UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Fisica e Astronomia "Galileo Galilei"

Corso di Laurea in Fisica

Tesi di Laurea

Development of a ML-based model-independent analysis strategy at the LHC

Relatore

Prof. Marco Zanetti

Laureando

Rocco Ardino

Anno Accademico 2018/2019

Abstract

Searching for New Physics is the primary goal of the CMS experiment at the LHC. Performing such search without relying on a specific theory extending the Standard Model (SM) is of paramount importance but at the same time highly non-trivial. Recently, several proposal have been made in that sense, in particular exploiting the power of modern ML techniques. The goal of this thesis is to apply such model-independent strategies to concrete physics cases, aiming at detecting signals solely by comparing the collision data with what predicted by the SM.

Contents

Chapter 1

Introduction

1.1 The progress of Particle Physics

Today in High-Energy Particle Physics there are several theoretical models which are able to describe the outcome of almost every present experiment. One of them, probably the most advanced one, is the Standard Model (SD) and it will be called in the following as reference model. However, althought SM has proved to be extremely successful in predicting a wide variety of particle processes with great accuracy, there are still unexplained phenomena. In fact:

- it doesn't explain gravitation;
- it doesn't account a mass for neutrinos;
- it doesn't explain the existence of dark matter.

Future experiments will be able to explore never observed before phenomena, or they will mesaure known phenomena with a even better accuracy. These facts suggest that new physics (i.e. physical laws not yet established) exists and its research is actually one of the most challenging problems in High Energy Physics.

1.1.1 The research for new physics

Searching for new physics concretely means searching for discrepancies between the data and the reference model. This problem can be phrased in a more technical way. What we are able to do is take repeated measurements of a multi-dimensional random variable x, experimentally speaking. Then, we can build a Probability Density Function (PDF) using experimental data and test the reference model distribution against the actual data. The first difficulty encountered with this approach is that the true underlying data distribution will be quite similar to the reference one. It means that, if data contain new physics effects, they will be localized in a low-probability region where only a small fraction of event is present, or they will be spread in a large region of the x space.

The most widely employed approach to the problem is to search for specific new physics models. It has the advantage to be physically informative, even if the compatibility of the data with the reference model is confirmed. But there is a critical disadvantage: a statistical test which is designed to be sensitive to one specific hypothesis is typically insensitive to data departures of a different nature from the one expected. So, even if new physics is present in the data, it would not be discovered because it doesn't belong to the class of hypothetical models we are searching for.

1.1.2 The need for a model-independent approach

It is necessary to define the meaning of 'model-independent'. We call a certain approach 'model-independent' when the alternative distributions don't follow from a physical model, but are selected with other criteria, such as flexibility. It means that the distributions can adapt to the true underlying data distribution for an appropriate choice of the free parameters.

The main reason for which we demand a model-independent approach is the advantage of sensitivity to a large variety of new physics scenarios, including those that are not predicted by any of the models constructed until now.

1.2 Why do we choose Neural Networks?

Neural Networks are increasing their importance in high energy physics since they are widely employed. The main reason for their success is due to their properties of unbiased approximants. In fact, it is possible to prove mathematically that a Network of a certain complexity can approximate every function sufficiently regular. Employing them to parametrize alternative distributions for model-independent new physics searches is thus a highly motivated attempt.

Technically speaking, we can exploit Neural Networks to find an anomalous behavior, relative to the reference model, of the entire data sample with which we train the Network.

Chapter 2

Introduction to Neural Networks

2.1 Artificial Neural Networks

Nature has inspired many inventions, so it seems logical to look at the architecture of brain for inspiration on how to build an 'intelligent machine'. This is the key idea which inspired Artificial Neural Networks (a.k.a. ANNs). In fact, an ANN is a computing system inspired by the biological neural networks which consitute animal brain. Therefore, an ANN is based on a set of connected units or nodes called artificial neurons for the comparison with biological neurons. Each connection, like the synapses in a biological brain, can transmit a signal from one artificial neuron to another. An artificial neuron that receives a signal can process it and then signal additional artificial neurons connected to it. For the sake of simplicity, from now on out we will denote ANN with Neural Networks, or more simply NNs.

In common NN implementations, neurons are arranged in consecutive layers, with the first and last ones called respectively input and output layers. Each layer can be fully connected with the subsequent layer, i.e. every single neuron of a layer is connected with all the neurons in the next layer. It is the case of a fully connected Neural Network. In other implementations, neurons can also be only partially connected in the meaning previously defined. For example, taking into account a network with 4 input neurons, 2 output neurons and a hidden layer with 8 neurons, the difference between the two types is highlighted by Figure 2.1 and 2.2.

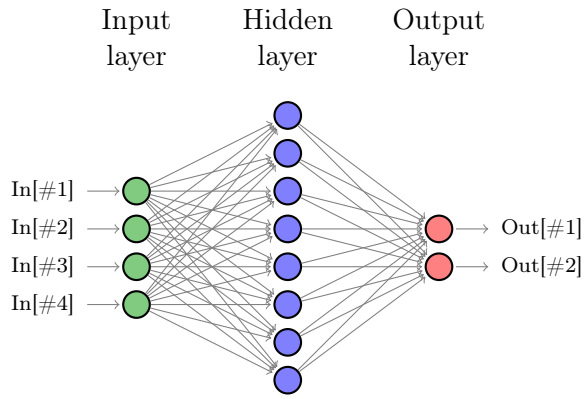


Figure 2.1: Fully connected ANN.

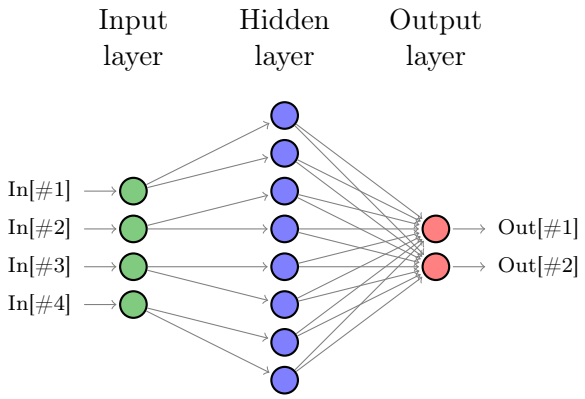


Figure 2.2: Partially connected ANN.

We will consider fully connected NNs to explain in a more technical manner their working principles. In particular, we will consider a network composed of a number of layers L. A Neural Network can be described mathematically by the composition of several function (nested units) associated to a certain layer:

$$f_{\mathrm{NN}}(\vec{x}) = f \circ g \circ h \circ \cdots \circ \vec{x} = f(g(h(\ldots(\vec{x})))) \tag{2.1}$$

with $\vec{x}$ the input of the NN and f_{NN} its output. The units in this representation can be of two kinds:

- Linear transformations:
$$h(\vec{x}) = \mathbf{w}^l \vec{x} + \vec{b}^l \tag{2.2}$$
where:
  - ▷ $\mathbf{w}^l$ is a matrix of free parameters, called weights, whose elements $[\mathbf{w}]^l_{i,j}$ are the connections values between the j^{th} neuron in the $(l-1)^{\text{th}}$ layer and the i^{th} neuron in the l^{th} layer.
  - ▷ $\vec{b}^l$ is a vector of free parameters, called biases, of the neurons in the l^{th} layer.
- Non-linear transformations, which don't depend on free parameters:
$$g(\vec{x}) = [g(x_1), \dots, g(x_{\text{N}})] \tag{2.3}$$

It is possible to deduce through this formalism that a Neural Network is a way to parametrize a set of functions, whose elements are spanned by weights and biases. Given the number of layers L of the NN, this set will be denoted as $\mathcal{F}_{\text{L}}$:

$$\mathcal{F}_{\text{L}} = \{f_{\text{NN}}(x; \mathbf{w}, \vec{b}),\ \forall\ \mathbf{w}, \vec{b}\} \tag{2.4}$$

The question of interest now is how to find the parameters $\mathbf{w}$ and $\vec{b}$ that suit best to our purpose. It is possible to achieve this accomplishment through a process called training, which is one of the key ideas behind the philosophy of Neural Networks and Deep Learning. Hence we need to implement an algorithm capable of updating the free parameters of the net to male them suitable to describe a dataset $\mathcal{D}$ given to the input layer. Moreover, a good choice of the architecture of the NN, of the algorithm and of the weights and biases could generalize the results of the procedure to a bigger dataset $\tilde{\mathcal{D}}$.

In the following sections a step by step procedure of the training process will be explained. Moreover, several tecniques will be listed, considering that there isn't a fixed scheme in the algorithm, but a certain choice can fit better to a certain problem than another one.

2.2 Choice of NN architecture and activation functions

The flexibility of NNs is also one of their main drawbacks. In fact, there are many hyperparameters to tweak. But what do we mean with the word hyperparameter? It was previously described that a NN can parametrize a wide set of functions with the right choice of weights and biases, which are the free parameters of the net. however, there are many other parameters we have to consider and tune if we want to answer our demands. They are going to be presented step by step in the implementation of the learning procedure.

The first thing we have to take into account in the development of a NN is its achitecture, i.e. the number of layers and the number of neurons for each layer. The architecture of a NN is the first of the hyperparameters of the net. After this step, the activation functions for every layer are to be selected among a wide variety of solutions.

2.2.1 Number of hidden layers

For many problems, it is possible to begin with a single hidden layer and get reasonable results. As a matter of facts, it can be proved that a NN just like the previous one can model even the most complex functions provided it has enough neurons. This was the reason for which many researchers thought for a long time that there was no need to investigate any deeper NNs. However deep networks (i.e. with more than one hidden layer) are much more powerful than shallow ones. They have a much higher parameter efficiency because they are capable of modelling complex functions using exponentially fewer neurons than shallow nets. Through this expedient the train process could be sped up for the fewer number of free parameters.

