Alma Mater Studiorum · University of Bologna

School of Science
Department of Physics and Astronomy
Master Degree in Physics

Improving MadGraph importance sampling efficiency by using neural network

Supervisor:
Prof. Fabio Maltoni

Submitted by:
Luca Beccatini

Co-supervisor:
Dr. Olivier Mattelaer

Academic Year 2021/2022

Contents

1 Introduction **1**

2 Monte Carlo **3**
2.1 Numerical quadrature rules for integration 4
2.2 Monte Carlo integration 6
2.3 Variance reducing techniques 8
2.3.1 VEGAS-algorithm 10
2.4 Multi-channel Monte Carlo 11
2.4.1 Single-Diagram-Enhanced multi-channel integration 12
2.5 Sampling random variables 13
2.6 phase-space sampling 14

3 MadGraph integration and unweighting **18**
3.1 MadGraph integration algorithm 18
3.2 Events unweighting algorithm 19
3.2.1 Standard unweighting method 19
3.2.2 Surrogate unweighting method 21

4 Artificial neural network: basic idea and implementation **25**
4.1 Artificial neural networks 26
4.2 neural network surrogate 30
4.2.1 Implementation on $pp \to t\bar{t}$ process 30
4.2.2 Implementation on $pp \to e^-e^+ggd\bar{d}$ process 32

5 Results **35**
5.1 Results on $pp \to t\bar{t}$ 35
5.2 Results on $e^-e^+ggd\bar{d}$ 44
5.3 Comparison to the original algorithm 56

6 Conclusions **69**

Acknowledgements **74**

Abstract

In this work, we present a two-staged unweighting method for Monte Carlo event generation. This method is based on the use of an Artificial neural network surrogate, which is trained to predict the events weights. The surrogate is used in a first unweighting step in order to avoid the evaluation of the true weights for the rejected events. Then, a second unweighting accounts for the error committed by the surrogate. This algorithm can accelerate the unweighting thanks to the much faster evaluation of the surrogate in respect to the evaluation of true weight, and to a small allowed overweight in the final sample. We test this algorithm for two scattering processes at 13 TeV, which are: $pp \to t\bar{t}$ and $pp \to e^- e^+ g g d \bar{d}$.

Chapter 1

Introduction

At LHC, collisions that take place among high energy protons, allowing to study properties of the fundamental interactions and expand our knowledge of particle physics. In order to test this, data are systemically compared to theoretical predictions, which are provided by numerical codes generating events that fully simulate the physical process. MadGraph5_aMC@NLO [1] (here called MadGraph) is one of the main tools used for the simulations of the hard matrix-element (the first step of the simulation) and it used to generate billions of events. Thus, its efficiency and its speed are relevant parameters for the total CPU time during these simulations. Its speed is actually spotted has one of the factor to improve to be able to tackle the challenge of the limited CPU resource needed for the next phase of LHC: High Luminosity HLC [2, 3].

In this work, we describe an alternative method to speed up the generation of events following a given distribution using Machine Learning methods. A neural network surrogate is used to approximate the weight of the events, and we use these predictions in an acceptance-rejection sampling method to perform a fist unweighting. The evaluation of this surrogate is cheaper than the evaluation of true function, and for complex processes it can be much faster than the standard one. Only at this point, we compute the true weight of the accepted events, and we use it to perform a second unweighting, where we account for the error in the neural network predictions. This method allows avoiding the computation of the weights of many events, which are rejected in the first unweighting [4].

In section 2, we are going to describe the basics of Monte Carlo integration, giving a look to the VEGAS-algorithm and to the Single-Diagram-Enhanced multi-channel integration, which is used by MadGraph to compute the cross-section of events. Then we are going to briefly discuss how to generate samples in the phase-space of high-energy physics. In section 3, we will introduce the MadGraph integration algorithm and the standard method to unweight the events. After that, we will describe the proposed unweighting method based on the neural network surrogate. In section 4, we will describe how a neural network works, with a particular attention to the parameters tuned during

this work. Then we will describe the implementation of the neural network to the processes that we studied. In section 5, we will present the results of this approach on the processes that we studied, and we will make a comparison with the original algorithm. Finally, in section 6, we will give some concluding remarks and present some future outlook.

Chapter 2

Monte Carlo

In general, with Monte Carlo methods, we refer to a broad class of computational algorithms that solve mathematical problems using random variables with a known probability distribution. These methods are applied in a variety of fields for a wide range of applications, and are most useful when it is difficult or impossible to use other approaches.

One can say that a first rude variant of a Monte Carlo method was used in 1777 in the Buffon's problem [5]. In this first experiment, the value of π has been measured by counting how many times a needle, that was thrown over a sheet of paper, intersected with equally spaced parallel lines. Then, Enrico Fermi in the 1930s, used the Monte Carlo method for neutron diffusivity problem. However, he did not publish this work, as confirmed by Emilio Segrè [6], that was a student and a collaborator of Enrico Fermi. During the second world war, the Monte Carlo method was applied in the Manhattan Project to calculate the neutron diffusion in fissionable material [7]. The first theoretical essay about Monte Carlo methods belongs to Nicholas Metropolis and Stanislaw Ulam in 1949 [8]. While John von Neumann programmed one of the first computers, called ENIAC [9], to compute Monte Carlo calculations. The name "Monte Carlo method" was chosen by Nicholas Constantine Metropolis because it is based on probability, like gambling, and the roulette of the famous Monte Carlo Casino is a random numbers generator.

In this chapter, we will briefly introduce the Monte Carlo methods. We will start from the numerical quadrature rules, which belongs to the numerical integration family, as Monte Carlo methods, and help us to understand the reason Monte Carlo methods are so relevant in some fields. Then we will describe the basic idea behind Monte Carlo integration, followed by the description of some variance reducing techniques, which are fundamental to improve the efficiency of Monte Carlo integrations. We will show how to use Monte Carlo methods to produce samples with a given distribution, with a particular attention for phase-space sampling in high-energy physics at colliders.

2.1 Numerical quadrature rules for integration

The numerical quadrature rules are a large class of algorithms for calculating the numerical value of a definite integral. They rely on the evaluation of the integrand function on a finite set of points, that is used to produce an approximation of the integral. Usually the result is obtained through a weighted average of these integrand evaluations. In general, these integration methods do not produce an exact solution, but they try to reach a given level of precision with the fewest possible number of function evaluations.

The numerical integration methods are opposed to the analytical integration, where we can find an exact solution by finding the antiderivative. There could be several reasons to prefer the numerical integration methods to the analytical ones, for example if the integrand function is known only at certain points. Other examples are when it is impossible or too difficult to find the antiderivative of integrand, or when it is possible but it is much easier to compute a numerical approximation than the antiderivative.

A generic quadrature rule in one dimension is defined as:

$$\int_a^b dx\ f(x) = \sum_{i=1}^{N+1} w_i\, f(x_i) + Err(f) \quad , \tag{2.1}$$

where x_i are the set of $N+1$ quadrature points, w_i are the set of quadrature weights and $Err(f)$ is the error term of the integrand function. The error term is null (and so the numerical integration is exact) if the integrand is a polynomial of degree lower or equal to N. The numerical quadrature rules are divided into two categories: the Newton-Cotes type rules, which evaluate the integrand at equally spaced points, and the Gaussian quadrature rules, which evaluate the integrand at particular points (not equally spaced). In this work, we are going only to briefly introduce the first type of numerical rules.

The simplest method of the Newton-Cotes type formulae is the trapezoidal rule [10], that consists into approximate the one-dimensional area of the integral into consecutive trapezoids attached one to the other. Given a twice-differentiable integrand f, its integral between two points x_0 and $x_0 + \Delta x$ can be written as:

$$\int_{x_0}^{x_0+x} dx\ f(x) = \frac{\Delta x}{2}\Big(f(x_0) + f(x_0+\Delta x)\Big) - \frac{(\Delta x)^3}{12} f^{(2)}(\xi) \quad , \tag{2.2}$$

where ξ is a point included in the interval $[x_0, x_0+\Delta x]$ and $f^{(2)}$ is the second antiderivative of f. The eq.(2.2) represents the integral of f as the integral of the trapezoid described by the points $f(x_0)$, $f(x_0+\Delta x)$ and their projections on the x axis and a correction term. In general, to locate the point ξ, in order to compute the correction term, we need to know the integral $\int_{x_0}^{x_0+x} dx\ f(x)$, which would make needless the computation of the correction term. For this reason, usually the correction term is neglected, and an error is introduced in the numerical evaluation.

The eq.(2.2) can be extended to a generic interval $[x_0, x_N]$, that is divided by N quadrature points, such that $x_i = x_0 + i\Delta$, with $i = 0, .., N$, obtaining the formula for the numerical integral:

$$\int_{x_0}^{x_N} dx\ f(x) = \frac{x_N - x_0}{N} \sum_{i=0}^{N} w_i f(x_i) - \frac{1}{12} \frac{(x_N - x_0)^3}{N^2} \tilde{f}^{(2)} \quad , \tag{2.3}$$

where $w_i = 1/2$ for $i = 1, N$ and $w_i = 1$ for $i = 2, .., N-1$. The correction term is proportional to $\tilde{f}^{(2)}$ that is given by:

$$\tilde{f}^{(2)} = \frac{1}{N} \sum_{j=0}^{N} f^{(2)}(\xi_j) \quad , \tag{2.4}$$

where each point ξ_j is included in the interval $[x_{j-1}, x_j]$. Notice that, also here, the correction term is usually unknown and for this reason an error that is proportional to $1/(N+1)^2 \approx 1/N^2$ is introduced.

