,),activation='relu'))
model.add(Dense(1))
model.compile(optimizer = Adam(learning_rate=0.001,
    beta_1=0.9, beta_2=0.999, amsgrad=False), loss = 'mse')
```

Total parameters: 4391. The differences are the total number of parameters and the deep of the layers. We will see that the second one gives more accurate predictions but is slower. This is caused by the greater number of parameters: NN with many parameter need more data and more time, but gives better predictions.

2.7 Analysis of the prediction

We will see that after getting data, scaling it, training the NN and predicting some matrix elements, we need to measure the accuracy and time needed. The accuracy is obtained as the difference between predictions and real values from MadGraph, then It was made an array with all the relative errors. It was plotted an histogram of the array to see how many values with a certain precision in there.
The other important thing to do with the errors is to see how they distribute on the phase space. For example one could have a region of angles and energies where the NN struggle to make a prediction, and it is relevant to understand why it happens.
The last thing is measure the time of the training plus the time of the prediction, compare it to the time of madgraph and see if it is convenient to use ML and, in case, what is the minimum number of points to make NN a good choice.

Chapter 3

Results

Here we will use Machine Learning techniques to model the structure of a matrix element to improve the computation speed of numerical amplitudes for scattering processes at colliders. Specifically, we train a neural network to learn the features of a given scattering amplitude. We assess the performance in terms of CPU-time and of the numerical accuracy, referring also to the work [6].
The processes considered are

$$gg \to t\bar{t} \tag{3.1}$$

$$gg \to hh \tag{3.2}$$

$$gg \to ZZ \tag{3.3}$$

Indeed, we have considered time dependence as linear in both madgraph and ML calculations. In particular we used the equations of straight lines:

$$T_{MC} = t^0_{MC} N^{ev}_{MC} \tag{3.4}$$

where T_{MC} is time spent by Madgraph and N^{ev}_{MC} is number of points generated in this time, and

$$T_{ML} = t^0_{ML} N^{ev}_{ML} + t^{train}_{ML} \tag{3.5}$$

where T_{ML} is time spent by the NN, t^{train}_{ML} is the training time and N^{ev}_{ML} is, again, number of points generated in this time. Knowing T and N for both one can find t^0. We assumed the training time as a constant even though it changes with N^{ev} because of its weak dependence.
Angles considered were only between 10 and 170 degrees to avoid possible issues related to numerical stability. Using *Normalization* angles, energies and matrix elements will always be rescaled between 0 and 1 values:

$$\tilde{\theta} = \frac{\theta - \theta_{min}}{\theta_{max} - \theta_{min}} \tag{3.6}$$

$$\tilde{\sqrt{s}} = \frac{\sqrt{s} - \sqrt{s_{min}}}{\sqrt{s_{max}} - \sqrt{s_{min}}} \tag{3.7}$$

where $\sqrt{s_{min}}$ and $\sqrt{s_{max}}$ are respectively 2x mass of the product particles and 3 TeV.
Another important thing is that after the predictions data were always rescaled back to make the relative errors array. In Appendices A and B we will also provide results

for $gg \to t\bar{t}$ obtained by dividing the phase space into sub regions to be fitted by independent NN. We'll see that when excluding specific strips of angles results show lower errors, but for strips of energy it doesn't almost change anything.

3.1 $gg \to t\bar{t}$

We see that in graph from MC calculations used for ML 3.1 the matrix elements increase quickly for angles near the boundaries, 0 and π, despite is quite flat in the middle. One can suppose that the NN will struggle to model that rapid change.

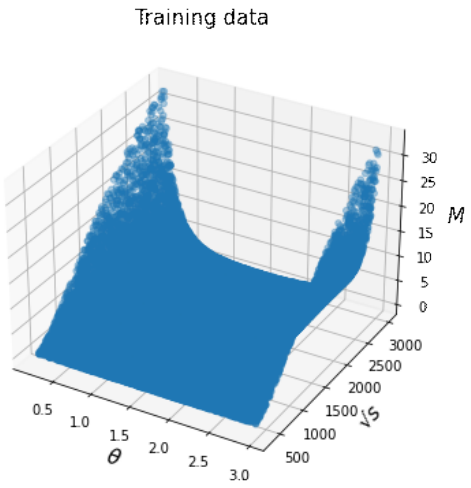

Figure 3.1: $gg \to t\bar{t}$ process, real data from MadGraph

3.1.1 First NN

In fig 3.2 we see the prediction the NN. Obviously not much can be said looking only the graph, but it's useful to get an idea. In the histogram fig 3.3 the main part of the error lies between 2 and 3, which means the errors are between $\frac{1}{100}$ and $\frac{1}{1000}$ values.
In the map fig 3.4 we see the area with worst values are the ones with maximum and minimum energy.

Time

- Time for generating 10 Millions elements madgraph [s]: 140.70
- Training time of the NN [s]: 9786.27
- Prediction time of 1000000 of matrix elements [s]: 14.29
- It comes out that the minimum number of points is $n = -361221377$

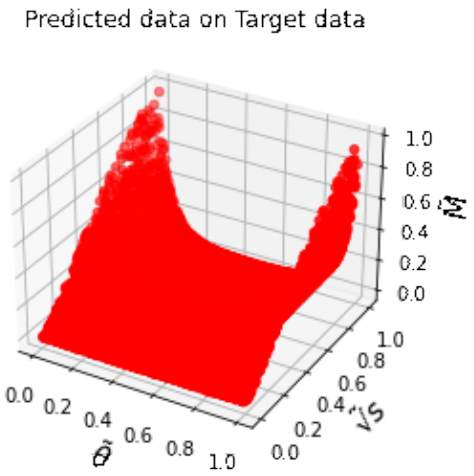

Figure 3.2: $gg \to t\bar{t}$ process, prediction first NN

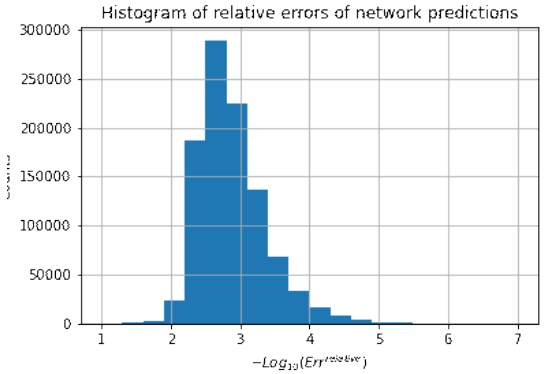

Figure 3.3: $gg \to t\bar{t}$ process, histogram of relative errors of the first NN

If the minimum number of points is negative means that madgraph computes matrix elements faster that what NN does. Therefore, for this specific and very simple process, there is no gain to use a NN with respect to the exact matrix element.