Another important reason to prefer deep networks for more complex problems has a heuristic justification. In fact, this kind of NNs is more adaptable to the hierarchical structure of real-world data.

Lower hidden layers model low-level features, intermediate hidden layers combine these low-level features to model intermediate-level features, and the highest hidden layers and the output layer combine these intermediate features to model high-level features. Moreover, not only does this hierarchical architecture help deep NNs converge faster to a good solution, it also improves their ability to generalize to new datasets.

In summary, for many problems it is possible to start with just one or two hidden layers and it will work just fine. For more complex problems, the strategy is to gradually raise the number of hidden layers until the model starts overfitting the training dataset.

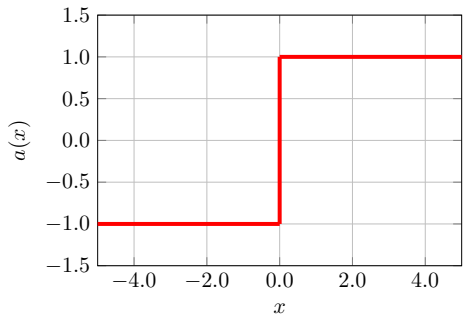
2.2.2 Number of neurons per hidden layer

Obviously the number of neurons in the input and output layers is determined by the type of input and output required by the problem. For the hidden layers, a common practice is to size them to form a funnel, with fewer and fewer neurons at each layer. This possible architecture is motivated by the fact that many low-level features can combine into far fewer high-level features. However, this practice is not as common now and it is a valid solution to use the same size for all hidden layers. The advantage of the last practice is that there is only one hyperparameter to tune for the number of neurons per layer, instead of one hyperparameter per layer. Just like for the number of layers, we can try to increase the number of neurons gradually until the network starts overfitting the data.

In general, it is more productive to increase the number of layers than the number of neurons per layer. Finding the perfect amount of neurons is quite difficult. A simpler approach is to pick a model with more layers and neurons than we actually need and then stop the training process earlier in order to prevent it from overfitting. It is worth to employ also some regularization techniques that will be presented in the following sections.

2.2.3 Activation functions

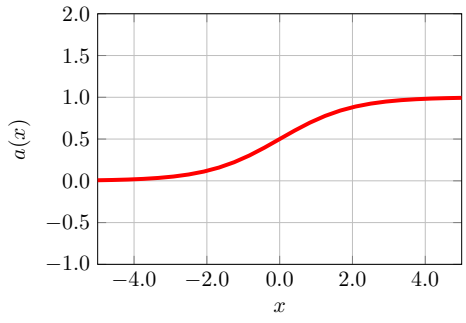
For each training instance, the algorithm employed feeds it to the network and computes the output of every neuron in each consecutive layer. This output is given by a function called 'activation'. The selection of the activation functions for every layer is not only pretty important if we want to get excellent results, but it is also crucial to speed up the computation, in particular if the expected training time is relative long. It is possible to assemble a network with a different activation per layer depending on what we want to do. Several functions that suits to the purpose are presented below along with their advantages and disadvantages.



Step function

$$a(x) = \begin{cases} 1 & x \geq 0 \\ -1 & x < 0 \end{cases} \tag{2.5}$$

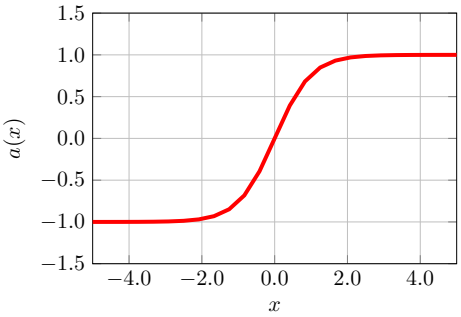
A neuron with this activation is called linear threshold unit (LTU). A single LTU can be used for simple linear binary classification problems. In general a network of LTU could not be sufficient to get reasonable results.



Sigmoid function

$$a(x) = \frac{1}{1 + \mathrm{e}^{-x}} \tag{2.6}$$

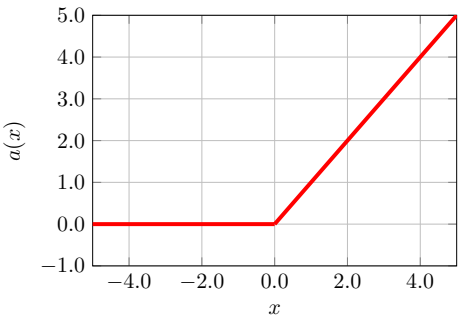
The sigmoid function has a well-defined nonzero derivative everywhere, allowing the algorithms presented in next sections to work properly and make progresses at every step. It is probably one of the most employed for its properties.



Hyperbolic tangent function

$$a(x) = \tanh x \tag{2.7}$$

Just like the sigmoid function it is S-shaped, continuous and differentiable, but the difference is that the output value ranges from -1 to 1 (instead of 0 to 1), which tends to make each layer's output more or less normalized at the beginning of training. This often help to speed up convergence.



ReLU function

$$a(x) = \max(0, x) \tag{2.8}$$

It is continuous but not differentiable at $x = 0$. However, in practice it works very well and has the advantage of being very fast to compute. It does not have a maximum output value.

2.3 Loss functions

Another key concept of the training algorithm is evaluating if the network is working right. For this purpose, we have to quantify how much network predictions are wrong and we can do it by choosing a 'loss function', or also called 'cost function' or 'objective function'. There is a wide variety also for this kind of functions, but they are more related to the problem to solve than the activations. Choosing a wrong loss means getting worse results for our model. Sometimes a loss function is built from scratch to solve a single problem, so it is built ad hoc for a purpose.

In general, most deep learning algorithms involve optimization if some sort. Optimization referes to the task of either minimizing or maximizing a function $f(x)$ by altering x. A more common practice is to phrase the problem in terms of minimizing $f(x)$. Maximization is equivalent to minimization processes if we consider the function $-f(x)$.

A selection of possible techniques that can be used for optimization is presented in section 2.5. In the space below some of the most common loss functions are given along with the terms of use.

- ▷ **Mean Squared Error (MSE or L2 Loss)**:
  As the name suggests, Mean Square Error is measured as the average of squared difference between predictions y_i and actual observations $\hat{y}_i$.

$$L_{\text{MSE}} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.9}$$

  It is only concerned with the average magnitude of error irrespective of their direction. However, due to squaring, predictions which are far away from actual values are penalized heavily in comparison to less deviated predictions. Plus MSE has nice mathematical properties which makes it easier to calculate gradients.

- ▷ **Mean Absolute Error (MAE or L1 Loss)**:
  Mean absolute error is measured as the average of sum of absolute differences between predictions y_i and actual observations $\hat{y}_i$.

$$L_{\text{MAE}} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|^2 \tag{2.10}$$

  Like MSE, this as well measures the magnitude of error without considering their direction. Unlike MSE, MAE needs more complicated tools such as linear programming to compute the gradients. Plus MAE is more robust to outliers since it does not make use of square.

▷ **Cross Entropy Loss (Negative Log Likelihood)**:
This is the most common setting for classification problems. Cross-entropy loss increases as the predicted probability for a class out of M possibilities diverges from the actual label.

$$H(y_o, p_o) = -\sum_{i=1}^{M} y_{o,i} \log p_{o,i} \tag{2.11}$$

Terminology:
M = Number of classes
y = Binary indicator (0 or 1) if class label i is the correct classification for observation o
p = predicted probability observation o is of class i
So, when the label $y_{o,i}$ is equal to 1, the other terms of the sum disappear. An important aspect of this is that Cross Entropy Loss penalizes heavily the predictions that are confident but wrong.

▷ **Kullback-Leibler Divergence (KL)**:
In mathematical statistics, the Kullback–Leibler Divergence (also called relative entropy) is a measure of how one probability distribution is different from a second, reference probability distribution. translated in terms of our optimization problem:

$$D_{\mathrm{KL}}(p||q) = -\left(\sum_{i=1}^{M} y_{o,i} \log p_{o,i} - \sum_{i=1}^{M} y_{o,i} \log y_{o,i}\right) \tag{2.12}$$

2.4 Backpropagation algorithm

When we use a feedforward neural network to accept an input x and produce an output $\hat{y}$, information flows forward through the network. The input x provides the initial information that then propagates up to the hidden units at each layer and finally produces $\hat{y}$. This is called forward propagation.

During training, forward propagation can continue onward until it produces a scalar loss $L(\mathbf{w}, \vec{b}$. The backpropagation algorithm, often simply called backprop, allows the information from the loss to then flow backward through the network in order to compute the gradient of the loss respect to the free parameters $\mathbf{w}$ and $\vec{b}$. Numerically evaluating such an expression can be computationally expensive. The backpropagation algorithm does so using a simple and inexpensive procedure.

The fundamental equations of the algorithm are presented below.

$$\delta^L = \nabla_a C \odot \sigma'(z^L) \tag{2.13}$$

$$\delta^l = [(\mathbf{w}^{l+1})^T \delta^{l+1}] \odot \sigma'(z^L) \tag{2.14}$$

$$\frac{\partial C}{\partial b_i^l} = \delta_i^l \tag{2.15}$$

$$\frac{\partial C}{\partial \mathbf{w}_{ik}^l} = a_k^{l-1} \delta_i^l \tag{2.16}$$

Translated in form of pseudo-code, the backpropagation algorithm appears like the one in the space below. It has to be specified that this algorithm is adapted for a fully connected network. In the case of a not-fully connected network the code becomes a bit more complex, but the concept behind is the same.

Algorithm 1 Forward propagation through a typical deep neural network and the computation of the cost function. The loss $L(\hat{y}, y)$ depends on the output $\hat{y}$ and on the target y. The symbol θ will be used to indicate both weights and biases.