The Simpson's rule [10] represents an improved method in respect to the trapezoidal rule by the evaluation of each region in three points, instead of two. In particular, the integral of a four-times-differentiable function in an interval $[x_0, x_2]$ can be written as:

$$\int_{x_0}^{x_2} dx\ f(x) = \frac{\Delta x}{3} \Big(f(x_0) + 4f(x_1) + f(x_2) \Big) - \frac{(\Delta x)^5}{90} f^{(4)}(\xi) \ , \tag{2.5}$$

where $x_2 = x_1 + \Delta x = x_0 + 2\Delta x$. Generalizing to a larger interval divide in $N/2$ regions (N must be an even number), we find the compound formula:

$$\int_{x_0}^{x_N} dx\ f(x) = \frac{x_N - x_0}{N} \sum_{i=0}^{N} w_i f(x_i) - \frac{1}{180} \frac{(x_N - x_0)^5}{N^4} \tilde{f}^{(4)}(\xi) \ , \tag{2.6}$$

where $w_i = 1/3$ for $i = 1, N$, $w_i = 2/3$ for $i = 2, 4, .., N-2$ (even numbers) and $w_i = 4/3$ for $i = 3, 5, .., N-1$ (odd numbers). From eq.(2.6) we can notice that the error in the Simpson's rule is proportional to $1/N^4$.

It is possible to show that increasing the number of evaluations of the integrand in the region Δx, we have an increase of the accuracy of the numerical integration method. In particular, for an odd number of evaluation, say $2N-1$, the error term is proportional to:

$$c_{2N-1} (\Delta x)^{2N+1} f^{2N}(\xi) \ , \tag{2.7}$$

where c_{2N-1} is a constant, while for an even number of evaluation, say $2N$, the error term is proportional to:

$$c_{2N} (\Delta x)^{2N+1} f^{2N}(\xi) \ . \tag{2.8}$$

Each quadrature rule that has an error term proportional to f^{2N} is called of degree $2N-1$ and it is exact for polynomials up to the same degree.

Notice that a quadrature rule of degree k requires for the integrand function f to be k-times-differentiable and for $f^{(k)}$ to be continuous. Otherwise, if the integrand is not smooth enough, the error estimate is not reliable. Therefore, it is not always possible to increase the error estimate, increasing the order of the quadrature rule at will. Moreover, it is possible to show that for a large number of degree, the weights of the integrand in the Newton-Cotes formulae become large and of mixed sign, which can lead to large numerical cancellations.

One of the biggest issues about the Newton-Cotes formulae is that they are not efficient for multi-dimensional integral. In fact, we could adapt these rules to a generic d-dimensional integral by viewing it as an iteration of one-dimensional integral and perform a quadrature rule in each of them. However, the error term would be proportional to $N^{k/d}$, where N is the total number of points taken in each dimension and k is the degree of the quadrature rule used in each dimension. For a large number of dimensions the quadrature rules are affected by the so-called curse of dimensionality, where the volume of the space increases so rapidly with the number of dimensions that the available data become sparse. This implies that to achieve a reliable result, the required data must grow exponentially with the number of dimensions, becoming very computationally expensive.

2.2 Monte Carlo integration

The Monte Carlo integration methods belong to the family of numerical integration algorithms, and they are based on the evaluation of an integral through random sampling of the integrand. They differ from the quadrature rules because the points where integrand is evaluated are neither fixed nor carefully selected.

For example, we consider the integral of $f(u_1, .., u_d)$, which is a square-integrable function depending on d variables $u_1, .., u_d$. The integral is performed over the hypercube, $[0,1]^d$ and for simplicity we will describe a point in the hypercube as $x \equiv (u_1, .., u_d)$. So, we can write the integral I of f as:

$$I = \int dx f(x) = \int d^d u \, f(u_1, .., u_d) \; . \tag{2.9}$$

The Monte Carlo estimate E of I can be written as:

$$E = \frac{1}{N} \sum_{i=1}^{N} f(x_i) \; . \tag{2.10}$$

where x_i are N points of the hypercube independently sampled from a uniform distribution. We can notice that E has the same form of a quadrature rule for numerical integration, except that points are randomly sampled.

Using the law of large numbers, it is possible to show that the Monte Carlo estimate converges to the true value of the integral:

$$lim_{N\to\infty}\frac{1}{N}\sum_{i=1}^{N}f(x_i)=I \quad . \tag{2.11}$$

In the case of finite sample, this convergence is not valid, but we can obtain a probabilistic error bound of the estimate in function of its variance. The variance of the function f, $\sigma^2(f)$, is defined as:

$$\sigma^2(f)=\int dx(f(x)-I)^2 \quad . \tag{2.12}$$

The root square of the variance of f is called the standard deviation of f, $\sigma(f)$. It is possible to show that the error in the Monte Carlo estimate is on average:

$$\Delta E=\frac{\sigma(f)}{\sqrt{N}} \quad . \tag{2.13}$$

More details can be found in [11].

Notice that the convergence rate of the error is fixed to $1/\sqrt{N}$, that is a relatively slow rate. It implies that, for example, if we want to divide by 2 the error, we need to use a sample 4 times larger.

Through the central limit theorem, we can show that the probability that Monte Carlo estimate E is between $(I-a\sigma(f)/\sqrt{N})$ and $(I+b\sigma(f)/\sqrt{N})$ is given by:

$$lim_{n\to\infty}Prob\left(-a\frac{\sigma(f)}{\sqrt{N}}\leq E-I\leq b\frac{\sigma(f)}{\sqrt{N}}\right)=\frac{1}{\sqrt{2\pi}}\int_{-a}^{b}dt\,e^{-\frac{t^2}{2}} \quad . \tag{2.14}$$

This relation implies that the error in the Monte Carlo estimate is independent of the dimension d of the integral, which is a very important property of Monte Carlo integration.

Usually, we use an estimate S^2 for the variance $\sigma^2(f)$, due to the difficulty to compute its exact value

$$S^2=\frac{1}{N-1}\sum_{i=1}^{N}(f(x_i)-E)^2=\frac{1}{N}\sum_{i=1}^{N}(f(x_i))^2-E^2 \quad . \tag{2.15}$$

Initially we required that the function f is square-integrable, in fact, this is condition is required for the reliability of the error estimate. Otherwise, if the function f is only integrable, this will not be valid but the Monte Carlo estimate E will still converge to the true value I. Finally, if we substitute the estimate for the variance, eq.(2.15), in the

error formula of the Monte Carlo estimate, eq.(2.13), we find an estimate for the error of E:

$$\Delta E = \frac{1}{\sqrt{N}} \sqrt{\frac{1}{N}\sum_{i=1}^{N}(f(x_i))^2 - E^2} \quad . \tag{2.16}$$

Therefore, we have seen that Monte Carlo integration error goes like $1/\sqrt{N}$ independently of the dimensions. This is a big advantage of the Monte Carlo integration, especially for integrand with a large dimensions number, due to the fact that most of the integration methods, like the quadrature rules, have a convergence rate which decreases with the increase of the dimensions. For example, if we compare the Monte Carlo integration with the Simpson's rule, eq.(2.6), we have that the error in the first case goes like $N^{-1/2}$ and in the second case like $N^{-4/8}$. Thus, for a number of dimensions d lower than 8, the Simpson's rule is being better than the Monte Carlo integration, but for higher dimensionality it is much worse.

2.3 Variance reducing techniques

The Monte Carlo convergence rate is relatively slow and the sample size can be a limit for the estimate error. Thus, in order to improve the quality of Monte Carlo integration we need to reduce variance.

There are several variance reducing techniques, here we introduce some of the most relevant:

- importance sampling, that consists into a change of variable

$$\int dx f(x) = \int \frac{f(x)}{p(x)} p(x) dx = \int \frac{f(x)}{p(x)} dP(x) \tag{2.17}$$

  where $p(x) = \frac{\partial P(x)}{\partial x}$. If $p(x)$ is a positive-valued function and is normalized to unity, then it can be interpreted as a probability density function. Moreover, if we can generate a sample $x_1, .., x_N$ according to $P(x)$, we can evaluate the Monte Carlo estimate as

$$E = \frac{1}{N}\sum_{1=1}^{N} \frac{f(x_i)}{p(x_i)} \quad . \tag{2.18}$$

  The statistical error of the Monte Carlo integration becomes $\sigma(f/p)/\sqrt{N}$. Usually, to decrease the variance, we choose a function $p(x)$ that approximates $f(x)$ in shape, such that f/p is nearly constant, and that it is possible to generate easily samples according to $P(x)$. However, this method can be problematic in case of functions $p(x)$ which become null when $f(x)$ remains positive, generating an infinity in the variance.

- control variates, that consists into add and subtract a function $g(x)$ which approximates $f(x)$

$$\int dx f(x) = \int dx (f(x) - g(x)) + \int dx g(x) \quad . \tag{2.19}$$

If the primitive of $g(x)$ is known, then the statistical error of the Monte Carlo integration becomes $\sigma(f - g)/\sqrt{N}$. The variance will be lower if $g(x)$ is close to $f(x)$. Usually, if we have a function $h(x)$ which approximates $f(x)$, it is better to use the importance sampling method if $f(x)/h(x)$ is almost constant, and it is better to use the control variates if $f(x) - g(x)$ is almost constant [12]. Moreover, the control variates method is more stable than importance sampling because null values in $g(x)$ do not produce infinities in the variance [1].