3.1.2 Second NN

Prediction in fig 3.5. Surprisingly errors in fig 3.6 are worse. Even if the error maps, fig 3.7, of this NN shows worse vales too, it's important to notice there are areas where predictions are more accurate. In fact looking at the scale of the two pictures, one sees that the second has a higher peak, of 3.75 which is absent on the other. This means that even if generally speaking the first NN is better, there are some areas where a NN can perform better than the other. It's hard to say exactly, in this specific case, why this happens. The important thing one can deduce is that with a rapid data variation the NN struggle to have always a good accuracy, so one thing to do to resolve the problem is to split the domain in many sub-regions. We will return on this in Appendix A and B.

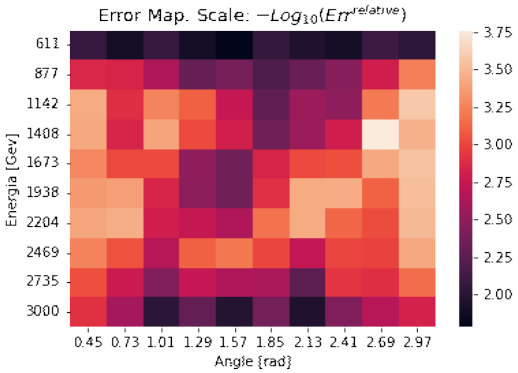

Figure 3.4: $gg \to t\bar{t}$ process, error map of the first NN. Note that the values reported for energy and angles in the intervals are the upper boundaries, which are also the lower boundaries for the next interval.

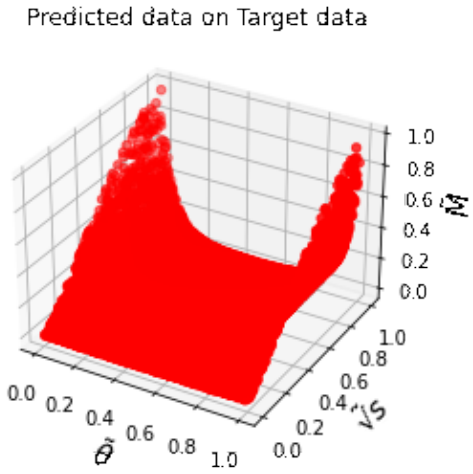

Figure 3.5: $gg \to t\bar{t}$ process, prediction of the second NN

Time

- Time for generating 10 Milions elements madgraph $[s]$: 140.70
- Training time of the NN $[s]$: 8419.79
- Prediction time of 1000000 of matrix elements $[s]$: 24.15
- It comes out that the minimum number of points is $n = -853467387$

3.1.3 Conclusion

All in all, considering the fact that without loops to compute madgraph is very quick and the prediction of the NN were not so accurate, we can say using ML for the calculations of this specific matrix element does not bring any particular advantage.

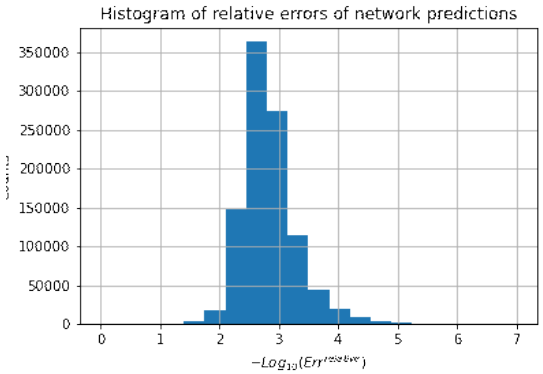

Figure 3.6: $gg \to t\bar{t}$ process, histogram of relative errors of the second NN

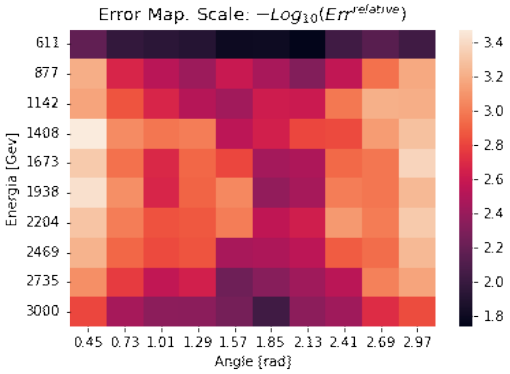

Figure 3.7: $gg \to t\bar{t}$ process, error map of the second NN.

3.2 $gg \to hh$

In picture 3.8 (and 3.9) we can see the plotted data from Monte Carlo calculation. We remark some peculiar features of the matrix element, specific to this process: moving away from the kinematic threshold, $\sqrt{s} = 2m_h$, the matrix element grows steeply for all angles; afterwards it reaches a kind of plateau, but at high energies the angular dependence is highly non flat, featuring a kind of dome. These structures may be non-trivial to be learnt by our NN.

3.2.1 First NN

In the histograms 3.10 the main part of errors are up to 10^{-3}, better that what we see previously. Unfortunately there are some bad values too. In the map 3.11 we can see that where the graph change quicker, the NN makes the worst predictions. In particular in the region close to threshold, and in parts where the dome grows fast. All of this is another example of what was said before.

Time

- Time for generating 10 Millions elements madgraph [s]: 29845.92

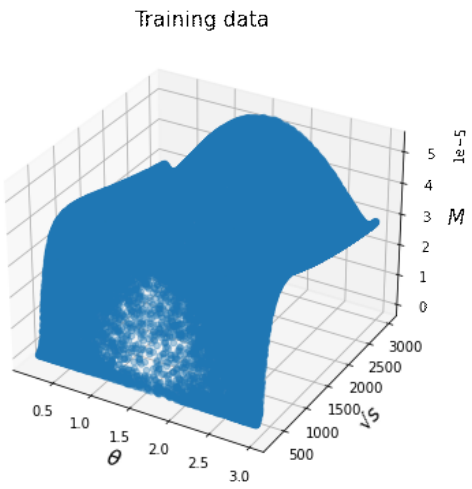

Figure 3.8: $gg \to hh$, real data from MadGraph

- Training time of the NN $[s]$: 3490.95
- Prediction time of 1000000 of matrix elements $[s]$: 13.91
- It comes out that the minimum number of points is $n = 1175110$

This time n is positive, which means the NN can predict elements faster than Madgraph. After a number of n points generated, the NN became convenient in terms of time needed, time for training included.