Require: Network depth, l
Require: $\mathbf{w}^{(i)}$, $i \in \{1, \ldots, l\}$, the weight matrices of the model
Require: $b^{(i)}$, $i \in \{1, \ldots, l\}$, the bias parameters of the model
Require: x, the input to the process
Require: y, the target output

procedure FORWARD PROPAGATION(x)
  $h^{(0)} = x$
  for $k = 1, \ldots, l$ **do**
    $a^{(k)} = b^{(k)} + \mathbf{w}^{(k)} h^{(k)}$
    $h^{(k)} = f(a^{(k)})$
  end for
  $\hat{y} = h^{(l)}$
  return $\hat{y}$, $L(\hat{y}, y)$
end procedure

Algorithm 2 Backward computation for the deep neural network of Algorithm 1, which uses, in addition to the input x, a target y. This computation yields the gradients on the activations $a^{(k)}$ for each layer k, starting from the output layer and going backwards to the first hidden layer. From these gradients, which can be interpreted as an indication of how each layer's output should change to reduce error, one can obtain the gradient on the parameters of each layer.

procedure BACKWARD COMPUTATION(x) ▷ After the forward computation
  $g \leftarrow \nabla_{\hat{y}} L = \nabla_{\hat{y}} L(\hat{y}, y)$ ▷ Compute the gradient on the output layer
  for $k = l, l-1, \ldots, 1$ **do**
    $g \leftarrow \nabla_{a^{(k)}} L = g \odot f'(a^{(k)})$
    $\nabla_{b^{(k)}} L = g$
    $\nabla_{\mathbf{w}^{(k)}} L = g h^{(k-1)\ T}$
    $g \leftarrow \nabla_{h^{(k-1)}} L = \mathbf{w}^{(k)\ T} g$
  end for
  return $\nabla_{b^{(k)}} L$, $\nabla_{\mathbf{w}^{(k)}} L$, $k \in \{1, \ldots, l\}$
end procedure

2.5 Updating free parameters: Optimizers

Now that we have a solid method to compute the derivatives respect to free parameters of the loss function L, we can update weights and biases according to the derivatives. In fact, those quantities can be seen as a measure of how much the parameters are to be changed in order to get results that resemble the output. The basic idea is to change $\mathbf{w}$ and $\vec{b}$ of a quantity equals to the corrisponding derivative multiplied by $-\eta$, where η is another hyperparameter called learning rate. It is in general a small number that represents the speed of the learning process. It is fixed at the beginning of the procedure for some algorithms, for others it is variable during the training process, starting from a predefined value.

We focus now on the algorithms that execute the updaing phase of the parameters. They are called optimizers, just like the title of the section suggests and also for them there is a wide variety of choices. Some of them are still under study because their generalization power has not be proven completely. This is the case of "adam" algorithm. We'll start to explain one the simplest, the Stochastic Gradient Descent (SGD), and then we'll introduce the so called fast optimizers. training a very large deep neural network can be painfully slow, but this kind of optimizers, along with some other techniques, gives a huge boost to the computation.

2.5.1 An example of algorithm with fixed learning rate

Algorithm 3 Stochastic Gradient Descent (SGD) update at training iteration k

Require: Learning rate η_k
Require: Initial parameter θ
procedure SGD($\{x^{(i)}\}_{i=1,\dots,n}$) ▷ Take as input a dataset of n elements
  while stopping criterion not met **do**
    Sample a minibatch $\{x^{(1)}, \dots, x^{(m)}\}$ from the training set with corresponding targets $y^{(i)}$.
    $\hat{g} \leftarrow +\frac{1}{m}\nabla_\theta \sum_i L(f(x^{(i)};\theta), y^{(i)})$ ▷ Compute gradient estimate
    $\theta \leftarrow \theta - \eta\hat{g}$
  end while
  return θ
end procedure

2.5.2 An example of algorithm with adaptive learning rates

Algorithm 4 Adam algorithm

Require: Step size ϵ (suggested default: 0.001)
Require: Exponential decay rates for moment estimates, ρ_1,$\rho_2 \in [0, 1)$ (suggested defaults: 0.9 and 0.999 respectively)
Require: Small constant δ, usually 10^{-8}, used to stabilize division by small numbers
Require: Initial parameter θ
Require: Minibatch dimension m
procedure ADAM($\{x^{(i)}\}_{i=1,\dots,n}$) ▷ Take as input a dataset of n elements
  Initialize 1st and 2nd moment variables, $s = 0$, $r = 0$
  Initialize time step $t = 0$
  while stopping criterion not met **do**
    Sample a minibatch $\{x^{(1)}, \dots, x^{(m)}\}$ from the training set with corresponding targets $y^{(i)}$.
    $\hat{g} \leftarrow +\frac{1}{m}\nabla_\theta \sum_i L(f(x^{(i)};\theta), y^{(i)})$ ▷ Compute gradient estimate
    $t \leftarrow t + 1$
    $s \leftarrow \rho_1 s + (1 - \rho_1)g$ ▷ Update biased 1st moment estimate
    $s \leftarrow \rho_2 r + (1 - \rho_2)g \odot g$ ▷ Update biased 2nd moment estimate
    $\hat{s} \leftarrow \frac{s}{1-\rho_1^t}$ ▷ Correct bias in 1st moment
    $\hat{r} \leftarrow \frac{r}{1-\rho_2^t}$ ▷ Correct bias in 2nd moment
    $\Delta\theta = -\epsilon\frac{\hat{s}}{\sqrt{\hat{r}}+\delta}$ ▷ Compute parameter update
    $\theta \leftarrow \theta + \Delta\theta$
  end while
  return θ
end procedure

2.6 Advanced tecniques

It is possible to improve performances of a Neural Network through some other techniques. In particular, those techniques become necessary when the model we want the net to learn is pretty much complex and so the architecture of the net too.

2.6.1 Regularizers

Regularization is one of the central concerns of the field of machine learning, rivaled in its importance only by optimization. Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error. In practice, the most common

way to regularize a model that learns a function $f(x;\theta)$ is adding a penalty called regularizer to the cost function. An example employed when weight decay is applied to the parametes is the following one:

$$\Omega(\mathbf{w}) = \mathbf{w}^{\mathrm{T}}\mathbf{w} \tag{2.17}$$

2.6.2 Dropout

The most popular regularizatioon technique for deep neural networks is arguably dropout. It was proposed by [ADD REFERENCE] in 2012 and further detailed in [ADD OTHER REFERENCE]; it has proven tobe successful.

It is a very simple algorithm: at every training step, every neuron (including the input neurons but excluding the output neurons) has a probability p of being temporarily "dropped out", meaning it will be entirely ingnored during this training step, but it may be active during the next step. The hyperparameter p is called the dropout rate and it is typically set to 50%. After training, neurons don't get dropped anymore.

A possible explanation of the effectiveness of dropout could be given. Neurons trained with dropout can't co-adapt with their neughbouring neuron. They have to be useful as possible on their own. They can't rely excessively on just a few input neurons and so they must pay attention to each of their input neurons. They end up being less sensitive to slight changes in the inputs. In the end, we get a more robust network that generalizes better.

2.6.3 Parameter initialization strategies

If we want to get better results from training procedure and to decrease training time, it is crucial to adopt a good initialization strategy of the free parameters. If we start with a set of parameters quite near to the set that minimizes the loss function, the training algorithm will rapidly find the path towards the local/global minimum of the loss function.

Several initialization strategies have been developed in the research field of Deep Learning. In the space below some of them are presented:

- **Xavier and He initialization**.
  The connection weights must be initialized randomly ad described in Table 2.1, where $n = n_{\text{inputs}}$ and $m = n_{\text{outputs}}$ are respectively the number of input and output connections for the layer whose weights are being initialized (also called fan-in and fan-out). The distribution used to initialization has to be chosen between a normal distribution with mean 0 and standard deviation σ and a uniform distribution between $-r$ and r. He initialization is the name given to this kind of initialization with ReLU activations.

Activation function	Uniform distribution $[-r, r]$	Normal distribution $(0, \sigma)$
Logistic	$r = \sqrt{\dfrac{6}{n+m}}$	$\sigma = \sqrt{\dfrac{2}{n+m}}$
Hyperbolic tangent	$r = 4\sqrt{\dfrac{6}{n+m}}$	$\sigma = 4\sqrt{\dfrac{2}{n+m}}$
ReLU	$r = \sqrt{2}\sqrt{\dfrac{6}{n+m}}$	$\sigma = \sqrt{2}\sqrt{\dfrac{2}{n+m}}$

Table 2.1: Xavier and He initializations.

Chapter 3

CERN research for exotic particles through LHC collider

Searching for exotic particles never observed before implies a need for an enormous amount of energy in the processes involved in their creation. In fact, if we want to study something never obseved before at the scales of energy normally reached, it is highly unlikely we will find something new and interesting. This is the reason for which the Large Hadron Collider, namely LHC, was designed and built.

LHC is actually the world's largest and most powerful particle accelerator. Its project was conceived in 1976 when the European particle physics community began to discuss building a Large Electron Positron (LEP) collider at CERN, and it was realized between 1998 and 2008. It required a collaboration of over 10000 scientists and hundreds of universities and laboratories, as well as more than 100 countries. It first started up on 10 September 2008, and remains the latest addition to CERN's accelerator complex.