- stratified sampling, that is based on subdivide the domain Ω into $\Omega_1, .., \Omega_k$ non-overlapping regions. The Monte Carlo estimator becomes

$$E = \sum_{j=1}^{k} \frac{v_j}{N_j} \sum_{i=1}^{N_j} f(x_{i,j}) \tag{2.20}$$

where v_j is the volume of the region Ω_j and N_j is the number of points sampled from the region Ω_j, while the resulting variance is

$$\sum_{j=1}^{k} \frac{v_j^2}{N_j} \sigma_j^2(f) \quad . \tag{2.21}$$

As a general rule, to minimize the variance we have to take in each region a number of points N_j proportional to the variance of that region Ω_j [11]. This method is more useful in the case of integrand with low dimension and without singularities or rapid oscillations [13]. However, a bad choice of the subspaces Ω_j and of the number of points N_j can lead to an increase of the variance.

- antithetic variates, that is based on taking correlated points instead of independent points. This method uses the

$$var(f_1 + f_2) = var(f_1) + var(f_2) + 2covar(f_1, f_2) \tag{2.22}$$

where the covariance term $covar(f_1, f_2)$ can be negative [14]. Thus, carefully choosing points that are negative correlative, it is possible to decrease the variance.

[1] In the case of event generation, the control variates method could be more problematic. In fact, it can generate events with negative weights, causing a loss in the efficiency. In particular for events at the leading order, where we usually have positive weights. While the control variates method could be used for the next-to-leading order calculations where negative weights are introduced by definition.

One relevant difference between variance reduction techniques is how they learn about the function to integrate. The techniques described above are based on the information known from the integrand a priori. However, there is another type of techniques, called adaptive Monte Carlo methods, which learn about the function as the algorithm proceeds. For example, an adaptive stratified sampling method consists into concentrate more samples where the integrand has the most variation in order to reduce the variance. The domain is subdivided in subspaces and the Monte Carlo integration is performed in each subspace to estimate the variance of that region. Then, if the variance of a region is larger than a given threshold, that region is further subdivided in smaller subspaces. This process is iterated up to each region has an acceptable variance.

2.3.1 VEGAS-algorithm

The VEGAS-algorithm [15, 16] is an adaptive Monte Carlo method, and it is widely diffused in high energy physics. It consists into subdividing the domain in subspaces and using importance sampling, taking more samples in those regions where the integrand is largest. This is done by dividing the domain in a rectangular grid and perform the integration in each region. Depending on the magnitude of the integrand in each subspace, the grid is improved for the next iteration. The goal of these iterations is to achieve an optimal grid with step-like probability density functions $p(x)$.

The Monte Carlo estimate of a given iteration is:

$$E_j = \frac{1}{N_j} \sum_{i=1}^{N_j} \frac{f(x_i)}{p(x_i)} \tag{2.23}$$

and the estimate for the variance is:

$$S_j^2 = \frac{1}{N_j} \sum_{i=1}^{N_j} \left(\frac{f(x_i)}{p(x_i)} \right)^2 - E_j^2 \tag{2.24}$$

where N_j is the number of points of iteration j. These estimates are used to compute a cumulative estimate such that:

$$E = \left(\sum_{j=1}^{m} \frac{N_j}{S_j^2} \right)^{-1} \left(\sum_{j=1}^{m} \frac{N_j E_j}{S_j^2} \right) \quad , \tag{2.25}$$

where m is the number of total iterations.

To compute an integral in d dimensions with the importance sampling, VEGAS uses a separable probability density function $p(x)$ such that:

$$p(x) = p_1(u_1) \cdot .. \cdot p_d(u_d) \quad . \tag{2.26}$$

Thus, it is possible to study individually each axis and obtain d different probability density functions p_i. Thanks to this, the number of bins of the grid goes like $K \cdot d$, where K is the number of bins in each dimension. Otherwise, using a not separable probability density function, the number of bins would go as K^d, generating a curse of dimensionality. However, this method implies that the result of VEGAS depends on the validity of the approximation 2.26.

2.4 Multi-channel Monte Carlo

In the case where the integrand function $f(x)$ has several sharp peaks, there could not be a single transformation able to map efficiently all the peaks. Thus, the importance sampling would obtain poor results trying to reduce the variance of the integrand function, in particular if the peaks belong to different regions. To resolve this problem, we could use the multi-channel Monte Carlo techniques[17]. The only condition for this method is to know a transformation which can flatten each peak, separately. Each transformation is called channel, and it has a probability density function $p_i(x)$, which is non-negative and normalized to unity. Moreover, each region has a mapping function $x = P_i^{-1}(y)$ from random numbers y, generated following $p_i(x)$, into the region of integration. Let us consider m different channel and m non-negative constants λ_i, with $\sum_{i=0}^{m} \lambda_i = 1$. Each λ_i corresponds to the weight of its channel. Given the total number of integrand evaluations N, each channel is evaluated about $N_i = N\lambda_i$ times. We can rewrite the integral I as

$$I = \int dx f(x) = \sum_{i=1}^{m} \lambda_i \int dP_i(x) \frac{f(x)}{p(x)} \tag{2.27}$$

with $p(x) = \sum_{i=0}^{m} \lambda_i p_i(x)$ and $dP_i(x) = dx\, p_i(x)$. The Monte Carlo estimates becomes

$$E = \frac{1}{N} \sum_{i=1}^{m} \sum_{j_i=1}^{N_i} \frac{f_{j_i}(x)}{p_{j_i}(x)} \quad . \tag{2.28}$$

The integration error is

$$\Delta E = \sqrt{\frac{W(\lambda) - I^2}{N}} \tag{2.29}$$

where $W(\lambda)$ is given by

$$W(\lambda) = \sum_{i=0}^{m} \lambda_i \int dx P_i(x) \Big(\frac{f(x)}{p(x)}\Big)^2 \quad . \tag{2.30}$$

We can notice that the parameters λ_i do not affect the Monte Carlo estimate E, thus we can tune those parameters to minimize the function $W(\lambda)$ and so also the integration error. Moreover, due to the non-dependence of the integral I on the parameters λ_i, the tuning of the parameters can be done also during the integration.

2.4.1 Single-Diagram-Enhanced multi-channel integration

To compute the cross-section of a given process, we need to integrate a generic squared amplitude over the phase-space of the final-state particles. If we look to the cross-section of a generic $2 \to n$ parton-level process, the integral to compute is:

$$\sigma = \frac{1}{flux} \sum_{a,b} \int dx_a \, dx_b \, f_a(x_a) \, f_b(x_b) \int d\Phi_n |\mathcal{M}_{a,b\to n}|^2 \mathcal{J}_{\Phi_n} \tag{2.31}$$

where $f_a(x_a), f_b(x_b)$ are the parton distribution function of the initial parton a, b, Φ_n is the phase-space of the final particles, $\mathcal{M}_{a,b\to n}$ is the amplitude of the process and $\mathcal{J}_{\Phi_n}$ is the Jacobian factor of variable mappings of the Lorentz invariant phase-space element Φ_n.

To compute the cross-section, MG5aMC uses a revisited multi-channel Monte Carlo method, called Single-Diagram-Enhanced multi-channel integration. The idea behind this method consists into taking advantage of the physical content of the process to obtain a natural basis f_i to decompose the integrand function f. In fact, the peak structure of a channel f_i is equal to the peak structure of the amplitude squared $|\mathcal{M}_i|^2$ of the corresponding Feynman diagram. For this reason, using a standard Feynman diagram expansion is possible to rewrite f as a sum of m channels such that the peak structure of each diagram is efficiently mapped by p_i. This basis is given by:

$$f_i = \frac{|\mathcal{M}_i|^2}{\sum_{i=1}^{m} |\mathcal{M}_i|^2} |\mathcal{M}_{a,b\to n}| \quad . \tag{2.32}$$

Thanks to the basis defined by eq.(2.32), we can rewrite the integral of the amplitude over the phase-space as

$$I = \int d\Phi |\mathcal{M}_{a,b\to n}|^2 = \sum_{i=1}^{m} \int d\Phi_i |\mathcal{M}_{a,b\to n}|^2 \frac{|\mathcal{M}_i|^2}{\sum_{j=1}^{m} |\mathcal{M}_j|^2} \quad . \tag{2.33}$$

Now we can define the values $\lambda_i \equiv \frac{|\mathcal{M}_i|^2}{\sum_{j=1}^{m} |\mathcal{M}_j|^2}$ as the channel weights, obtaining the multi-channel integral

$$I = \sum_{i=1}^{m} \int d\Phi_i \lambda_i |\mathcal{M}_{a,b\to n}|^2 \quad . \tag{2.34}$$

About the validity of the weights, it is trivial to show that $\sum \lambda_i = 1$ with $\lambda_i \geq 0 \forall i$.

In the classical limit, where the interference terms between the Feynman diagrams are small, we have that:

$$\sum_{i=1}^{m} \frac{|\mathcal{M}_{a,b\to n}|^2}{\sum_{j=1}^{m} |\mathcal{M}_j|^2} \approx 1 \quad . \tag{2.35}$$

Therefore, it is possible to efficiently map the channel f_i through an appropriate probability density function p_i derived from the propagator of the corresponding Feynman diagram. Moreover, in the classical limit we are able to compute the integral I as the sum of integrals of single diagrams squared:

$$I \approx \sum_{i=1}^{m} \int |\mathcal{M}_i|^2 \quad . \tag{2.36}$$

The integral of eq.(2.36) has the advantage to have poles easy to identify, which greatly simplifies the use of importance sampling. To check the validity of the approximation of eq.(2.35) and other sources of deviation from the ideal case, MadGraph relies on a modified VEGAS algorithm [18].