3.2.2 Second NN

The behavior here is similar to the other one, fig 3.12 . Looking at histogram 3.13 and at map 3.14, the first NN is better. The same consideration made before are still valid.

Time

- Time for generating 10 Millions elements madgraph $[s]$: 29845.92
- Training time of the NN $[s]$: 10885.55
- Prediction time of 1000000 of matrix elements $[s]$: 23.99
- It comes out that the minimum number of points is $n = 3676715$

The time needed for training and prediction is much higher, due to the greater complexity of the NN. As consequences, n is bigger.

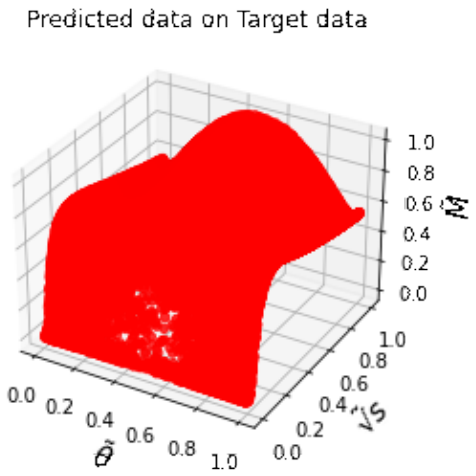

Figure 3.9: $gg \to hh$, prediction of the first NN

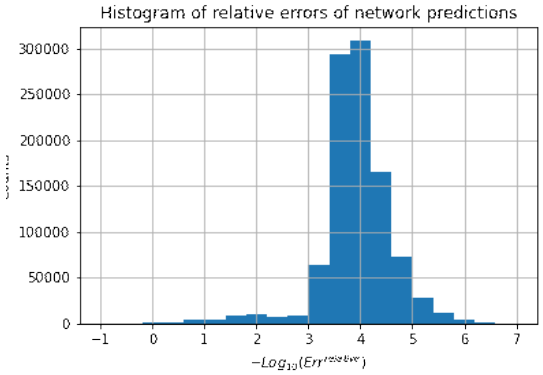

Figure 3.10: $gg \to hh$, histogram of the relative error of the first NN

3.2.3 Conclusion

For this process, errors in the histograms are between 3 and 4, better than $gg \to t\bar{t}$. However, some areas are result difficult to model. In fact even if the average values of the worst areas are 2.4, the histograms shows some very bad values, some of them are between 0 and 1, which means that the relative error is almost as big as the matrix element itself.
Time results are satisfying, especially for the first NN, only 1175110 points to generate means the NN is convenient. In conclusion, for the better areas of the maps the NNs can be trusted quite easily, not the same for the worst unless one is willing to accept that among their values there are ones with great errors.

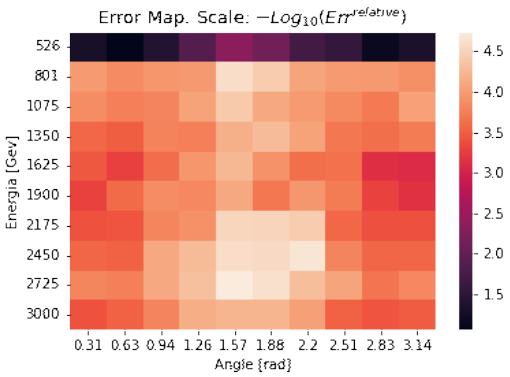

Figure 3.11: $gg \to hh$, map of the errors of the first NN

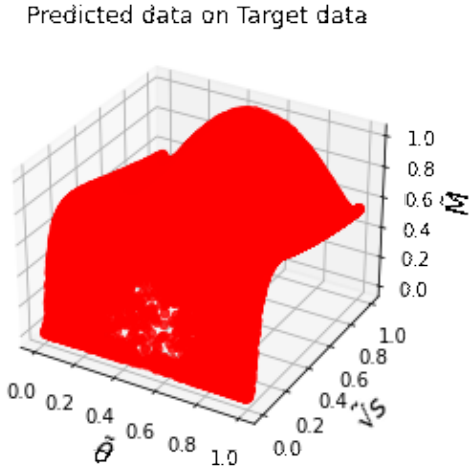

Figure 3.12: $gg \to hh$, prediction of the second NN

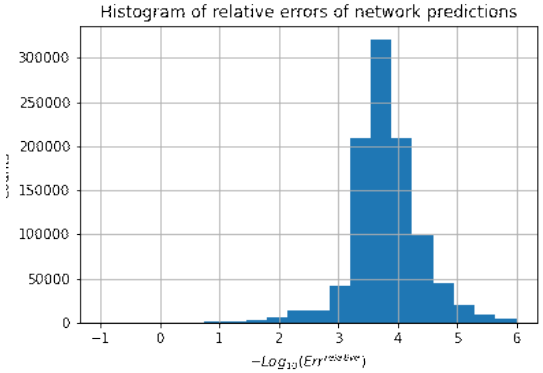

Figure 3.13: $gg \to hh$, histograms of the relative errors of the second NN

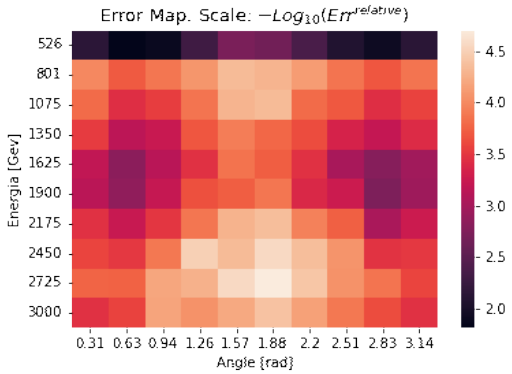

Figure 3.14: $gg \to hh$, error map of the second NN

3.3 $gg \to ZZ$

For this last process we use a different angle and energy cut. The reason is a numerical instability in madgraph which cause to the matrix elements have huge no-sense values. Thus, energy cut is 200 GeV, thus above the kinematic threshold for this process and angle cut is at 30 and 150 degrees. This will also allow the NN to be more efficient. A graphic representation is fig 3.15.