3.1 An insight into LHC structure

The LHC consists of a ring 27-kilometers long of superconducting magnets with a number of accelerating structures to boost the energy of the particles along the way. Inside the accelerator, two high-energy particle beams travel at relativistic speed, close to the speed of light c. Before they are made to collide, they both are accelerated to a energy of the order of TeV. The two beams travel in opposite directions in separate beam pipes, which are fundamentally two tubes kept at ultrahigh vacuum. They are guided around the accelerator ring by a strong magnetic field mainteined by superconducting electromagnets, built from coils of special electric cable that operates in a superconductive state. It is of primary importance to conduct electricity with the lowest resistence, therefore loss of energy, possibile. Such technique requires cooling the magnets to a temperature close to 0 K, which is even colder than the outer space. In order to get those temperatures, much of the accelerator is connected to a distribution system of liquid helium and the cooling phase takes a long time.

Thousands of magnets of different varieties and sizes are exploited to direct the beams around the accelerator. In total, there are:

- 1232 dipole magnets whose length is 15 metres and their purpose is bending the beams.
- 392 quadrupole magnets, each 5-7 metres long, which focus the beams, thanks to their properties of magnetic lens.

Just prior to collision, another type of magnet is used to "squeeze" the particles closer together to increase the chances of collisions. In fact, the particles can be imagined as very tiny objects, so it is clear that the task of making them collide must be thought with such precision.

The beams inside LHC are made to collide at four locations around the accelerator ring, corrisponding to the positions of four particle detectors: CMS, ATLAS, ALICE and LHCb. All the controls for the accelerator, its services and technical infrastructure are housed at the CERN Control Centre.

3.2 CMS experiment

The acronym CMS means Compact Muon Solenoid and it is a general-purpose detector amongst one of the four mentioned before. It has a broad physics programme ranging from studying the Standard Model, namely SM, to searching for extra dimensions and particles that could make up dark matter, whose existence belongs to physics beyond the SM. CMS also stands for one of the largest international scientific collaborations in history, involving 4300 particle physicists, engineers, technicians, students and support staff from 182 institutes in 42 countries (by February 2014).

It is built around a huge solenoid magnet. This takes the form of a cylindrical coil of superconducting cable that generates a field of 4 Tesla. The importance of this value can be better understood thinking that it is 100000 times the magnetic field of Earth. The field is confined by a steel "yoke" that forms the bulk of the detector, whose weight is about 14000 tons. It was built in 15 sections, which were reassembled in situ. The detector gets its name from the fact that:

- It is 15 metres high and 21 metres long, so it is quite compact for all the detector material it contains.
- It is designed to detect particles known as muons (and indicated with μ) very accurately.
- It has the most powerful solenoid magnet ever made.

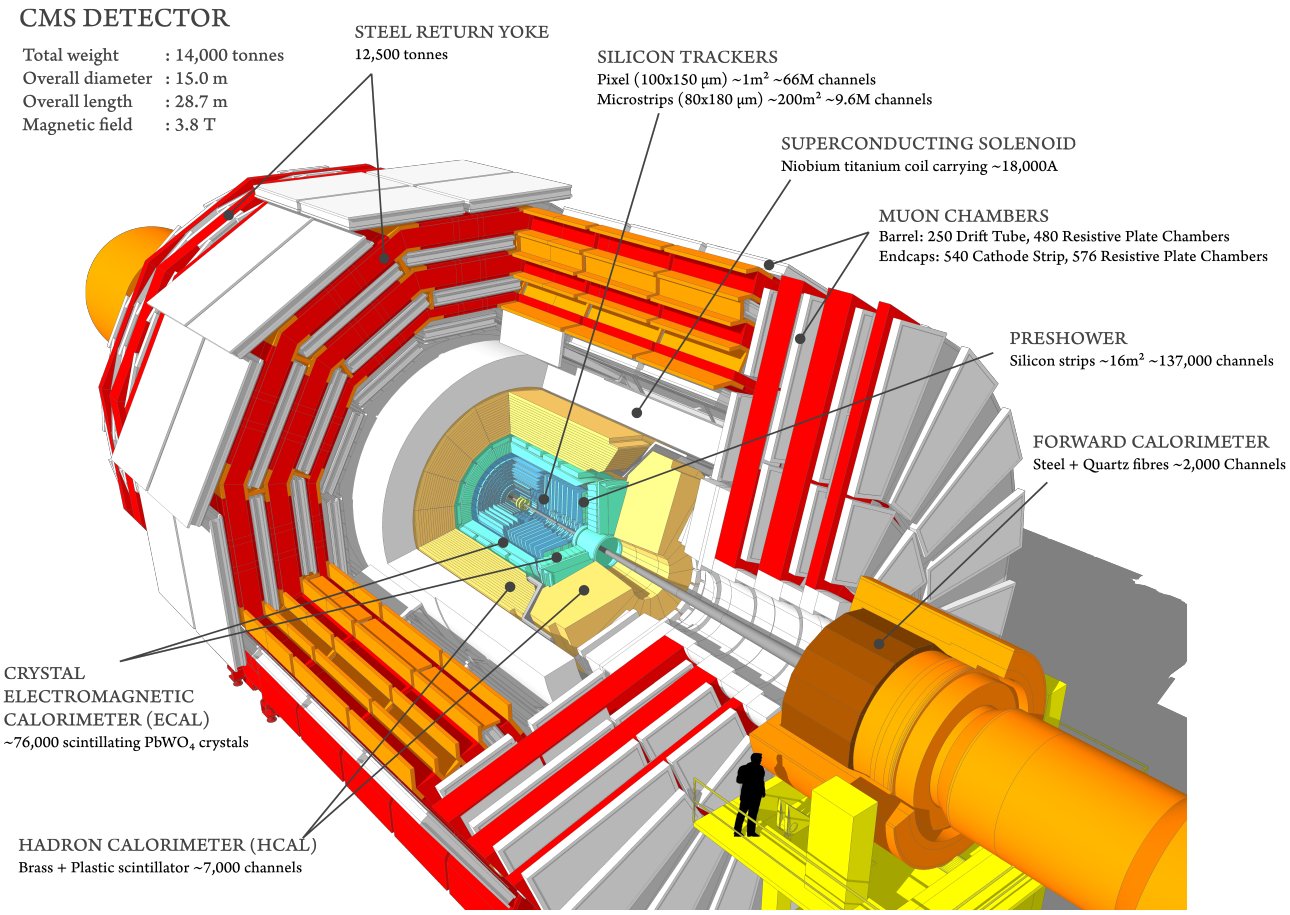


Figure 3.1: CMS detector section.

The detector is shaped like a cylindrical onion, with several concentric layers of components. These components help prepare "photographs" of each collision event by determining the properties of the particles produced in that particular collision. This is done by:

- **Bending particles:**
  A powerful magnet is needed to bend charged particles as they fly outwards from the collision

point. This process is done in order to identify the charge and to measure the momentum of the particle. In fact, positive and negative charged particles bend in opposite directions in the same magnetic field and, in particular, high momentum particles bend less compared with low-momentum ones.

The solenoid magnet is formed by a cylindrical coil of superconductiong fibres. A current up to 18500 A flows within these coils and it encounters no resistence for the phenomenon of superconductivity. The magnetic field of 4 T must be confined to the volume of the detector and is done by the steel "yoke" that forms the bulk of the detector's mass.

- **Identifying tracks:**
  It is fundamental to identify the paths taken by the bent charged particles and it must be done with extreme precision. This is done by a silicon tracker made of around 75 million individual electronic sensors arranged in concentric layers. A charged particle that passes through these layers interacts electromagnetically with the silicon and produces a signal. The information carried by individual signals can be joined together in order to identify the track of the traversing particle.

- **Measuring energy:**
  The energy of the various particles produced in each collision is a crucial information in the understanding of what happened in the collision process. This information is collected by calorimeters, which are particle detectors made to measure the energy. Inside CMS there are two kind of these. The Electromagnetic Calorimeter (ECAL) is the inner layer of the two and measures the energy of electrons and photons by stopping them completely. The other particles, i.e. hadrons, fly through the ECAL and are stopped in the outer layer by the Hadron Calorimeter (HCAL).

- **Detecting muons:**
  The final particles that CMS observe directly are the muons. They are not stopped by any calorimeter, so there are other special detectors interleaved with the return yoke of the solenoid. It is possible to measure each muon's momentum both inside the superconducting coil (by the tracking devices) and outside of it (by the muon chambers).

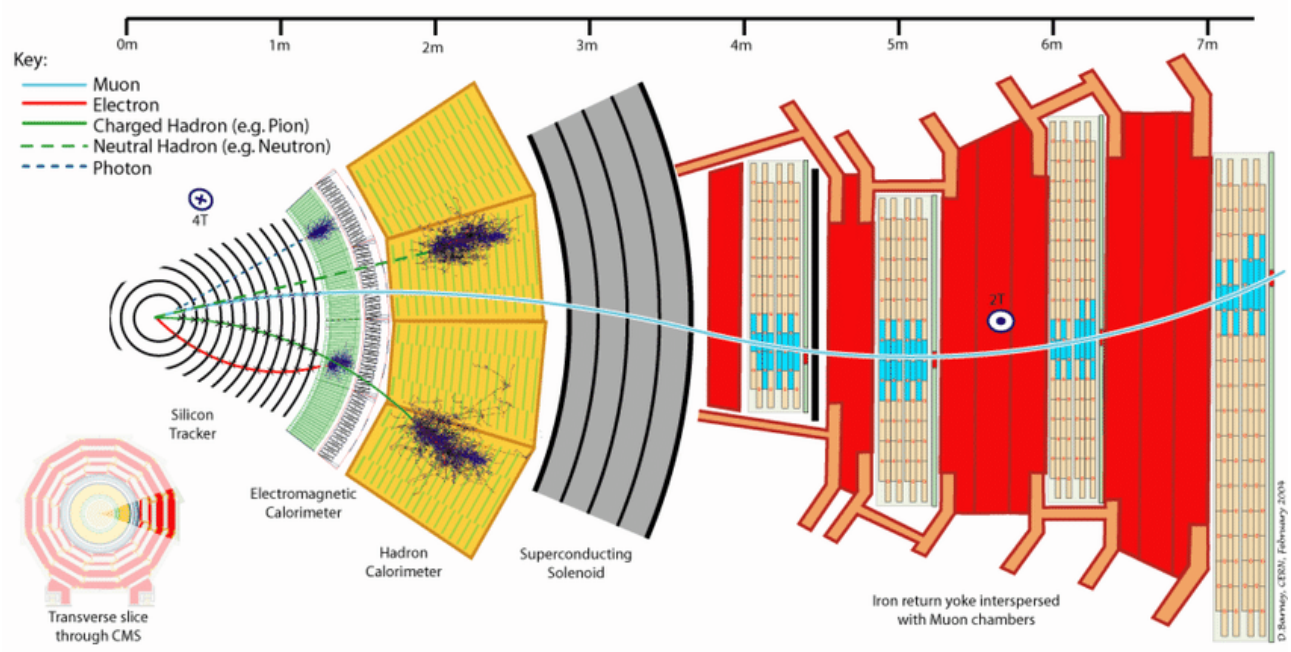


Figure 3.2: CMS detector transverse section.