The standard approach of the multi-channel Monte Carlo, described in (2.4), requires computing the weight of each channel for each point in the phase-space. This issue can be problematic in the case of a large number of channels or for computationally expensive probability density functions $p_i(x)$. In the case of Sherpa [19], the evaluation of the channel weight is as prohibitive as the computation of the matrix-element, spending roughly half of the total CPU time. Note that even if the weight of a channel becomes very small during the tuning of the weights λ_i, the time spent to compute the weight of that channel does not change. Instead, the Single-Diagram-Enhanced multi-channel method does not require computing the weight in each channel for each phase-space point. This implies that the computational cost of the integration does not increase with the number of channels.

A positive side of the MG5aMC approach is that the number of channel to be included in the basis is arbitrary and, usually, it is smaller than the number of Feynman diagrams. For example, this approach avoids multiple Feynman diagrams with identical particles in the final state or not-relevant channels with no peaks or multiple Feynman diagrams mapped in a similar way, like radiation coherence.

Another positive aspect is that the integration is done in an embarrassingly parallel way [2]. Thus, the computation of different channels can be performed by different resources without any communication between them and then combine the results at the end.

2.5 Sampling random variables

Usually, random processes are described by given distribution functions $p(x)$, so, one needs to produce samples with that distribution. To generate a sampling with a given

[2]A computation is said embarrassingly parallel when it can be trivially separated into parallel tasks which require very little or no communication between them [20].

distribution we need to have initially a sample with some random distribution, for simplicity we can take the uniform one.

One simple method to do it is the inverse transform method. We consider a positive distribution function $p(x)$ and its cumulative distribution function, $P(x)$ defined as:

$$P(x) = \int_0^x dx' p(x') \tag{2.37}$$

such that $P(x)$ gives the probability that $x' \leq x$. If we take a uniform distributed random variable u between 0 and 1, we can generate random numbers x from $p(x)$ through the inverse of its cumulative density distribution as:

$$x = P^{-1}(u) \quad . \tag{2.38}$$

We can notice that to use this method we need to know the function P^{-1}.

Another general algorithm to generate sampling of random variables is the acceptance-rejection method [21]. This method consists into generate events from a uniform distribution and accept them with a probability depending on the target distribution $p(x)$. First of all, we need to define a constant c such that $p(x) \leq c \ \forall x$. Then we need to generate a candidate event x from a uniform distribution and u from a uniform distribution between 0 and 1. If $u \cdot c \leq p(x)$ the candidate event is accepted as an event of the distribution $p(x)$, otherwise it is rejected. This method does not require the inverse cumulative distribution, so it can be used when $P(x)$ is too difficult or impossible to compute.

2.6 phase-space sampling

To evaluate observables in high-energy physics at colliders, usually we need to integrate over the Lorentz-invariant phase-space $d\Phi_n(P, p_1, .., p_n)$ of the n outgoing particles, where P is the total four-momentum and p_i the four-momentum of the i outgoing particle, and over the Bjorken fractions of the initial states. Therefore, to evaluate the phase-space, we need to generate the $p_1, .., p_n$. The lorentz-invariant phase-space is given by:

$$\begin{aligned} d\Phi_n &= \prod_{i=1}^{n} \frac{d^4 p_i}{(2\pi)^3} \Theta(p_i^0) \delta(p_i^2 - m_i^2) (2\pi)^4 \delta^4 \Big(P - \sum_{i=1}^{n} p_i \Big) \\ &= \prod_{i=1}^{n} \frac{d^3 p_i}{(2\pi)^3 2E_i} (2\pi)^4 \delta^4 \Big(P - \sum_{i=1}^{n} p_i \Big) \end{aligned} \tag{2.39}$$

where m_i is the mass of the i outgoing particle. In the case of massless particles, the phase-space volume is:

$$\Phi_n = \int d\Phi_n = (2\pi)^{4-3n} (\frac{\pi}{2})^{n-1} \frac{(P^2)^{n-2}}{\Gamma(n)\Gamma(n-1)} \quad . \tag{2.40}$$

The phase-space of n particles can be written as the phase-space of two fictitious particles decay:

$$d\Phi_n = \frac{1}{2\pi} dQ^2 d\Phi_j(Q, p_1, .., p_j) d\Phi_{n-j+1}(P, Q, p_{j+1}, .., p_n) \tag{2.41}$$

with $Q = \sum_{i=1}^{j} p_i$.

Using this last relation, it is possible to generate a n-body phase-space considering the whole event as obtained from a sequence of fictitious two-body decays [22]. For example, in the case of 4 outgoing particles A, B, C, D, it could be described like

$$ABCD \rightarrow ABC + D \rightarrow AB + C + D \rightarrow A + B + C + D\ . \tag{2.42}$$

To apply this method we substitute sequentially the: eq.(2.41) in the eq.(2.39), obtaining the formula:

$$d\Phi_n = \frac{1}{(2\pi)^{n-2}} dM_{n-1}^2 \cdot .. \cdot dM_2^2 d\Phi_2(q_n, q_{n-1}, p_n) \cdot .. \cdot d\Phi_2(q_2, p_1, p_2) \tag{2.43}$$

where $q_i = \sum_{j=1}^{i} p_j$ is the total four momentum of the fictitious particles i and $M_i^2 = q_i^2$ is its invariant mass squared. Each invariant mass M_i has an allowed region equal to $(m_1 + .. + m_i)^2 \leq M_i^2 \leq (M_{i+1} - m_{i+1})^2$. The phase-space for two particles produced by q_i in the rest frame of q_i can be written as:

$$d\Phi_2(q_i, q_{i-1}, p_i) = \frac{1}{(2\pi)^2} \frac{\sqrt{\lambda(q_i^2, q_{i-1}^2, m_i^2)}}{8q_i^2} d\varphi_i\, d\cos\theta \tag{2.44}$$

where λ is the Källen function defined as:

$$\lambda(a^2, b^2, c^2) = (a^2 - b^2 - c^2)^2 - 4b^2c^2 \quad . \tag{2.45}$$

In the case of two outgoing particles ($i = 2$), the previous formula becomes

$$d\Phi_2 = \frac{|\vec{p}|}{16\pi^2 M} d\cos\theta\, d\varphi \quad , \tag{2.46}$$

where the modulus of the tri-momenta are given by the Källen function λ

$$|\vec{p}| = \frac{\sqrt{\lambda(M^2, m_1^2, m_2^2)}}{2M} \quad . \tag{2.47}$$

This case requires only to generate the direction of one outgoing particle, which is uniformly distributed over the unit sphere. Then the other particle will move in the opposite direction.

The algorithm of this approach [11] can be summarized by:

1. set $i = n$, $q_i = P$ and $M_i = \sqrt{q_i^2}$

2. change to the q_i rest frame
3. generate 2 uniform random variable $u_{i,1}, u_{i,2}$ and set $\phi_i = 2\pi u_{i,1}$ and $cos\theta_i = u_{i,2}$
4. if $i > 2$, generate a uniform random variable $u_{i,3}$ and set $M_{i-1} = m_1 + .. + m_{i-1} + u_{i,3}(M_i - m_i)$; if $i = 2$ set $M_1 = m_1$
5. evaluate $|\vec{p}_i'(M_i^2, M_{i-1}^2, m_i^2)|, p_i'$ and $q_{i-1}' = \left(\sqrt{|\vec{p}_i'|^2 + M_{i-1}^2}, -\vec{p}_i'\right)$
6. return to the original frame
7. set $i = i - 1$ and go to step 2

This algorithm produces weighted events, where the weight of the events is

$$w = (2\pi)^{4-3n} 2^{1-2n} \frac{1}{M_n} \prod_{i=2}^{n} \frac{\sqrt{\lambda(M_i^2, M_{i-1}^2, m_i^2)}}{M_i} \quad . \tag{2.48}$$

An other method to generate n-body phase-space is using the RAMBO algorithm [23]. The RAMBO algorithm samples the phase-space almost uniformly and it produces uniformly weighted events for n massless outgoing particles. Let us study the quantity

$$\begin{aligned} R_n &= \int \prod_{i=1}^{n} \frac{d^4 q_i}{(2\pi)^3} \theta(q_i^0)\delta(q_i^2)(2\pi)^{4-2n} f(q_i^0) \\ &= (2\pi)^{4-2n}\Big(\int_0^\infty dx x f(x)\Big)^n \end{aligned} \tag{2.49}$$

The R_n formula is similar to the phase-space volume formula for massless particles but here the momenta q_i are not bounded by momentum conservation relations and the momenta have some weight function f such that f keeps the total volume finite. The not bounded four momenta q_i are related to the physical four momenta p_i through the relations

$$p_i^0 = x(\gamma q_i^0 + \vec{b}\vec{q}_i) \ , \quad \vec{p} = x(\vec{q}_i + \vec{b}q_i^0 + a(\vec{b}\vec{q}_i)\vec{b})) \tag{2.50}$$

where:

$$\begin{aligned} &Q = \sum_{i=1}^{n} q_i \ , \quad M = \sqrt{Q^2} \ , \quad \vec{b} = -\frac{1}{M}\vec{Q}, \\ &\gamma = \frac{Q^0}{M} \ , \quad a = \frac{1}{1+\gamma} \ , \quad x = \frac{\sqrt{P^2}}{M} \quad . \end{aligned} \tag{2.51}$$

If we apply a change of variables to R_n and we choose $f(x) = e^{-x}$, then it is possible to show that $R_n = \Phi_n S_n$ with:

$$S_n = 2\pi(P^2)^{2-n} \frac{\Gamma(\frac{3}{2})\Gamma(n-1)\Gamma(2n)}{\Gamma(n+\frac{1}{2})} \quad . \tag{2.52}$$

Thus, starting from the four momenta q_i, we can compute the phase-space of n massless particles. The algorithm is given by:

1. generate $4n$ uniform random variables $u_{i,1}, .., u_{i,4}$ between 0 and 1. Set the q_i four vectors equal to

$$
\begin{aligned}
q_i^0 &= -ln(u_{i,3}u_{i,4}) \\
q_i^x &= q_i^0 \sqrt{1-(2u_{i,1}-1)^2} cos(2\pi u_{i,2}) \\
q_i^y &= q_i^0 \sqrt{1-(2u_{i,1}-1)^2} sin(2\pi u_{i,2}) \\
q_i^z &= q_i^0 (2u_{i,1}-1)
\end{aligned}
\tag{2.53}
$$

2. evaluate the physical four momenta p_i through the transformation of eq. (2.50).