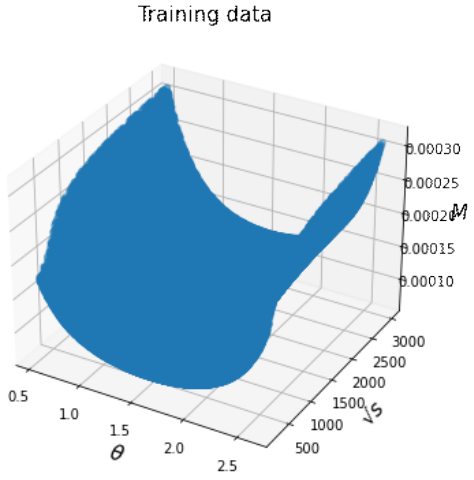

Figure 3.15: $gg \to ZZ$, real data from madgraph

3.3.1 First NN

In fig 3.16, the prediction of this NN. The great result shown in the histogram fig 3.17

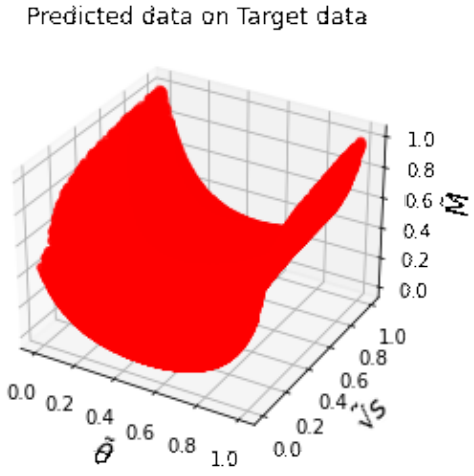

Figure 3.16: $gg \to ZZ$, prediction of the first NN

is the absence of values below 2, so every value predicted has relative error at least up to $\frac{1}{100}$. The map looks better 3.18 than in the other processes, but surprisingly bad for

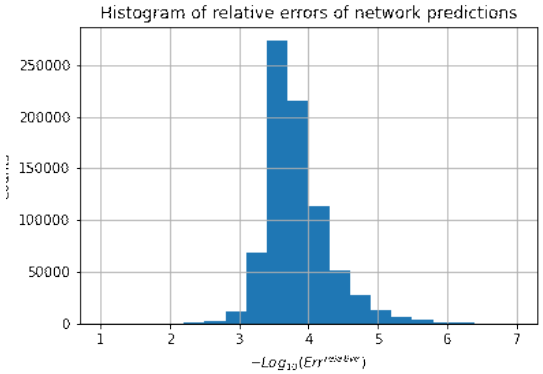

Figure 3.17: $gg \to ZZ$, histogram of the first NN

low energies. Probably it is caused by that kind of step we can see in the graph.

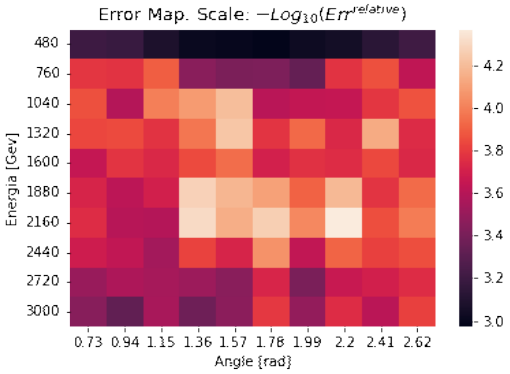

Figure 3.18: $gg \to ZZ$, map of the first NN

- Time for generating 8 Millions elements madgraph [s]: 93624.08
- Training time of the NN [s]: 11240.97
- Prediction time of 794466 of matrix elements [s]: 16.26
- It comes out that the minimum number of points is $n = 696595$

Due to the slowness of Madgraph, n is very low. For $gg \to ZZ$ process, timing for this first NN is very convenient.

3.3.2 Second NN

Prediction of the NN in fig 3.19. This time, the second NN performs approximately the same. In histogram 3.20 we see that only a few data are under 3, this makes the NN very solid.

In fig 3.21 we see the scale is good, the worst areas are around 3.0 on average.

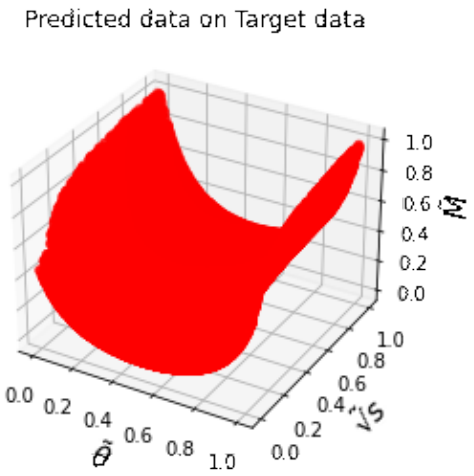

Figure 3.19: $gg \to ZZ$, prediction data second NN

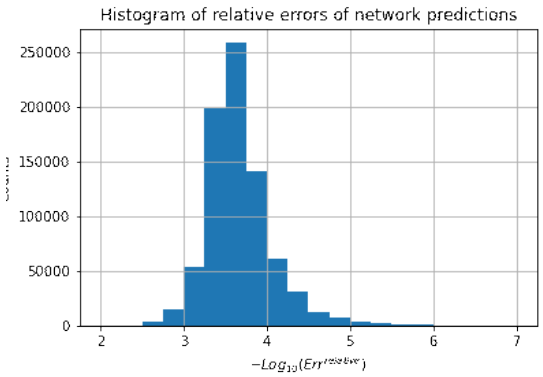

Figure 3.20: $gg \to ZZ$, histogram of the second NN

Time

- Time for generating 10 Millions elements madgraph $[s]$: 93624.08
- Training time of the NN $[s]$: 11523.97
- Prediction time of 794466 of matrix elements $[s]$: 19.85
- It comes out that the minimum number of points is $n = 714332$

Just a bit slower than the first.

3.3.3 Conclusion

$gg \to ZZ$, is the process that gave the better results and there are many factors that justify it. One of them is angle and energy cuts, which reducing the domain let the data be more regular. Another is the huge amount of time needed to madgraph to calculate the matrix elements, bigger by far then what we had for $gg \to hh$, even if both of them

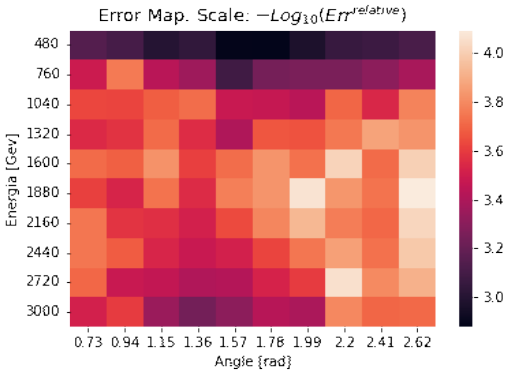

Figure 3.21: $gg \to ZZ$, map of the second NN

are loops. This time the ML techniques represent a valid alternative to MC algorithms, if a precision up to 10^{-3} is enough.