Thanks to the ensemble of all these components, it is possible to detect the energies of the products of the collision, except the energies carried by neutrinos ν. In fact, these are elementary particles that

can't be detected by this kind of detectors. Each layer of the composition gives an information about the processes happened after the collision. By putting together all of those informations, the intermediate states of the processes could be studied as well as the laws that regulate the whole process.

As stated before, every layer can give informations about an intermediate state. So the particles observed are not the initial products of the process, but they are the traces of pre-existing unstable particles that decays in other products before the detector could observe them. The intermediate unstable states are called resonances and to better understand them it is appropriate to describe the classical definition.

3.3 From data to discoveries: resonance formation study

Resonant phenomena are ubiquitous in physics at both the macroscopic and microscopic levels. Resonances have an extremely important role in hadron spectroscopy.

In Classical Physics, a resonance is a phenomenon that occurs when the frequency at which a periodical force is applied is equal or nearly equal to one of the natural frequencies of the system on which it acts. The visible effect of the resonance is that the system oscillates to a larger amplitude respect to the one when a force is applied with a different frequency.

Let's analyze this phenomenon in the case of an oscillating system with an external force that drives the oscillation. The second order differential equation that describes the system is the following one:

$$\frac{\mathrm{d}^2x}{\mathrm{d}\theta^2} + 2\gamma\omega_0\frac{\mathrm{d}x}{\mathrm{d}\theta} + \omega_0^2 x = \frac{F_0}{m}\sin\Omega t = \Gamma\sin\Omega t \tag{3.1}$$

where F_0 is the amplitude of the driving force, Ω is its angular frequency, ω_0 is the undamped angular frequency and γ is the damping ratio. The general solution of this ODE is a combination of sinusoidal function exponentially damped and another sinusoidal function that describes a stationary state:

$$x(t) = Ae^{-\gamma\omega_0}\sin\left(2\gamma\omega_0 + \phi\right) + B\sin\left(\Omega t - \delta\right) \tag{3.2}$$

where:

$$B = \frac{\Gamma}{\sqrt{(\omega_0^2 - \Omega^2)^2 + 4\gamma^2\omega_0^2\Omega^2}} \tag{3.3}$$

$$\delta = \arctan\left(\frac{2\gamma\omega_0\Omega}{\omega_0^2 - \Omega^2}\right) \tag{3.4}$$

The relation for the amplitude of the solution in the stationary state shows there is a maximum value for B, which is reached for a particular value of the angular frequency of the drive. It means that the system becomes resonant for a certain value of Ω and the transfert of energy from the drive to the system is maximized. The plots for different values of the parameters ω_0 and γ are presented below along with the corresponding plot for the phase shift δ.

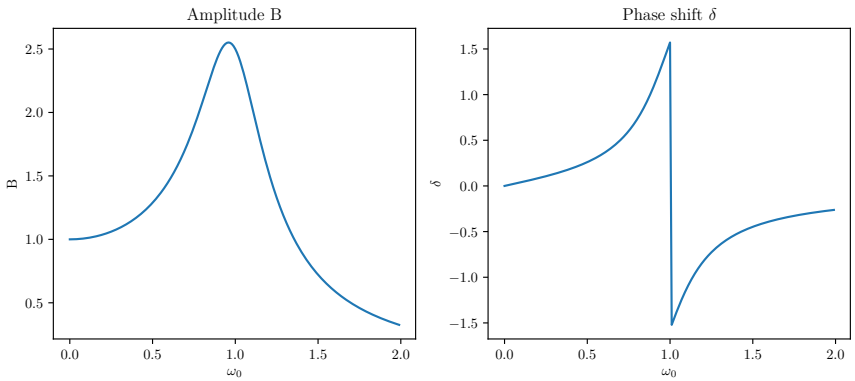


Figure 3.3: Resonance plot.

In subnuclear physics, the extremely unstable hadrons can be observed as 'resonances' in two different ways. The first approach is searching for very unstable particles by measuring as a function of energy the cross-section of processes of the following type:

$$a + b \to c + d + \dots \tag{3.5}$$

In a quantum scattering process the relevant functions of energy are the scattering amplitude and the cross section, which is the measured observable. Near a resonance, the distribution of σ will remark the shape of the resonant peak in Figure 3.3.

The second approach, corresponding to a specific class of experiments, is based on resonance production. To better explain the method, let's suppose we are studying the following process:

$$a + b \to c + d + e \tag{3.6}$$

where we are searching for a particle decaying into the stable or metastable particles c, d. In this example, this particle will called r, which obviously stands for resonance. If r exists, the decay process could develop into more channels:

$$a + b \to r + e \to c + d + e \tag{3.7}$$

In these cases, the mass of $c+d$, called M_{cd}, is expected to be equal to the mass of r, called M_r, and it follows a Breit-Wigner distribution peaked at M_r and with width Γ. If the decay goes directly in the final state (this is the non-resonant process), M_{cd} can have any value in the range imposed by physical constraints of energy and momentum conservation, as it is a three-body decay process, and there is no peak in the distribution. Now, we have to measure the energies and the moment for each event by computing:

$$M_{cd} = \sqrt{(\mathcal{E}_c + \mathcal{E}_d)^2 - (\vec{p}_c + \vec{p}_d)^2} \tag{3.8}$$

The resonance appears as a peak on a smooth background in the M_{cd} distribution.

Chapter 4

$Z \to \mu^+\mu^-$ decaying process

The process studied in this thesis work is the decay of a Z boson, generated by proton-proton collisions, into a pair of muon and antimuon. The scheme of the dacay is presented below:

$$pp \to Z \to \mu^+\mu^- \tag{4.1}$$

This is the prevision in the hypothesis of SM, where the Z boson is commonly represented by the symbol Z^0. The presence of a new physics scenario is modeled by the existence of a new boson, which is commonly indicated in literature by Z'. In this case, the scheme of the dacay becomes:

$$pp \to Z' \to \mu^+\mu^- \tag{4.2}$$

The new physics process can by studied by searching for resonant phenomena, for example bumps in the distributions of a set of observables in a zone where only background is expected to be present. However, this is not the only case possible in the wide variety of new physics scenarios. In fact, the existence of a new particle could not lead to a resonant peak in the distribution of the high level features, but to a different shape in the same distributions. For example, there could be a different slope in the exponential falling distribution in the SM hypothesis.

Neural Networks can be employed in both cases, maybe with different performances. The purpose of this thesis work will focus on the new physics scenario of resonant phenomena. The study of the other cases is left to future work.

In the following discussion the Z^0 boson will be generally described along with its properties and the features analyzed for the purpose of this work will be presented.

4.1 The Z^0 boson

The Z boson is an elementary particle which is the carrier of the weak force, along with the W boson. The difference with the ladder is that while the W boson is charged, the Z boson is neutral. It was discovered in 1983 at CERN at the Super Proton SSynchrotron.

Since Z is neutral, the sum of the charges of its decay products must be 0 for the conservation of charge. So it goes without saying that Z must decay into a pair of a particle and its antiparticle. There are several possibilities for the decay process:

- In 10% of Z decays charged lepton-antilepton pairs are produced. Therefore there are three sub-cases for the pairs:
  - ▷ electron-positron;
  - ▷ muon-antimuon;
  - ▷ tau-antitau;
- In 20% of cases it decays into a neutrino-antineutrino pair. However, they are invisible to the detector since they don't interact with anything, in fact they have no charge. A way to state their presence is to look for some energy or trnsverse momentum missing after the collision. There are three possibilities for neutrino decay as well as for the pair of leptons.

- In 70% of cases the decay gives a quark-antiquark pair. These appear as particle showers called "jets" in the detector. Concerning the possibilities for this mode of decay, quarks have a property called "colour" and there are six different types of quark (up, down, charm, strange, top, bottom). Therefore there are 18 possibilities for the quark-antiquark pair decay.

The set of possible cases counts 24 possibilities, 21 of which are visible. In this work only one will be considered, i.e. the muon-antimuon decay.

4.2 High Level Features for the analysis of the decay

The raw data taken by the detector after the collision is a set of low-level features, namely LLFs, such as the muon and antimuon momenta. Since we are trying to understand the process in three dimensions with one of the axis aligned with the y component of the first muon, the LLFs of interest are $p_{1,x}$, $p_{1,z}$, $p_{2,x}$, $p_{2,y}$, $p_{2,z}$ To better study the decaying process, it is preferred to combine the low level features into the high level ones, namely HLFs.

Symbol	HLF name
$p_{T,1}$	Transverse momentum 1
$p_{T,2}$	Transverse momentum 2
η_1	Pseudorapidity 1
η_2	Pseudorapidity 2
$\Delta\phi$	Difference of azimuthal angles

Table 4.1: High Level Features legend.