This method produces events with the same weight, that is given by

$$
w_0 = (2\pi)^{4-3n}(\pi/2)^{n-1}\frac{(P^2)^{n-2}}{\Gamma(n)\Gamma(n-1)} \quad . \tag{2.54}
$$

Now it is possible to pass from the phase-space of n massless particles to the phase-space of n massive particles with a change of variables in the momenta, assuming that P^2 stays the same. In fact, we can define the four momenta k_i such that:

$$
k_i^0 = \sqrt{m_i^2 + \xi^2 (p_i^0)^2}, \quad \vec{k}_i = \xi \vec{p}_i \tag{2.55}
$$

where ξ is the solution of the equation:

$$
\sqrt{P^2} = \sum_{i=1}^{n} \sqrt{m_i^2 + \xi^2 (p_i^0)^2} \quad . \tag{2.56}
$$

Usually eq.(2.56) has no analytic solution, so ξ must be computed numerically. The phase-space respect the massive momenta k_i has no longer constant weight and the weight of each event becomes $w = w_0\, w_m$, with $w_m = w_m(P, k_1, .., k_n)$.

Chapter 3

MadGraph integration and unweighting

MadGraph is a framework which aims to provide all the elements necessary for the study of the Standard Model and Beyond the Standard Model phenomenology. MadGraph is the new version of the MadGraph softwares, which involves several automated tools to perform different tasks.

In particular, MadGraph can produce elements like the computations of cross-sections, the generation of hard events and their matching with event generators, and the use of a variety of tools relevant to event manipulation and analysis. It is an open source software that can generate matrix elements at the tree-level for any renormalizable or effective Lagrangian based model. It allows the user to select a given process in a High Energy Physics, like a decay or a $2 \to n$ scattering, through the choice of the initial and final particles. Moreover, it allows for additional criteria about the process like: requiring resonances and decay chains or excluding resonances and internal particles. The standard reference for the use of the code is ref. [1].

3.1 MadGraph integration algorithm

MadGraph is one of the main tools which belong to the MadGraph family. Starting from a given process, MadGraph generates all the relevant Feynman diagrams for that process and produces the mappings for the integration over the phase-space.

The algorithm for the diagram generation used by MadGraph take advantage of the information of the model to produce only valid diagrams. This is done through consecutive iterations, which create sub-diagrams from the diagrams by merging legs[24]. The output result of MadGraph is a computer code that is used to evaluate the matrix element at a given phase-space points. This work is done by a package, called MadEvent, which receives as input the computer code produced by MadGraph. Then, MadEvent

can produce the cross-section or decay width calculation and unweighted event generation. After the generations of the events, they can be passed to any shower Monte Carlo program, where partons are perturbatively evolved through the emission of QCD radiation, and eventually turned into physical states through hadronization.

As we explained in chapter 2, Monte Carlo techniques are used to perform the integration of the squared amplitude over the phase-space. However, the amplitude usually has many sharp peaks in different regions of the phase-space, which makes really difficult the integration. In general, these peaks are due to one or more propagators in one or more regions which become large. Usually, this happens in proximity of a collinear-divergent limit or in the limit of a virtual massive particle, that has a small width, reaching its mass-shell. In order to solve this issue, we need to locate these peaks and then to map them onto several different sets of variables, called as channels. The position and the shape of the peaks in the matrix elements can be obtained with a standard Feynman diagram expansion.

3.2 Events unweighting algorithm

As we have seen in the section (2.6), the phase-space sampling methods generate weighted events. However, it is recommended to unweight the events in order to reduce the size of the sample of events and, so, decrease the CPU cost.

3.2.1 Standard unweighting method

In general, the events unweighting methods are based on the acceptance-rejection techniques [25]. Starting from N_0 weighted events, we use a rejection sampling algorithm to obtain N unweighted events, with $N \leq N_0$ corresponding to the accepted events. The unweighting efficiency is defined as $\varepsilon = \frac{N}{N_0}$, but for large samples N we can describe the efficiency as

$$\varepsilon = \frac{\sum_{i=0}^{N_0} w_i}{N_0 w_{\text{thres}}} = \frac{\langle w \rangle}{w_{\text{thres}}} \quad , \tag{3.1}$$

where w_i is the weight of the i-th event, w_{thres} is the maximum of the weights and $\langle w$ is the mean weight over the sample. This last formula represents the expected efficiency given by the sum of the probability to keep each event, and it has the advantage to avoid the fluctuations of the acceptance-rejection method.

Usually it is better to avoid choosing w_{thres} as the true maximum of the weights. In fact, in case of large samples, there could be some rare outliers much larger respect to the other events. As we can see from eq.(3.1), $w_{\text{thres}} \gg \langle w \rangle$ would lead to an unweighting efficiency close to zero. For this reason, in case of large outliers, it is better to define a reduced maximum w_{thres} such that $w_{\text{thres}} \leq max(w)$. All the events with $w \leq w_{\text{thres}}$ will be completely unweighted, with a neutral correction weight $\tilde{w} = 1$, but those “overweight”

events with $w > w_{\text{thres}}$ will have a correction weight $\tilde{w} = w/w_{\text{thres}} > 1$. This method is called partially unweighting of the events because it produces a sample which contains completely unweighted events ($\tilde{w} = 1$) and events with a correction weights ($\tilde{w} > 1$) to account for their probability to be accepted larger than 100%.

The main advantage of the partial unweighting is that it produces always samples with a larger efficiency than the complete unweighting (except in case of statistical fluctuation). However, the main issue of this method is that we need to generate a larger sample of partially unweighted events to obtain the same precision achieved by a smaller completely unweighted sample. To evaluate this flaw between the two samples, we introduce the effective sample size N_{eff} as an estimate of the size of a simple random sample required to achieve the same precision of the partially unweighted sample of size N, with $N_{\text{eff}} < N$. To estimate, N_{eff} we use the Kish factor α [26] that is given by:

$$\alpha = \frac{\left(\sum_{i=0}^{N} \tilde{w}_i\right)^2}{N \sum_{i=0}^{N} (\tilde{w}_i)^2} \quad . \tag{3.2}$$

Then, the effective sample size is obtained as

$$N_{\text{eff}} = \alpha N \quad , \tag{3.3}$$

where N is the size of a fully unweighted sample with unitary weight. This implies that if we use a partially unweighting method, to have the equivalent distribution of events (up to the statistical fluctuation) of a completely unweighting methods with a size sample N, we need to generate a sample with a size equal to N/α.

The Kish factor is equal to 1 in case of no overweight and it is lower than 1 in case of overweight, with its value that decreases with the increase of the overweight in the sample. Therefore, a large overweight in the final sample (that corresponds to a small α), must be compensated by the generation of a larger sample, which could compromise the gain in performance of the partially unweighting method.

A classical unweighting algorithm with overweight can be summarized as:

1. Compute the reduced maximum weight w_{thres}.
2. Generate a phase-space point u.
3. Compute the weight w of the point u.
4. Generate a uniform random r between 0 and 1.
5. If $w > r \cdot w_{\text{thres}}$: accept the phase-space point u and set its correction weight $\tilde{w} = max(1, w/w_{\text{thres}})$; otherwise reject u.
6. Iterate from step 2.

This algorithm is equivalent to the acceptance-rejection method described in the section (2.3). If w_{thres} corresponds to the true maximum, there is no overweight, so $\alpha = 1$.

Theoretically an optimal sampler, would generate events with almost uniform weights, in order to have an unweighting efficiency close to 100%. However, this is far from the real case. For example, in the processes that we are going to study with MadGraph we have an unweighting efficiency about between 1% and 10%, depending on the process. Variance reduction techniques, like those described in the section (2.3), are already used to improve these results.

3.2.2 Surrogate unweighting method

A problematic aspect of the event generation is the evaluation of the events weight, that can have a high computational cost. Moreover, this method can be particularly inefficient due to the fact that the evaluation is done for all events, even those which have a very small probability to be kept. To improve this aspect, we can substitute the evaluation of the weights with a surrogate function that well approximates it and that has a much lower computational cost.

Following this idea, this study uses an algorithm for the event unweighting which takes advantage of the Artificial neural network in order to be more computationally efficient. This is done substituting the Artificial neural network prediction of the weights to the computation of the real weights of each event, at least in a first step. The neural network model is firstly trained over a large dataset of phase-space points and then it is used to predict the weights of the phase-space points to sample.