Chapter 4

Conclusion and Outlook

With the beginning of the Run III, and in view of the future High-Luminosity run, a steep increase is foreseen for the computational load needed to match the very high statistics of experimental data.
In this work we studied the gain from having used Machine Learning techniques to model the structure of a matrix element to improve the computation speed of numerical amplitudes for scattering processes at colliders, previously calculated with Madgraph which uses Monte Carlo algorithms.
Processes involved were $gg \to t\bar{t}$, $gg \to hh$ and $gg \to ZZ$.
The first one gave the worst result: considering that being a very simple process, the matrix-element evaluation is already very quick, besides the prediction of the NN were not so accurate. We can say using ML for the calculations of this specific matrix element does not bring any particular advantage.
On the contrary, $gg \to hh$ resulted in more convenient in terms of time needed, because the NNs used could predict matrix elements faster then what Madgraph did. Even though relative errors are mainly between 10^{-3} and 10^{-4} some areas are difficult to model, resulting in errors between 1 and 0.1.
The last process happened to be the one for which the gain is the largest. In terms of time used, NNs are very suitable, especially due to the huge amount of time needed by Madgraph for loop computation. The worst values of relative errors are of the order of 10^{-3}. One could say that ML techniques represent a valid alternative to MC algorithms in this case, if a precision up to 10^{-3} is enough.
We have observed difficulties for NNs to reproduce non trivial structures of the matrix elements, such as threshold enhancements. These kind of structures should therefore be removed, for example with suitable change of variables, similarly to what is done for numeric integration.
Alternatively, or in combination to that, one could divide the phase space in subsets [10]. In this case peaks are isolated and trained individually. This will be crucial for processes including more than 2 particles in the final state. Again, we stress that these challenges are very similar to the ones typical of numeric integration.
Given the results, the usage of NN to replace slow matrix elements may represent an opportunity to find a innovative way of approaching the problem of amplitude evaluation in high energy physics.

Appendix A

Domain cut for $gg \to t\bar{t}$ process

In this section we will look further if dividing the domain in sub-areas increases the performance of the NN, in terms of accuracy and time.

A.1 Angle Cut

Here we consider the same cut used for $gg \to ZZ$: from 30 to 150 degrees. In fig A.1 how graph looks like.

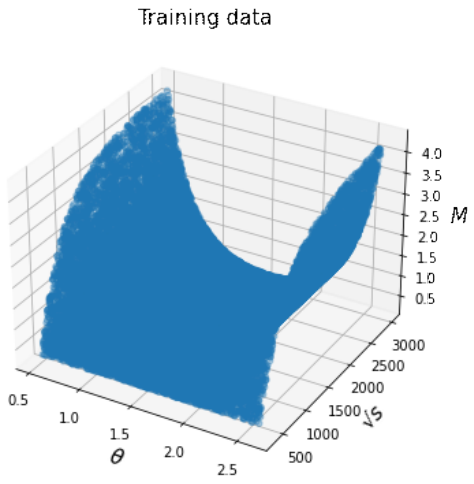

Figure A.1: $gg \to t\bar{t}$ with angle cut, real data

A.1.1 First NN

In fig A.2 the prediction made. The difference between histograms A.3 and 3.3 is evident, errors became 10 times smaller.

It's a huge improvement to have 2.8 on average in map A.4.

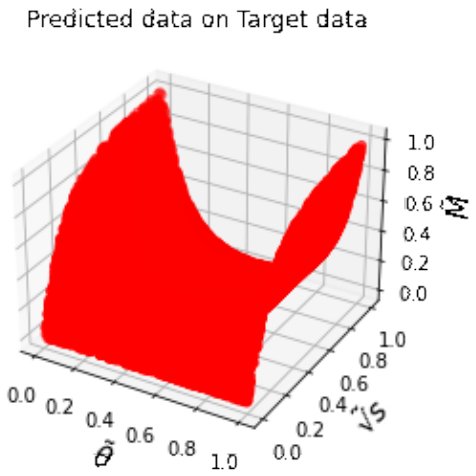

Figure A.2: $gg \to t\bar{t}$ with angle cut, prediction of first NN

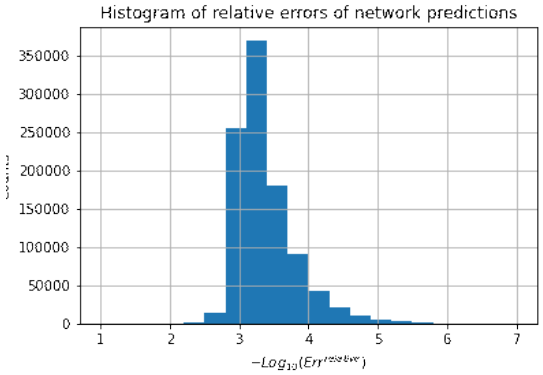

Figure A.3: $gg \to t\bar{t}$ with angle cut, histogram of first NN

Time

- Time for generating 10 Millions elements madgraph [s]: 140.697998
- Training time of the NN [s]: 6507.86
- Prediction time of 794466 of matrix elements [s]: 21.47
- It comes out that the minimum number of points is $n = -878932793$

The NN is still not a good choice due to the excessive time for prediction and training. However one notices the marked improvement compared to the first NN (and the second too) in the other $gg \to t\bar{t}$ process.

A.1.2 Second NN

Looking at predictions fig A.5 and histogram A.6 we can say that what we said for the first NN is still valid.