The relations employed to obtain the values for HLFs are presented in the space below. The intermediate steps to get the results are presented as well for completeness. Note that although the polar angle θ is a HLF too, the pseudorapidiy is preferred to it in high energy physics.

$$p_{T,1} = p_{1,x} \tag{4.3}$$

$$p_{T,2} = \sqrt{p_{2,x}^2 + p_{2,y}^2} \tag{4.4}$$

$$\eta_1 = \operatorname{arctanh} \frac{p_{1,z}}{p_1} \tag{4.5}$$

$$\eta_2 = \operatorname{arctanh} \frac{p_{2,z}}{p_2} \tag{4.6}$$

$$\Delta\phi = \phi_2 - \phi_1 \tag{4.7}$$

where:

$$p_1 = \sqrt{p_{1,x}^2 + p_{1,z}^2} \tag{4.8}$$

$$p_2 = \sqrt{p_{2,x}^2 + p_{2,y}^2 + p_{2,z}^2} \tag{4.9}$$

$$\phi_1 = \arccos \frac{p_{1,x}}{p_{T,1}} \tag{4.10}$$

$$\phi_2 = \operatorname{sign} p_{1,y} \arccos \frac{p_{2,x}}{p_{T,2}} + \pi(1 - \operatorname{sign} p_{1,y}) \tag{4.11}$$

4.3 The dataset analyzed

The dataset employed in this thesis work is a set of simulated data. The events are generated with the aid of the framework MadGraph5 assuming collisions of protons at 8 TeV. The detector response is taken into account by Delphes, as well as showering and hadronization by Pythia.

The dataset parametrizing the reference distribution was simulated according to SM predictions. The dataset with signal events are simulated assuming there is a resonant new physics phenomenon, visible in a new peak in the invariant mass distribution.

Histograms of the five momenta and of the HLFs are presented in the following plots. Only-reference, only-signal and signal+background data combinations are showed in different plots, in order to see more clearly the differences.

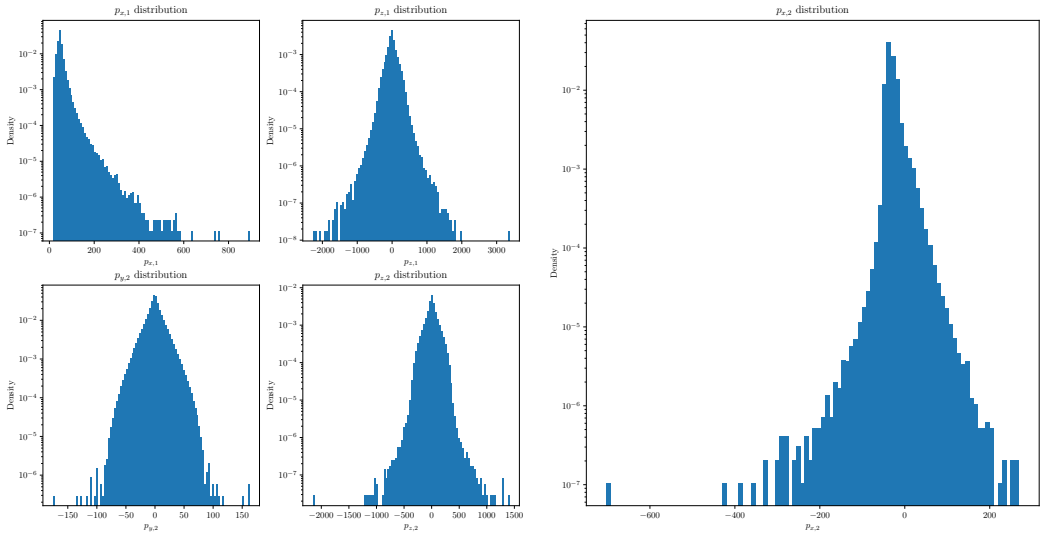


Figure 4.1: Data distribution.

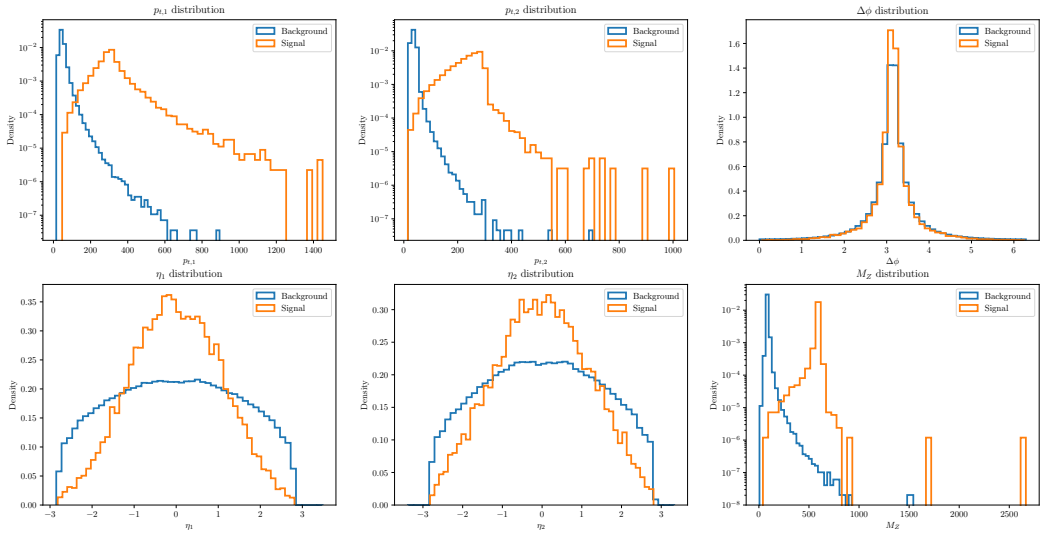


Figure 4.2: Features and invariant mass distributions.

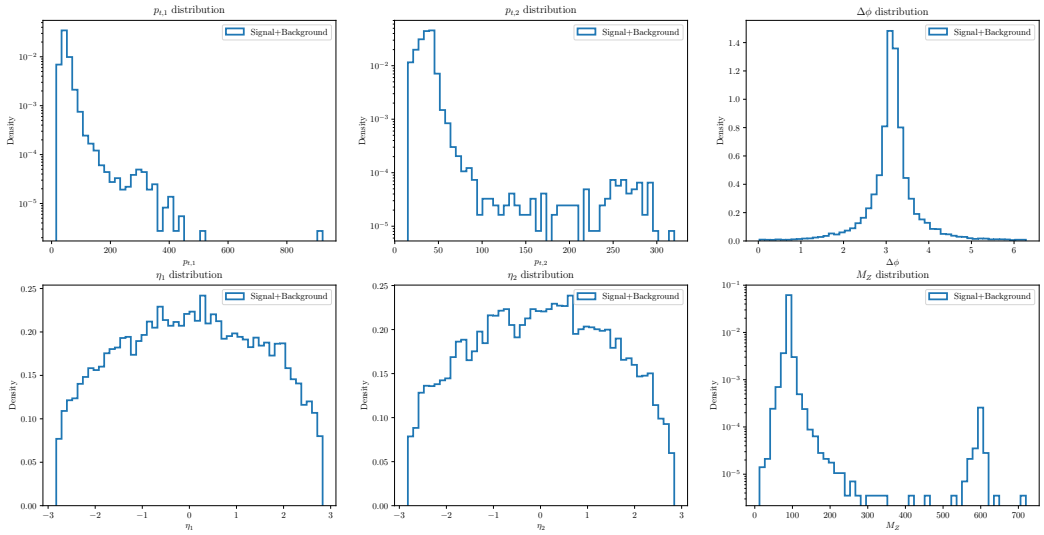


Figure 4.3: Features and invariant mass distributions with signal+background in the same dataset.

Chapter 5

Development of the algorithm for NP research

In this chapter the foundations of the algorithm employed for data analysis will be carefully explained step by step. The good practice is to begin from the statistical fundation of the algorithm itself as it set on not trivial concepts. In the next step a network for the purpose shall be built beginning from its architecture and then choosing opportune activation functions, optimizer and weight initialization. Moreover, several crucial aspects that may cause difficulties in the work progression will be stated and faced in order to find the best solution or at least a compromise if there isn't. Some of those problems to be faced are:

- The search for the degrees of freedom of the network employed.
- The look-elsewhere effect.
- The limitation of the phase-space of the free parameters of the net, which translates in the clipping of the weights. This point is strictly related to the previous one.

5.1 Statistical foundations

The entire construction of the statistical foundations of the procedure are accurately described in [14]. Here we recap the main steps to the construction of a hypotesis test and so of a variable with discriminating power.

5.1.1 Construction of a test statistic

The starting point is a dataset of repeated measurement of a d-dimensional random variable x, distributed as the reference distribution $n(x|R)$. To compare the reference distribution with observed data an alternative hypostesis is needed, which means another distribution $n(x|w)$. It is convenient to parametrize $n(x|w)$ in terms of $n(x|R)$. Moreover, we know that $n(x|w)$ is strictly positive and because the log-likelihood ratio will be used later, we will express it as:

$$n(x|w) = n(x|R)e^{f(x;w)} \tag{5.1}$$

where $f(x)$ is a function from a set $\mathcal{F} = \{f(x;w), \forall w\}$.