The prediction s of the phase-space point u is used to perform a first unweighting with a rejection sampling algorithm in respect a reduced maximum s_{thres} of the surrogate weights. Then, to consider the possible errors in the neural network predictions s, we perform a second unweighting over the ratio $x = w/s$ between the true weight and the surrogate weight. Similarly to the first unweighting, the second one is also performed respect a reduced maximum x_{thres}. The algorithm of this method can be summarized as:

1. Generate N_0 phase-space points u_i.
2. Compute the surrogate weight s_i for each u_i.
3. Compute the reduced maximum s_{thres}.
4. Generate N_0 uniform random number $r_{1,i}$ between 0 and 1.
5. If $s_i \geq r_{1,i} \cdot s_{\text{thres}}$: accept the event u_i and set its correction weight

$$\tilde{w}^{(1)} = \max\left(1, \frac{s_i}{s_{\text{thres}}}\right) \quad ; \tag{3.4}$$

otherwise reject it.

6. Compute the true weight of the N_1 accepted events.
7. Compute the ratio $x_i = w_i/s_i$ of the accepted events.
8. Compute the reduced maximum x_{thres}.
9. Generate N_1 uniform random number $r_{2,i}$ between 0 and 1.
10. If $x_i \cdot \tilde{w}_i^{(1)} \geq r_{2,i} \cdot x_{\text{thres}}$: accept the event u_i and set its correction weight

$$\tilde{w}_j^{(2)} = \max\left(1, \frac{x_i \tilde{w}_i^{(1)}}{x_{\text{thres}}}\right) \quad ; \tag{3.5}$$

    otherwise reject it.

At the end we obtain a sample of N_2 accepted events from the second unweighting, with correction weights $\tilde{w}_i^{(2)}$, where $i = 1, .., N_2$, and a Kish factor $\alpha = \alpha(\tilde{w}^{(2)})$. We define an unweighting efficiency for the first unweighting $\varepsilon = N_1/N_0$, another one for the second unweighting $\varepsilon = N_2/N_1$ and a total one $\varepsilon = N_2/N_0$, with $\varepsilon = \varepsilon_1 \cdot \varepsilon_2$. Theoretically, an optimal surrogate would imply $x_i = w_i/s_i \simeq 1$ for most the events, which would lead to $\varepsilon_2 \simeq 1$. This case would correspond to the maximal computational gain because we would compute the true weights almost only for accepted events. However, this is not possible and the values of x can span several orders of magnitude, depending on the accuracy of the predictions.

There are several techniques to define a reduced maximum, in this work we will use a technique called quantile reduction method. This method consists into define a maximum such that the contribution of the remaining overweighted events corresponds to a fraction of the total sum of the events. Given N events with weights $s_1, .., s_N$, sorted by the smallest up to the largest (such that $s_i < s_{i+1} \forall i$), and an overweight parameter r_s, the reduced maximum is defined as:

$$s_{\text{thres}} = \min\left(s_j \,\middle|\, \sum_{i=j+1}^{N_0} s_i \leq r_\text{s} \cdot \sum_{i=1}^{N_0} s_i\right) \quad . \tag{3.6}$$

In a similar way, we use the same maximum definition for the second unweighting, but in this case we evaluate the maximum respect the quantity $x_i \cdot \tilde{w}_{1,i}$. Given N_1 accepted events by the first unweighting, with ratios x_i and correction weights $\tilde{w}_i^1$, sorted such that $x_i \cdot \tilde{w}_{1,i} < x_{i+1} \cdot \tilde{w}_{1,i+1}$, and an overweight parameter r_x, the reduced maximum is defined as:

$$x_{\text{thres}} = \min\left(x_j \cdot \tilde{w}_{1_j} \,\middle|\, \sum_{i=j+1}^{N_1} x_i \cdot \tilde{w}_i^{(1)} \leq r_\text{x} \cdot \sum_{i=1}^{N_1} x_i \cdot \tilde{w}_i^{(1)}\right) \quad . \tag{3.7}$$

The choice of performing the second unweighting respect the quantity $x_i \cdot \tilde{w}_i^{(1)}$, instead of x_i, is motivated by the reabsorption of the first correction weights $\tilde{w}^{(1)}$. Thus, the final correction overweight $\tilde{w}$ of an unweighted event can be approximated to the correction overweight of the second unweighting $\tilde{w}^{(2)}$, given by the eq.(3.5), such that:

$$\tilde{w} \simeq \tilde{w}^{(2)} = \max\left(1, \frac{x \cdot \tilde{w}^{(1)}}{x_{\text{thres}}}\right) \quad . \tag{3.8}$$

Notice that, for any accepted events in the second unweighting, if $x_i \tilde{w}_i^{(1)} < x_{\text{thres}}$, then the overweight of the first unweighting is completely reabsorbed ($\tilde{w}_i^{(2)} = 1$). As we will see in the chapter 5, with a proper choice of the overweight parameters, the overweight of the first unweighting is completely reabsorbed in almost all the cases. Notice that there are no conditions on the choice of the overweight parameters of the two unweightings. Thus, they can be chosen also different values between them, $r_{\text{s}} \neq r_{\text{x}}$. We will discuss the choice of these parameters in the chapter 5.

To measure the possible gains in timing and efficiency of the surrogate method respect the standard unweighting method of MadGraph, we define the effective gain factor f_{eff} as

$$f_{eff} = \frac{T_{st}}{T_{su}} \quad , \tag{3.9}$$

where T_{st} and T_{su} are respectively the total time to produce N unweighted events for MadGraph and for the surrogate algorithm. We can rewrite this formula defining the average time $\langle t_{st} \rangle$ to compute the weight of an event and unweight it with the standard method, its initial number of events N_0^{st} and its efficiency ε_{st} such that

$$T_{st} = N_0^{st} \cdot \langle t_{st} \rangle = \frac{N}{\varepsilon_{st}} \langle t_{st} \rangle \quad . \tag{3.10}$$

We do the same also for the surrogate

$$T_{su} = N_0^{su} \cdot \langle t_{su} \rangle = \frac{N/\alpha}{\varepsilon_{su}} \cdot \Big(\langle t_{nn} \rangle + \varepsilon_1 \langle t_{st} \rangle \Big) \quad , \tag{3.11}$$

where $\langle t_{su} \rangle$ accounts for both the average times of the neural network surrogate evaluation and the first unweighting step ($\langle t_{nn} \rangle$) and of the true weights computation of the accepted events and the second unweighting ($\varepsilon_1 \langle t_{st} \rangle$). Notice that here we need to generate a larger sample, corresponding to N/α, to have the correct equivalent of N completely unweighted events. Moreover, while the surrogate evaluation and the first unweighting are performed for all the events, the evaluation of the true weight and the second unweighting are performed only for the accepted events by the first unweighting, with $N_1 = N_0^{su} \varepsilon_1$. Finally, if we substitute these two formulas into eq.(3.9) we find

$$f_{eff} = \alpha \, \frac{1}{\frac{\langle t_{nn} \rangle}{\langle t_{st} \rangle} \frac{\varepsilon_{st}}{\varepsilon_1 \varepsilon_2} + \frac{\varepsilon_{st}}{\varepsilon_2}} \quad . \tag{3.12}$$

To have a gain from the surrogate method respect the standard one, we need an effective gain factor larger than 1. For the reasons previously discussed, here we will consider only Kish factor values close to 1. From eq.(3.12) we expect better results when there are small standard efficiency, while for high standard efficiency we can have only small gains. Moreover, to obtain a good results we aim to achieve a fast neural network model, such that $\langle t_{nn} \rangle \ll \langle t_{st} \rangle$, and low number of weight computations, such that $\varepsilon_2 \gg \varepsilon_{st}$.

Looking at the eq.(3.12) we can notice that one condition to have $f_{eff} > 1$ is that the efficiency of the second unweighting must be larger than the standard efficiency, $\varepsilon_2 > \varepsilon_{st}$. In fact, if we consider the opposite case, $\varepsilon_2 < \varepsilon_{st}$, knowing that they produce the same number of events, it would imply that the surrogate method had computed more weights than the standard method, which of course would lead to a negative gain.

In an ideal scenario, we would have $\alpha = 1$, $\varepsilon_1 = \varepsilon_{st}$, $\varepsilon_2 = 1$ and a resulting effective gain factor

$$f_{eff} = \frac{1}{\frac{\langle t_{nn} \rangle}{\langle t_{st} \rangle} + \varepsilon_{st}} \quad , \tag{3.13}$$

that can be considered like an optimistic upper bound for f_{eff}.

To conclude this part, we highlight that effective gain factor refers only to this part of the event generation and does not propagate to the other stages. Thus, it has to be seen as an upper limit of a potential CPU time saving on the whole budget.

Chapter 4

Artificial neural network: basic idea and implementation

Machine learning is a subfield of the artificial intelligence and it aims to improve the performances on some given tasks through the identification of patterns in the data. Contrary to a more traditional (model-based) approach, where the model depends on our knowledge of the process of interest, the Machine Learning (data-driven) approach is to build a model with the information learned from the data. The Machine Learning can learn the relations between the inputs and the desired output during the training phase, in order to make predictions over unknown inputs.

There are different types of Machine Learning algorithms and they are characterized by their training phase:

- supervised learning, where the data provided to the system have one or more parameters and a label. The label is the information that the machine learning has to predict in function of the parameters. The labels correspond the true value of the outputs of the system. The supervised Machine Learning algorithm is mainly used to solve regression problems (with continuum labels) and classification problems (with discrete labels).