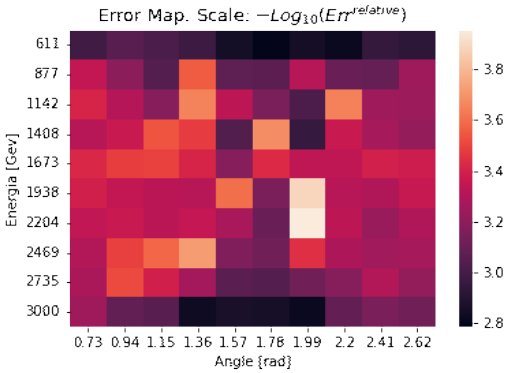

Figure A.4: $gg \to t\bar{t}$ with angle cut, map of first NN

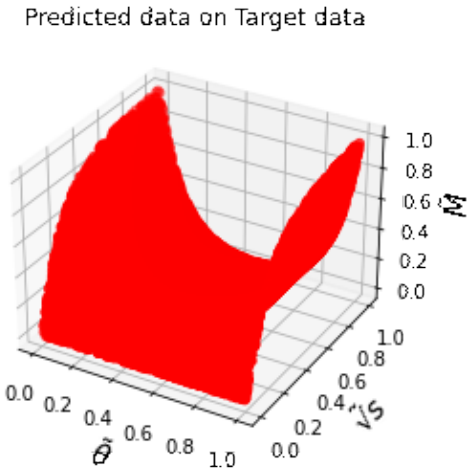

Figure A.5: $gg \to t\bar{t}$ with angle cut, prediction of second NN

Errors look pretty much the same, so even with this second NN the improvement is remarkable. Error map in fig A.7

Time

- Time for generating 10 Millions elements madgraph $[s]$: 140.697998
- Training time of the NN $[s]$: 9360.81
- Prediction time of 794466 of matrix elements $[s]$: 25.07
- It comes out that the minimum number of points is $n = -850766302$

A.1.3 Conclusion

The new NNs demonstrated to have better accuracy and shorter times for both training and prediction time, even through MadGraph times are definitely shorter. This confirms the idea that dividing the phase space in subsets is a valid way to improve performance.

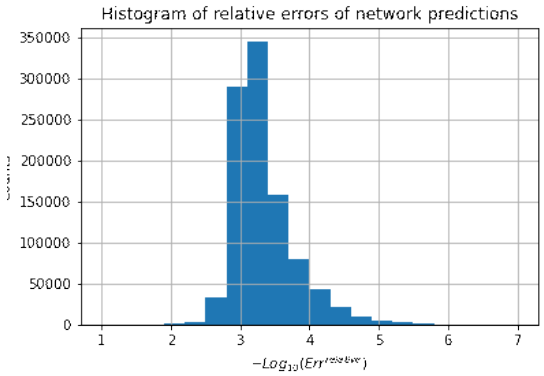

Figure A.6: $gg \to t\bar{t}$ with angle cut, histogram of second NN

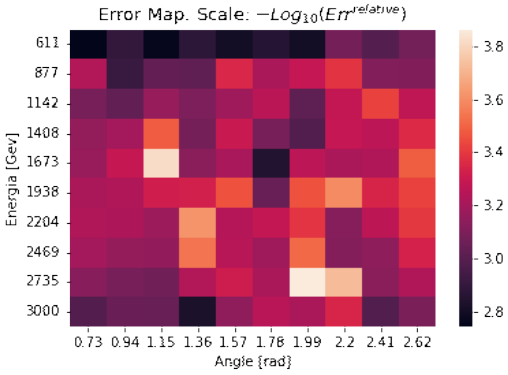

Figure A.7: $gg \to t\bar{t}$ with angle cut, map of second NN

Appendix B

Energy Cut

We will see now how dividing energy values in 2 subsets affects NN's performance. The cut is in the middle of the thresholds, which is approximately 1672.77 GeV. We will use the usual 2 NNs for every subset. Another important aspect is that this time points used per training and validation for each NN were less due to the division: 2.1 Million per training and 0.9 Million per validation, and 1 Million per prediction.

B.1 Cut from 1672.77 to 3000 GeV

The part of points considered is represented in fig B.1.

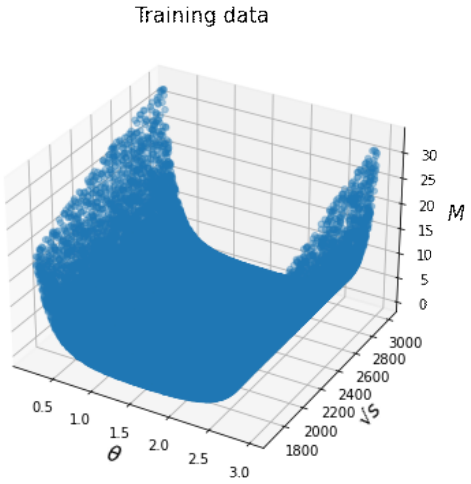

Figure B.1: $gg \to t\bar{t}$ with energy cut [1672.77,3000] GeV, real data from madgraph

B.1.1 First NN

As usual, data scaled in fig B.2. Looking at histogram B.3 we see the absence of values under 2, which makes the NN more reliable. Map B.4 reflects difficulties in the same zones found without cuts too.

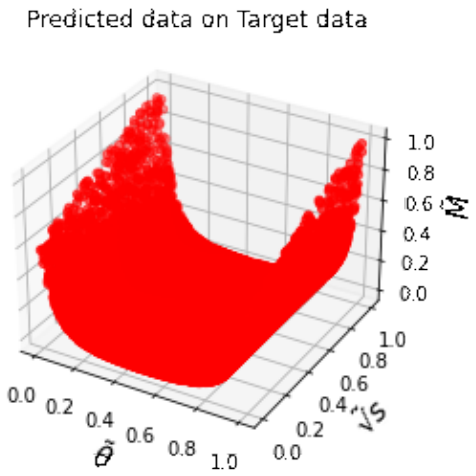

Figure B.2: $gg \to t\bar{t}$ with energy cut [1672.77,3000] GeV, first NN prediction

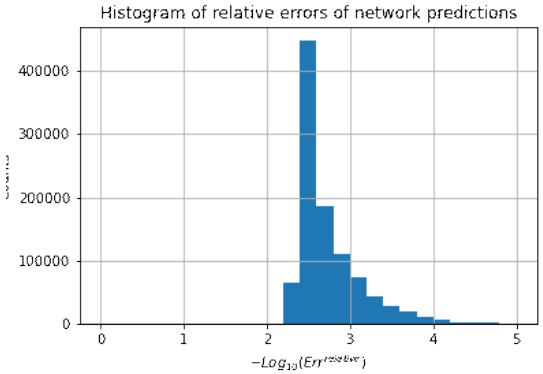

Figure B.3: $gg \to t\bar{t}$ with energy cut [1672.77,3000] GeV, histogram of the first NN

Time

- Time for generating 10 Millions elements madgraph [s]: 140.697998
- Training time of the NN [s]: 7804.82
- Prediction time of 794466 of matrix elements [s]: 22.99
- It comes out that the minimum number of points is $n = -897178846$

Time is still disappointing compared to madgraph, but we expected it. Speed performance of the NN are not much different from the first $gg \to t\bar{t}$ treated, probably because the most difficult part of the matrix structure, the one near to the threshold of angles, was not cut.