Now that the set of alternative hypotesis is built in a parametrized from, the optimal statistical test for the reference model is defined by the Neyman-Pearson construction, which is based on the maximum likelihood principle. The idea is to compare $n(x|R)$ with the best fit distribution $n(x|\hat{w})$, where $\hat{w}$ is the one that maximizes the likelihood. Hence the test statistic is:

$$t(\mathcal{D}) = 2\log\left[\frac{e^{-N(\hat{w})}}{e^{-N(R)}}\prod_{x\in\mathcal{D}}\frac{n(x|\hat{w})}{n(x|R)}\right] = -2\min_{\{w\}}\left[N(w) - N(R) - \sum_{x\in\mathcal{D}} f(x;w)\right] \tag{5.2}$$

where $N(R)$ is the expected number of events in the reference model and $N(w)$ is the expected number of events in the alternative hypotesis. $N(w)$ can be computed by:

$$N(w) = \int n(x|w)\mathrm{d}x = \int n(x|R)e^{f(x;w)}\mathrm{d}x \tag{5.3}$$

The test statistic needs a consistent definition of a p-value. In order to compute it, we have to associate a probability to the value of t, namely t_{obs}, obtained using the observed dataset. For this reason we have also to find the PDF of t in the reference hypotesis and we can do it by evaluating t a sufficient number of times on a large sample of toy datasets. By this way, we can obtain the observed p-value:

$$p_{obs} = \int_{t_{obs}}^{\infty} P(t|R)\mathrm{d}t \tag{5.4}$$

The key idea of [14], employed for the present work, is to parametrize the alternative hypotesis with neural networks for their properties of universal unbiased approximants.

5.1.2 Ideal test statistic

The results of the method that exploits the power of Neural Networks must be discussed in comparison with something we already know. It means that assuming we have complete knowledge of the distribution describing the observed data, an ideal test statistic could be defined and it will be the previous cited benchmark.

The alternate hypotesis with the assumption of complete knowledge will be denoted by "NP" and it has no free parameters. In this case the test statistic becomes:

$$t_{id}(\mathcal{D}) = 2\log\left[\frac{e^{-N(NP)}}{e^{-N(R)}}\prod_{x\in\mathcal{D}}\frac{n(x|NP)}{n(x|R)}\right] \tag{5.5}$$

According to the Neyman-Pearson lemma, t_{id} is the optimal disctiminant between the reference and the new physics hypotheses. It produces the smallest median p-value if NP is the true distribution of the data sample.

This test statistic is denoted as "ideal" because it is the most suited one to discover data departures from the reference model. However, it is not useful but as a benchmark because we can use it only when the true data distribution is known a priori. In our case we are testing the efficiency of a ML-approach and therefore the true distribution is known.

5.1.3 Adaptation of $t(\mathcal{D})$ as a loss function

The minimization in Equation 5.2 must be done numerically through the tools for neural network training discussed in the first chapter. Before doing this, we have to express Equation 5.2 as a loss function.

The first thing to do is to estimate $N(w)$. This task can be done through Monte Carlo methods:

$$N(w) = \frac{N(R)}{N_{\mathcal{R}}}\sum_{x\in\mathcal{R}} e^{f(x;w)} \tag{5.6}$$

Thus Equation 5.2 becomes:

$$t(\mathcal{D}) = -2\min_{\{w\}}\left[\frac{N(R)}{N_{\mathcal{R}}}\sum_{x\in\mathcal{R}}(e^{f(x;w)}-1) - \sum_{x\in\mathcal{D}} f(x;w)\right] \equiv -2\min_{\{w\}}\left[f(\cdot,w)\right] \tag{5.7}$$

L has the form of a loss function. It is convenient to write it as a single sum over events by introducing a target variable y such that:

- $x \in \mathcal{R} \Longrightarrow y = 0$;

- $x \in \mathcal{D} \Longrightarrow y = 1$.

Thus the explicit expression of L is:

$$L[f] = \sum_{(x,y)} \left[(1-y)\frac{N(R)}{N_{\mathcal{R}}}(e^{f(x)} - 1) - yf(x) \right] \tag{5.8}$$

The trained neural network, namely $f(x;\hat{w})$, is simply the maximum likelihood fit to the data and reference distributions log-ratio. It is the best approximant of the true underlying data distribution $n(x|T)$:

$$f(x;\hat{w}) \approx \log \left[\frac{n(x|T)}{n(x|R)} \right] \tag{5.9}$$

5.2 The algorithm

The network architecture choosen for the task has the following characteristics:

- 3 fully-connected layers, whit the first and the second ones with five neurons each and the last one with only one output neuron. Therefore the input is multidimensional as the input are the HLFs for every event.
- The activation functions choosen are sigmoids for the first and second layer, while a linear activation is employed for the last layer.

The loss function employed is reported in the previous section and it is a cross entropy loss function adapted for the task. Its minization is done by ADAM algorithm. The initialization of the free parameters is the normal random initialization. A clip to the weights is applied in the case they go out of prearranged interval. The discussion on the weight clipping is studied deeper in subsequent sections.

The scheme of the algorithm is the following:

- A sufficiently large reference sample is randomly selected from the whole dataset following the reference hypothesis. Then a data sample with background events is randomly selected from the entire dataset and no signal is added. The network is trained with these data and the output of the process is the minimized loss function. From this value, the final t is computed as minus two times the minimized loss function.
- The previous step is repeated a certain number of times in order to sample the distribution of t in the reference hypotesis (as no signal event is added in the data sample yet). The desidered number of t necessary to the following steps must be considered carefully because the procedure is computationally expensive and its complexity grows linearly with the number of events in the reference sample.
- When the distribution of t in the reference hypotesis is sampled, the first step is repeated but adding a certain number of signal events in the data sample. The output of the process, i.e. the minimized loss, is translated into the value of t_{obs}.
- The distribution of t and t_{obs} are used to compute the p-value p, which is the area in the tail.
- If the value of p is sufficiently small, it signals a tension with the reference hypotesis. Hence anomalies are detected and they can be analyzed through the log-ratio $f(x;\hat{w})$.

5.3 Computing resources

Suppose one wants to sample 200 values of t in the reference hypotesis. Moreover, suppose the execution of a single process for a single value of takes about 1 day. Doing this task on a single machine is computationally very expensive. Assuming this machine could run 10 processes at the same time without swapping, it would take 20 days to get the desired results. Considering the entire procedure

has to be repeated for different conditions and for several number of events in the reference sample, it is totally inconceivable to do everything on a single machine.

Another aspect has to been taken into account. Writing the code that executes the algorithms described above could be quite tricky and unavoidable mistakes in the phase of implementation don't agree with the long execution time.

In this section a solution for each of these difficulties encountered is presented.

5.3.1 TensorFlow back-end and Keras API

The programming language chosen for the work is Python 3. Its simplicity and power were crucial in the selection. Moreover, there exist very powerful tools that address machine learning problems which can make the code easier to write and understand.

The API employed for the task is Keras, which is an open source library specialized for automatic learning tasks. It was used as front-end, while TensorFlow was used as back-end. This one is the most common machine learning library up to now and it provides tools not only for deep learning problems, but for every machine learning problem conceivable. For more information, see [1], [2] and [3].

A bunch of code to build the network and to train it is reported below to give an idea of its clarity. In the following example the architecture of the net is (5,5,1) and the code is reported in a simplificated version.

```
import keras

inputs= keras.layers.Input(shape=(5,))
dense = keras.layers.Dense(5, activation='sigmoid')(inputs)
dense = keras.layers.Dense(5, activation='sigmoid')(dense)
output= keras.layers.Dense(1, activation='linear')(dense)

model = Model(inputs=[inputs], outputs=[output])

model.compile(loss=Loss,
              optimizer='adam')

model.fit(x_train,
          y_train,
          epochs=300000,
          batch_size=batch_size,
          verbose=0,
          shuffle=False)
```

5.3.2 LSF and clustering

In order to speed up the sampling of t distribution, the algorithm was parallelized on a cluster of machines. This is to be intended that different processes were executed simultaneously on different machines and not that a single process was executed on more machines.

The machines used for the task are Tier-2, sited in Legnaro and Padua. The processes sent to them are called bjobs and the sending procedure is done by LSF, which stands for Load Sharing Facilities. Every bjob must be sent by a script, launched through bsub command. Before doing this, a non-trivial configuration procedure of the machine through which the jobs were dispatched has to be done. To make this procedure easier, the access to this machine was done through cloudveneto. A virtual machine instance was created and connected to the lsf sender. It was then personalized to run a python script with these functionalities:

- It configures the environment variables of the machines in order to let the necessary packages accessible by the system. This phase was accomplished by the execution of another script accessible on cvmfs (CERN Virtual Machine File System). For more information on this tool are provided in [4].

- It creates the script that sends the job via LSF to the machines. This phase is executed a number of times equal to the number of toy samples, therefore equal to the number of t values we want.

The jobs sent to the machines are placed with their IDs in a queue. For this work, cms-local-queues were used. A list of the possible queues is reported in Table 5.1.

Queue name	Max job number	Max job runtime
local-cms-short	500	1440 min
local-cms-long	300	10080 min

Table 5.1: Available queues.

Since the complexity scales up linearly with the number of events in the reference sample, runtime for jobs with wide reference sample overtake the limit of 1440 minutes of the short queue. Therefore the long queue was a better choice for the task. A list of the machines accessible by this queue and their characteristics are displayed in Table 5.2.

Machine id	Number of CPUs	Max Memory	Max Swap
wp-05-01	2	64554M	8191M
wl-07-01	2	64537M	8191M
wl-07-02	2	64537M	8191M
wl-07-03	2	64537M	8191M
wl-07-04	2	64537M	8191M
wl-07-05	2	64537M	8191M
wl-07-06	2	64537M	8191M
wl-07-07	2	64537M	8191M
wl-07-08	2	64537M	8191M
wl-07-09	2	64537M	8191M
wl-07-10	2	64537M	8191M

Table 5.2: Machines and resources.

5.4 Degrees of freedom of the network

Trying to understand the exact number of the degrees of freedom of a neural network is not a trivial task. This number must reflect the wideness of the space of functions the network can span. A more complex network will obviously be characterized by a larger number of dof.