- unsupervised learning, where the data provided to the system have only features but no label. So the system is not able to estimate and output quantity. In this case the system can learn patterns and structures inside the data. The unsupervised Machine learning is used, for example, in clustering and feature extraction.

- reinforcement learning, which is an algorithm in between the 2 above algorithms. In this case the data has no label, but after the generation of the output, the system receives feedback on the output from the environment.

In principle, there are also approaches which are somewhat in between of supervised and unsupervised. Usually referred to as semi-supervised or weakly supervised.

In this work, we use only Artificial neural networks in a totally supervised approach. In the next section, we are going to briefly described the Artificial neural networks.

4.1 Artificial neural networks

An Artificial neural network is a computational network inspired to the biological neural networks of the human brain. Similarly to the biological neurons, the nodes of an artificial neural networks receive a signal, elaborate it and propagate it to the others nodes. Moreover, the artificial neurons, like the biological ones, are able to modify the degree of the connections between them depending on their experiences.

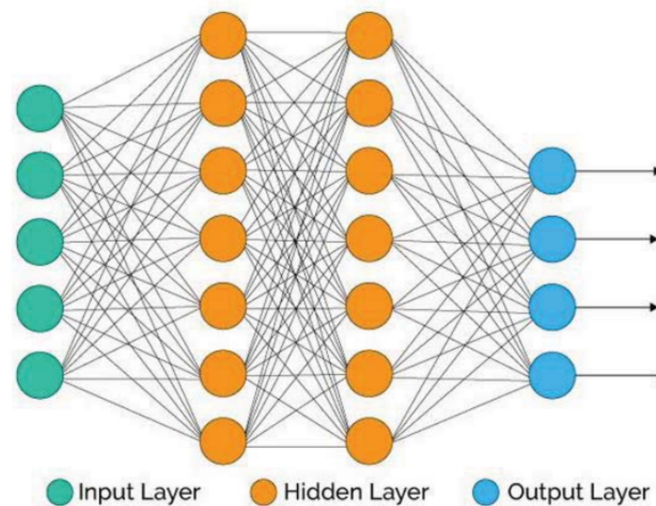


Figure 4.1: Fully connected artificial neural network architecture (Bre, Gimenez, and Fachinotti 2018).

As we can see in fig. (4.1), the nodes are divided in layers, which are of 3 different types:

- input layer, that is the first layer and it contains the information of a given event of the dataset, that will be propagated to the following layer. Each node of the input layer corresponds to a feature of the inputs.
- hidden layer, that is the type of layer which propagate the information from the input to the output. Usually, there are several hidden layers and they work like a black-box, because often we don't know how all the individual neurons work together to arrive at the final output.

- output layer, that is the last layer and its outputs are the predictions of the neural network for the event given as input. Each node of the output layer corresponds to a prediction and its true value is the corresponding label of that event.

In a fully connected neural network, each node is connected to all the nodes of the previous layer and of the following layer, in a web-like connection. The number of nodes and the number of layers in a neural network are free parameters. The same is true also for the sizes of the input and output layer, but they are usually fixed by the number of inputs that we want to pass to the neural network and by the number of outputs that we want to produce. Then, the architecture of the neural network depends on the type of problem that we are studying and on the performance that we want to achieve.

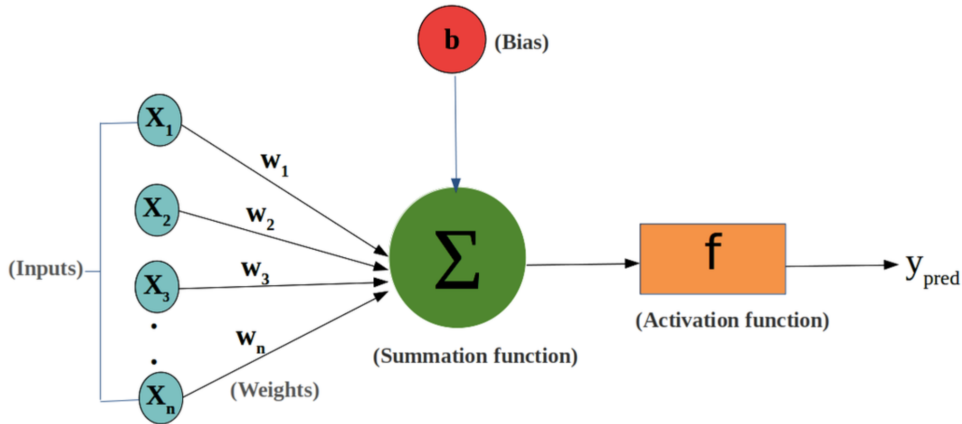


Figure 4.2: Representation of what happens inside a node. Image from [27].

A node is a mathematical function which receive the inputs from the previous layer and produce and an output, fig. (4.2). The output of a node depends only on the inputs provided to it and on the parameters of the node. These parameters are: the weights $w_0, .., w_{n-1}$ and the bias b, where n is the number of inputs $x_0, .., x_{n-1}$ to the node (or the number of nodes in the previous layer). The output y of a node is given by:

$$y_{pred} = f\Big(\sum_{i=1}^{n} x_i \times w_i + b\Big) \quad , \tag{4.1}$$

where f is the activation function of the node. The activation function has the role to introduce non-linearity into the output. Some relevant activation functions are:

- Sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$. It is commonly used in the output layer to predict a probability or for binary classification.

- ReLU function $\sigma(x) = max(0, x)$. It does not activate all the nodes, so it less computationally expensive than others (like the Sigmoid) and it has very simple derivative. It is commonly used in the hidden layers.

- linear function $\sigma(x) = x$. It is used mainly in the output layer, commonly for regression problems.

The process of computing the output of a given input is called forward propagation. It consists into setting the values of the input layers and to use them as inputs to compute the values of the nodes of the first hidden layer. Then the outputs of the first hidden layers become the inputs for the following layer and so on, up to the generation of the output. The data flows from layer to layer until it reaches the output layer.

During the training, the purpose of the neural network is to train the parameters of each node (weights and biases) in order to minimize a given loss function $Loss$, which represents the accuracy of the neural network predictions respect their true value. Some interesting loss functions in our case are:

- mean squared error (mse)

$$Loss = \frac{1}{n}\sum_{i=1}^{n}\Big(y_{pred}(i) - y_{true}(i)\Big)^2 \quad , \tag{4.2}$$

  where n is the number of outputs, $y_{pred}(i)$ is the predicted output by the i-th node of the output layer for a given input and $y_{true}(i)$ is the corresponding true value. It the most common loss function in regression problem.

- mean absolute error (mae)

$$Loss = \frac{1}{n}\sum_{i=1}^{n}|y_{pred}(i) - y_{true}(i)| \tag{4.3}$$

  It is similar to the mse and it can provide better results in case of large errors, but it has unstable solutions and possible multiple solutions.

- Logarithm of the hyperbolic cosine

$$Loss = \sum_{i=1}^{n}\log\Bigg(\cosh\Big(y_{pred}(i) - y_{true}(i)\Big)\Bigg) \quad . \tag{4.4}$$

  This function it is approximately equal to the mean squared error for small differences and to the mean absolute error minus the logarithm of 2 for large differences. So, it has similar properties to the Huber function, but it is twice differentiable and it does not require tuning a parameter.

A possible issue for the neural network training is the presence of outliers events, which are points relatively distant from other points in the multi-dimensional space of features. Outlier events produce a large *Loss* and they can affect also the prediction of other events. It is possible to partially mitigate the effects of these outlier events through the choice of the loss function. The mse loss function tends to strongly penalize predictions with a large error and so, the presence of some rare outlier events could be problematic during the training. On the other hand, a mae loss function is more robust to this type of event. To overcome the disadvantages of these two functions, a possible solution is to use a different loss function, that can have both the advantages of the mse and of the mae , like the logarithm of the hyperbolic cosine.

A method widely used to train the parameters of a neural network is the backward propagation algorithm. It consists into use the gradient of the loss function with respect to any weight or bias to update the value of that parameter. This is done by applying the chain rule to the derivative of the loss function starting from the output layer, up to the layer of that weight or bias. The backward propagation is computed in the opposite direction to the forward propagation.

These processes described until now (forward and backward propagations) are iterated for each input of the training dataset, during the training phase. The parameters of the neural network, are not updated after the forward propagation of each input, but after a given number of samples, that is called batch size. Depending on the batch size of a model, the learning algorithms are separated into three types: stochastic gradient descent, if they have a batch size equal to 1; batch gradient descent, if they have a batch size equal to the number of training samples; mini-batch gradient descent, if they have a batch size between one and the number of training samples.

Usually the model performs several cycles through the full training dataset and the number of these cycles is called the number of epochs. One of the risks during the training is that the neural network learns too many details in the training dataset along with the noise from the training dataset. This causes the neural network to lose the ability to generalize in respect to new data and so, worse results on unseen data. This problem is called overfitting and to avoid it we use a validation dataset. The validation dataset contains events which follow the same probability distribution as the training dataset, but they are independent of it. The parameters of the neural network are not trained over the validation dataset and it is used at the end of each epoch to check that the loss function over the validation dataset will not increase (that could be a sign of overfitting). To prevent this, TensorFlow [28] has a functionality called Early Stopping Callback, which stops the training after that validation loss stops to decrease for a given number of epochs. The number of maximum epochs before that the Early Stopping Callback interrupts the training is called patience. The validation dataset can be used to optimize the hyper-parameters of the neural network, which are non optimized during the training phase, like the number of hidden layers and of nodes, the learning rate and the batch size. It is suggested to use also a third dataset, called test dataset, to assess the

final model that is chosen using the validation dataset and to avoid correlations which could be present with the validation dataset.