B.1.2 Second NN

Data predicted are in fig B.5. Both histogram B.6 and map B.7 show values slightly worse than those of the first NN. They appear to be traslated downwards of 0.4.

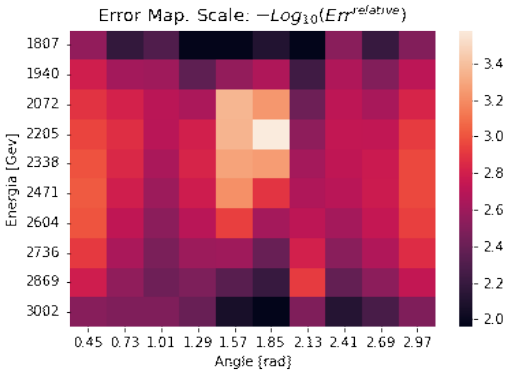

Figure B.4: $gg \to t\bar{t}$ with energy cut [1672.77,3000] GeV, errors map of the first NN

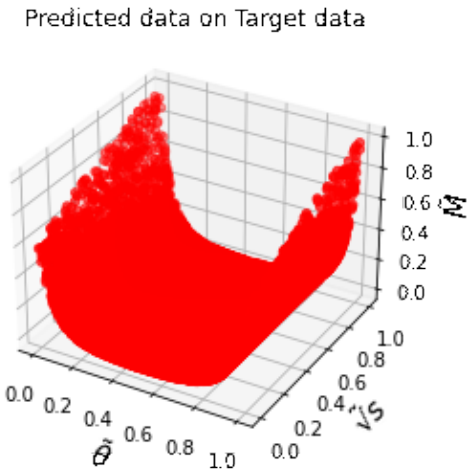

Figure B.5: $gg \to t\bar{t}$ with energy cut [1672.77,3000] GeV, second NN prediction

Time

- Time for generating 10 Millions elements madgraph [s]: 140.697998
- Training time of the NN [s]: 3122.79
- Prediction time of 794466 of matrix elements [s]: 23.29
- It comes out that the minimum number of points is $n = -346696146$

Time taken for the training is much less than all the other, but not time of the prediction, indeed n is still a bad value.

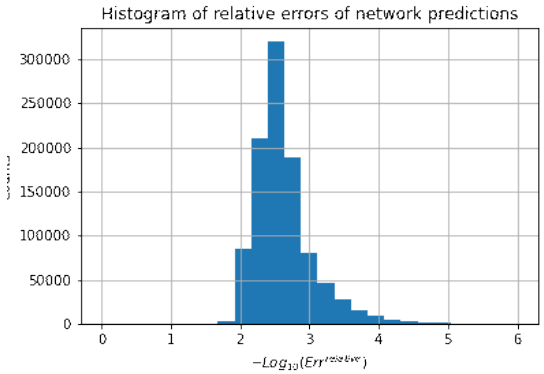

Figure B.6: $gg \to t\bar{t}$ with energy cut [1672.77,3000] GeV, histogram of the second NN

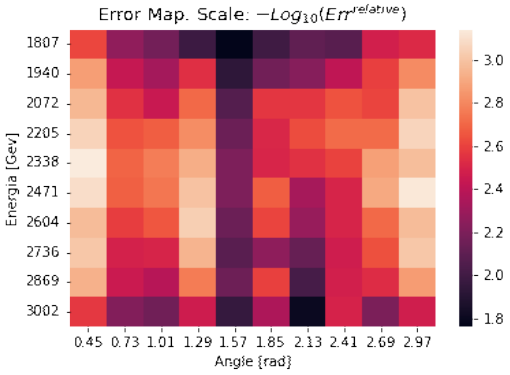

Figure B.7: $gg \to t\bar{t}$ with energy cut [1672.77,3000] GeV, errors map of the second NN

B.2 Cut from 346.2 to 1672.77 GeV

The part of points considered is represented in fig B.8

B.2.1 First NN

Predicted data in fig B.9. Histogram B.10 shows worse values, probably caused by the fact that for slow values of energy the matrix structure increases more rapidly near to the threshold of angle. Errors in map B.11 and in the histogram are pretty much the same for the process with the whole space phase.

Time

- Time for generating 10 Millions elements madgraph [s]: 140.697998
- Training time of the NN [s]: 4122.22
- Prediction time of 794466 of matrix elements [s]: 21.44
- It comes out that the minimum number of points is $n = -576543536$

The only real gain is time, which is much shorter, especially due to the lower number of point used.

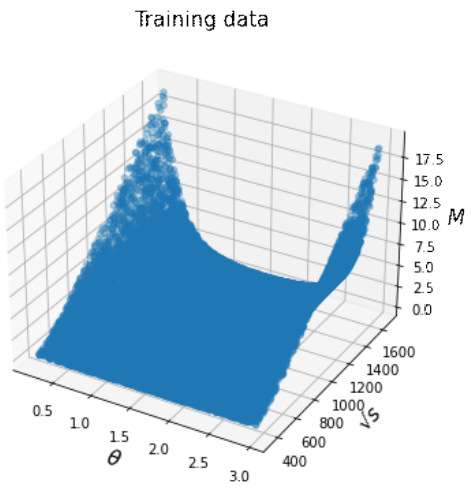

Figure B.8: $gg \to t\bar{t}$ with energy cut [346.2,1672.77] GeV, real data from madgraph

B.2.2 Second NN

Data predicted are in fig B.12.

There nothing to add to what was already said about histogram B.13 and map B.14.