A possible approach to determine the number of dof starts from the free parameters of the network. If there are n weights parameters and m biases, the number of dof ν can be estimated by:

$$\nu = n + m \tag{5.10}$$

Therefore, given the network architecture, the number of layers, neurons and the activations, we can give a definition adapted to the case we are studying. A scheme of the network employed is given in Figure 5.1, which makes easier to understand how the connections between different layers work and the way data flow through the net.

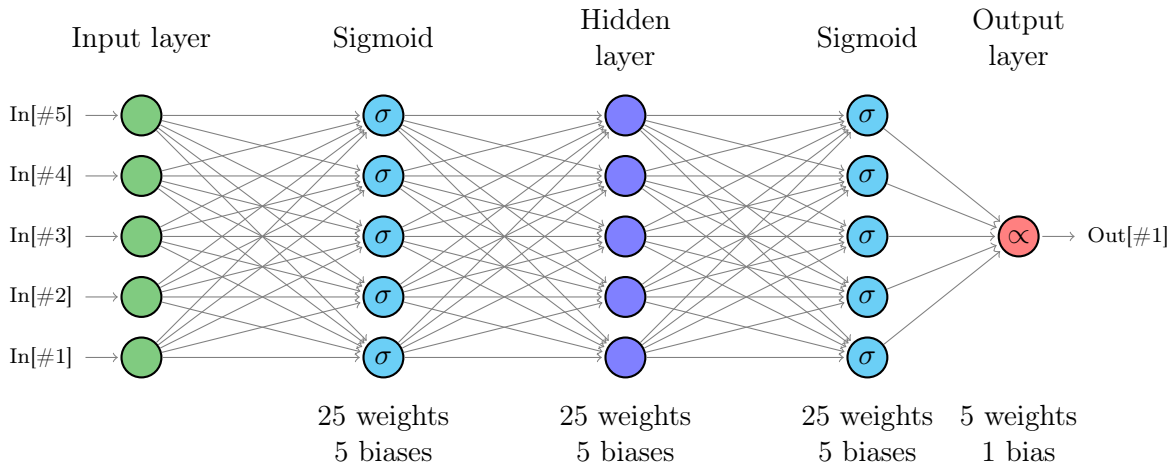


Figure 5.1: Neural Network scheme to understand the number of degrees of freedom.

Every neuron in the middle layers is connected with five neurons of the following layer, including also sigmoid layers. The neurons in the last sigmoid layer are connected with only one neuron, which is the unique one of the output layer. Considering that for every neuron except the ones in the input layer there is a bias parameter, the total number of n, m and ν previously defined is:

$$\begin{matrix} n = 80 \\ m = 16 \end{matrix} \implies \nu = 96 \tag{5.11}$$

Thus the expected dofs of the networks employed is 96.

5.5 The "look-elsewhere" effect

To explain this peculiar effect, let's take the example in [13]. Imagine a common case such as looking for a heavy particle decaying into a pair of hadronic jets. We assume we have complete knowledge of the background model and that we know what kind of bump a new physics signal would produce in the shape of background distribution. Because we don't know exactly were it might appear, we search everywhere in the data.

When data analysis is completed, suppose we find a significant bump at some particular mass value. To claim it is a new signal, the effect must reach or exceed the well known 5σ significance threshold. If this bump is below the 5σ level, but with a consistent significance, let's take 3.5σ, we can't claim a discovery although data seem to prove it. The look-elsewhere effect could have led us to a trap. We looked in many places for a possible signal and found a significant effect somewhere. The likelihood of finding something in the entire region we analyzed is greater than it would be if we had stated beforehand where the signal would be. In fact, we have to take into account the probability boost determined by looking in many places.

The 5σ rule was conceived with exactly this particular effect in mind. This threshold means extremely rare occurrence (just to have an idea, three out of ten million), so even including the look-elsewhere effect, it is still something to take quite seriously.

Employing neural network in the procedure exposed in this work introduces this effect because new physics signals are searched ranging in the entire dataset. This is the main limiting factor of the technique. However, this problem can't be avoided in any attempt to search for new physics model-independently. Thus it is not a limitation of the method itself.

5.6 Weight clipping

The algorithm necessarily requires some sort of regularization because the loss function, i.e. the cross entropy, is unbounded from below. It means that it approces negative infinity if the output of the network f diverges at some value of $x \in \mathcal{D}$. The highly negative values of the minimized loss function

become a high positive value of t. This problematic situation occurs only when the divergence in f is sharply localized, such that $f(x)$ stays finite for all $x \in \mathcal{R}$.

These dangerous configurations are avoided by enforcing an upper bound, namely a "weight clipping" parameter W, on the absolute value of each weight. However, the choice of this parameter is of critical importance and it must be tuned correctly if we want to get reasonable results. In fact, it is known from theoretical arguments that the distribution of t in the reference hypotesis will asympthotically follow a χ^2 distribution with a number of dof equals to the number of free parameters. If we set a too low weight clipping, the sampled distribution will be shifted to the left of the expected distribution. On the contrary, if we set a too high weight clipping, there will be a right shift. An example of the two cases is showed in Figure [DA INSERIRE].

[INSERIRE IMMAGINE]

[INSERIRE CODICE D'ESEMPIO PER IL CLIPPING]

Chapter 6

Tuning of weight clipping parameter

In the final part of the previous chapter the procedure of weight clipping was introduced and explained. In this one a systematic approch for weight clipping parameter tuning is developed. The procedure is then tested with several cases, where the difference is in the size of reference sample. It is important to highlight that in this procedure no signal event is given to the network because we are trying to search the optimal W for which the ditribution of t converges after training to a χ^2 ditribution with 96 dofs.

Beside searching the optimal weight clipping for each case, another aim is to find a trend for W in order to avoid a tuning when data samples of new sizes are given to the net. The cases studied for the test are listed below in Table 6.1.

n° of test	Reference size	Background size	Signal size
1	100k	20k	0
2	200k	20k	0
3	300k	20k	0
4	500k	20k	0
5	1000k	20k	0

Table 6.1: Sizes of the datasets for different tests.

6.1 A criterion for optimal weight clipping

As suggested in the previous discussion, the optimal weight clipping is reached when the sampled distribution of t resembles the expected ditribution of a χ^2 with 96 dofs. Therefore, after setting a value for W and sampling a certain number of observed t values, a compatibility test has to be applied with the expected ditribution. A good choice for this purpose is putting the sampled ts in a histogram and applying a χ^2 test between the latter and the expected distribution, put in a histogram as well. Now the possible cases are the following:

- If the observed χ^2 of the test statistic is beyond an equivalent threshold of 3σ, the observed distribution has converged to the reference one.
- If the threshold is not met, the weight clipping has to be lowered or increased depending on the trend of the observed χ^2 of test statistic during training.
  - ▷ If there is a valley and after reaching a minimum the observed χ^2 begins to grow, the weight clipping has to be lowered.
  - ▷ If there isn't a minimum but the observed χ^2 is still decreasing at the end of training, the weight clipping has to be increased.

It is obvious that this method could be computationally very expensive. Finding an optimal value for W requires more trials and every trial requires a certain time as the number of t needed to sample a distribution is $\gtrsim 10^2$. However, a good strategy to save a lot of time can be easily developed.

The starting point of the search should be the case with the smallest size of reference sample. In fact, training the network in this case requires a relative short time (for instance, ~ 8 h), so more trials can be done without an excessive loss of time. Found the optimal W for this case, the next reference sample in increasing order of size should be used for training. The value of optimal W is expected to be greater than the one in the previous case. The reason for this is giving more reference data to the network is equivalent to have more statistic and so the danger of finding divergences in the training is lower. Hence W can be set to a greater value as it is a regularization parameter. Moreover, if the size of reference is slightly increased, it is reasonable thinking that the optimal weight clipping should be slightly greater as well than in the previous case.

Bibliography

[1] Python 3 Documentation
https://docs.python.org/3/

[2] Keras Documentation
https://keras.io/

[3] TensorFlow Documentation
https://www.tensorflow.org/guide

[4] Cern Virtual Machine File System (CernVM-FS)
https://cernvm.cern.ch/portal/filesystem

[5] IBM Spectrum LSF V10.1 documentation
https://www.ibm.com/support/knowledgecenter/en/SSWRJV_10.1.0/lsf_welcome/lsf_welcome.html

[6] MadGraph5
https://cp3.irmp.ucl.ac.be/projects/madgraph/

[7] *Deep Learning*
Ian Goodfellow and Yoshua Bengio and Aaron Courville, 2016, MIT Press
http://www.deeplearningbook.org

[8] *Hands-On Machine Learning with Scikit-Learn & TensorFlow*
Aurélien Géron, 2017, Cambridge University Press.

[9] Large Hadron Collider (LHC)
https://home.cern/science/accelerators/large-hadron-collider

[10] CMS experiment at cern: detector description
https://cms.cern/detector

[11] *Introduction to Elementary Particle Physics*, 2nd edition
Alessandro Bettini, 2014, Cambridge University Press

[12] Decay of Z bosons
https://atlas.physicsmasterclasses.org/en/zpath_lhcphysics2.htm

[13] *Should you get excited by your data? Let the Look-Elsewhere Effect decide*
http://cms.web.cern.ch/news/should-you-get-excited-your-data-let-look-elsewhere-effect-decide

[14] *Learning New Physics from a Machine*
Raffaele Tito D'Agnolo, Andrea Wulzer, June 8, 2018
https://arxiv.org/abs/1806.02350

[15] *Searching for exotic particles in high-energy physics with deep learning*
P. Baldi, P. Sadowski & D. Whiteson, July 2, 2014
https://arxiv.org/abs/1402.4735