Time

- Time for generating 10 Millions elements madgraph [s]: 140.697998
- Training time of the NN [s]: 5865.25
- Prediction time of 794466 of matrix elements [s]: 26.03
- It comes out that the minimum number of points is $n = -499358746$

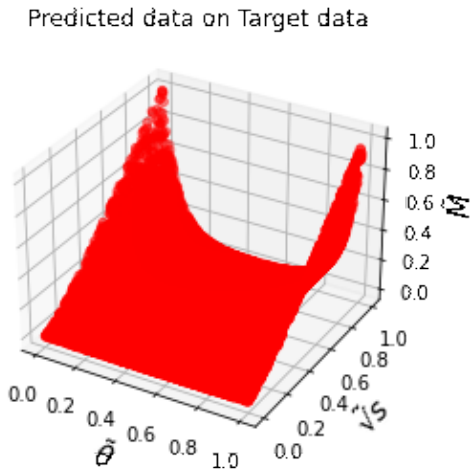

Figure B.9: $gg \to t\bar{t}$ with energy cut [346.2,1672.77] GeV, first NN prediction

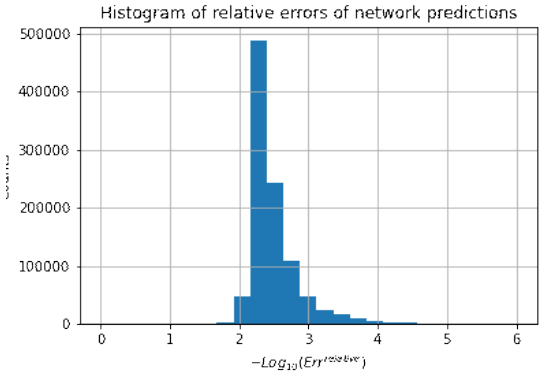

Figure B.10: $gg \to t\bar{t}$ with energy cut [346.2,1672.77] GeV, histogram of the first NN

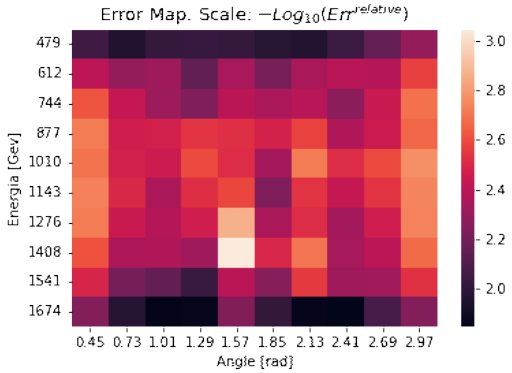

Figure B.11: $gg \to t\bar{t}$ with energy cut [346.2,1672.77] GeV, errors map of the first NN

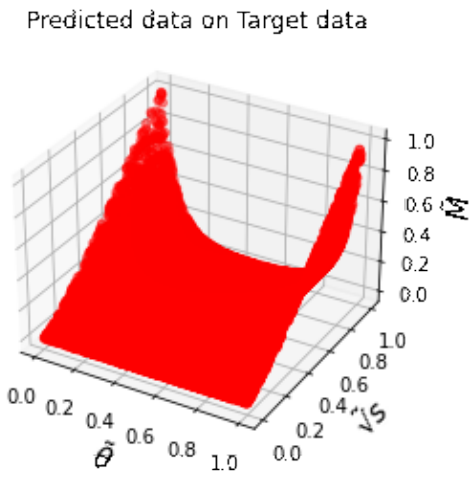

Figure B.12: $gg \to t\bar{t}$ with energy cut [346.2,1672.77] GeV, second NN prediction

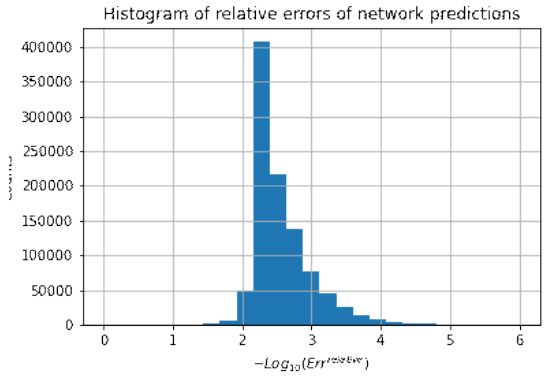

Figure B.13: $gg \to t\bar{t}$ with energy cut [346.2,1672.77] GeV, histogram of the second NN

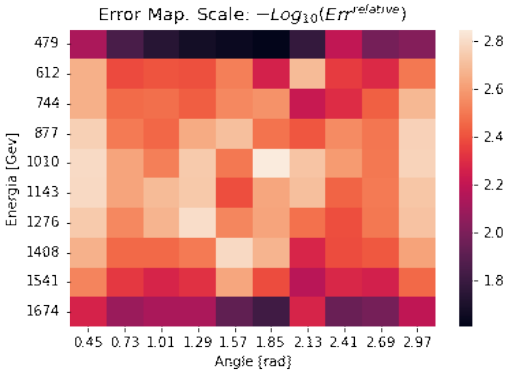

Figure B.14: $gg \to t\bar{t}$ with energy cut [346.2,1672.77] GeV, errors map of the second NN

B.3 Conclusion

Unlike the results of the angle cut, this time it's hard to see a substantial gain. Indeed, the most difficult part to model for the NNs, the one near to the threshold of the angle, was not isolated. This caused to the NNs to have the same problems encountered before. From this experience we understand that when we know the hardest areas to reproduce with the NN, the smartest thing to do is to divide the phase space in order to train these regions alone.

Bibliography

[1] G. Aad, T. Abajyan, B. Abbott, J. Abdallah, S. A. Khalek, A. A. Abdelalim et al., *Observation of a new particle in the search for the standard model higgs boson with the atlas detector at the lhc*, *Physics Letters B* **716** (2012) 1–29.

[2] CMS collaboration, S. Chatrchyan et al., *Observation of a New Boson at a Mass of 125 GeV with the CMS Experiment at the LHC*, *Phys. Lett. B* **716** (2012) 30–61, [1207.7235].

[3] P. Nason, *Introduction to perturbative qcd*, in *Particles And Fields*, pp. 409–486. World Scientific, 2002.

[4] P. Calafiura, J. Catmore, D. Costanzo and A. Di Girolamo, *Atlas hl-lhc computing conceptual design report*, tech. rep., 2020.

[5] J. Alwall, M. Herquet, F. Maltoni, O. Mattelaer and T. Stelzer, *Madgraph 5: going beyond*, *Journal of High Energy Physics* **2011** (2011) 1–40.

[6] F. Bishara and M. Montull, *(machine) learning amplitudes for faster event generation*, *arXiv preprint arXiv:1912.11055* (2019) .

[7] J. Notebook], "Jupyter notebook."

[8] Goolge, "Tensorflow."

[9] "condor."

[10] S. Badger and J. Bullock, *Using neural networks for efficient evaluation of high multiplicity scattering amplitudes*, *Journal of High Energy Physics* **2020** (jun, 2020) .