4.2 neural network surrogate

We have seen in section (3.1) that we can compute the cross-section of a $2 \to n$ partonic scattering like eq. (2.31), using Monte Carlo integration techniques. Now we want to train a neural network such that our model will be able to well predict the complete weight w of each event. Predicting the complete weight including the product between the matrix element and the Jacobian factor has the advantage to be a smoother function over the phase-space than the single functions.

In order to have an improvement in the unweighting performing features, we look for a neural network which is faster than the standard approach ($\langle t_{nn} \rangle \ll \langle t_{st} \rangle$) and flexible enough to achieve good predictions also for high-multiplicity scattering events. For these reasons, we chose a fully connected neural networks.

We applied the surrogate unweighting algorithm to two different processes, which are: $pp \to t\bar{t}$, as a first study of a relative simple process, and $pp \to e^- e^+ g g d \bar{d}$, as a more complex process with higher multiplicity.

4.2.1 Implementation on $pp \to t\bar{t}$ process

For simplicity, in the first process we studied only one of the MadGraph channels of integration. We studied the integration channel representing the s-channel of the scattering mediated by gluons. To not loose physical information about the process, the minimum number of inputs that we can use is equal to the number of the degrees of freedom. The formula for the degrees of freedom of a $2 \to n$ scattering with initial protons is given by:

$$d = 4n - n - 4 + 2 = 3n - 2 \ , \tag{4.5}$$

where the term $+4n$ corresponds to the four momenta of the final particles, the $-n$ term to the mass of the final particles which are on-shell, the -4 term to overal 4-momentum conservation and the term $+2$ to the parton distribution functions of the initial protons. Using the eq. (4.5) we obtain 4 degrees of freedom for the considered $2 \to 2$ process. Here, we used 6 inputs, in order to facilitate the learning of the neural network of the relevant physics information. These inputs are: the four momentum of the top quark and the parallel momentum and the energy of the anti-top quark

$$\frac{1}{\sqrt{S}}(p_x^t, p_y^t, p_z^t, E^t, p_z^{\bar{t}}, E^{\bar{t}}) \quad , \tag{4.6}$$

Each momentum is in the laboratory frame and it is normalized by the energy of the center of mass of the protons $\sqrt{S}$.

The output of the neural network is directly the weight of the event, w. Usually, the predicted output of a neural network are normalized between 0 and 1 or they are normalized with mean 0 and standard deviation 1. In this first part, we postponed the study of the normalization of the output for the other process. Using the inputs described above, we observed better results predicting directly the weights respect the absolute value of the natural logarithm of the weight, $y = |\log(w)|$. We underline that this result is due to the absence of the output normalization. In fact, for this process the weights are of the order of about 10^{-3} and it could be problematic for the neural network to predict values much smaller than 1. However, the inputs are normalized in order to be about of the same order of the weights, providing good results and avoiding output normalization.

The prediction of the weights of the events belongs to the regression problems, thus the activation function for the hidden layers that we used is the ReLU activation function and the linear activation function for the output layer. The weights are initialized with the heuristic initialization [29] and the optimizer is Adam [30]. Here, we tested only the mean squared error as loss function. We used a batch size of 1000 events, a learning rate equal to 10^{-3} and we trained the neural networks with epochs containing all training points in random order. The maximum number of epochs has been fixed to 1000 and the patience for the training to 30. The patience corresponds to the maximum number of epochs which the neural network can have without the decrease of the validation loss before the early stopping is activated and the best model (the one with the lowest validation loss) is saved as the final model.

The layers architectures that we tested were composed by 2, 3 or 4 layers, each one with the same number of nodes which was 16, 32 or 64. As expected, we observed an improvement of performance with the increase of layers and of nodes, but also an increase in the required time for the prediction, which almost flatten the increase in effective gain factor. For this reason, we chose an architecture with 3 hidden layers and 64 nodes each one.

NN hyper-parameter	value
hidden layers	3
nodes per layer	64
activation function	Relu
output activation function	linear
loss function	mse
optimizer	ADAM
learning rate	0.001
batch size	1000

Table 4.1: Summary of the hyper-parameters used in the neural network model for the $pp \to t\bar{t}$ process.

4.2.2 Implementation on $pp \to e^- e^+ g g d \bar{d}$ process

For the second process, we studied the MadGraph integration channel corresponding to the Feynman diagram represented in fig. (4.3). The choice of this channel is due to

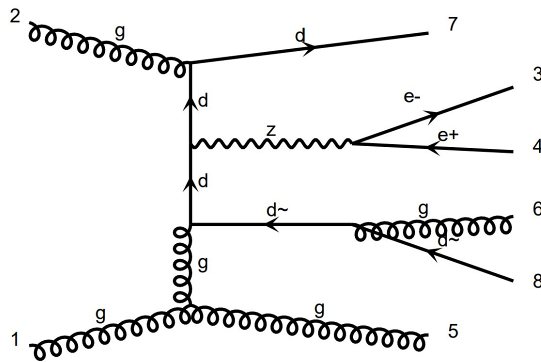


Figure 4.3: Feynman diagram corresponding to the integration channel G128. Image generated by MadGraph.

the fact that it is one of the most relevant channels for this process, due to its large cross-section respect to the other channels, and that it has a good accuracy on the cross-section.

Using the formula eq. (4.5), we can see that for this process the number of degrees of freedom is 16. However, we can decrease by 1 this number using the symmetry of the final state in respect to the transverse angle to the beam axis and applying a rotation on that angle. In this way, requiring that, for example, the transverse angle of the produced down quark is null for each scattering, we obtain 15 degrees of freedom. Alternatively we could take advantage of this symmetry, generating more events which are copies of the initial ones except for a rotation on this angle. This method, called data augmentation, would increase the dataset up to a factor $K + 1$, where K is the number of rotations applied, and would let the model learn the symmetry directly from the dataset. Here we choose to use this symmetry with the first suggested method in order to reduce the number of inputs by 1. This choice is justified by the fact that a larger number of inputs implies a larger dimension of the phase-space of the input variables. The same dataset of events, used with a larger number of inputs, results into a more sparse dataset in the phase-space of the inputs.

In this case, we used 16 inputs and we tested different normalisations of the inputs. Here we described only the one that had better results during the test:

- $\sqrt{\hat{S}}/\sqrt{S}$, where $\sqrt{\hat{S}}$ is the energy of the center of mass of the interaction and $\sqrt{S}$ is the energy of the center of mass of the protons
- β_{cm}, that is the relativistic beta factor of the center of mass in the laboratory frame

- $p_T^{e^-}/\sqrt{\hat{S}}, \theta^{e^-}, \phi^{e^-}$
- $p_T^{e^+}/\sqrt{\hat{S}}, \theta^{e^+}, \phi^{e^+}$
- $p_T^{g^1}/\sqrt{\hat{S}}, \theta^{g^1}, \phi^{g^1}$
- $p_T^{g^2}/\sqrt{\hat{S}}, \theta^{g^2}, \phi^{g^2}$
- $p_T^{d}/\sqrt{\hat{S}}, \theta^{d}$

where p_T is the transverse momentum of the particle, while θ and ϕ are respectively the angles in the parallel and transverse plane to the beam axis in the center of mass frame. For the outputs we tested 4 different normalisations, which were: the normalisation between 0 and 1 of the weights or the value of the logarithm of the weights, and the normalisation with mean 0 and standard deviation 1 of the weights or of the absolute value of the logarithm of the weights. In this case we had different performance for each output normalisation depending on the architecture of the neural network and the loss function used. In general, we observed slightly better prediction when we normalized the output with mean 0 and standard deviation 1 while predicting the logarithm of the weight. However, we observed the presence of a small negative bias in the prediction of the logarithm of weight, as shown in fig. (4.4), and we have not been able to cancel it. Anyway, the problem of this bias should not be problematic for the algorithm thanks to the second unweighting step, which accounts for the error committed by the neural network.

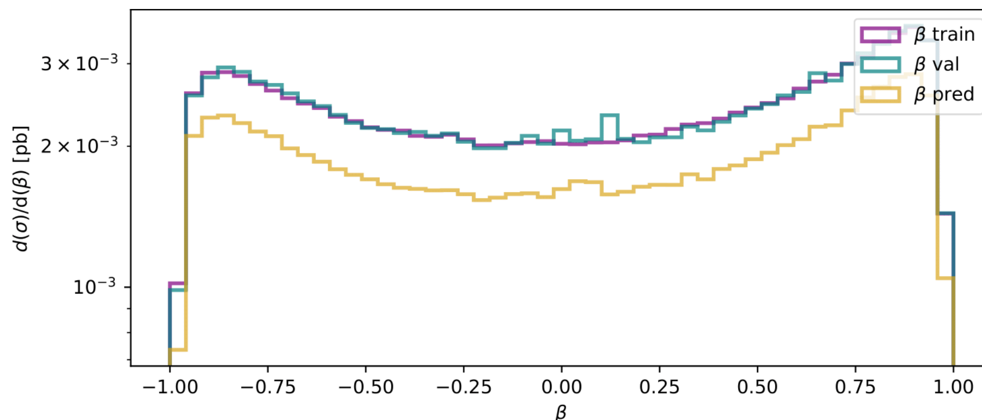


Figure 4.4: Weights distributions of the training, test and predicted datasets respect the β factor of the center of mass in the laboratory frame, for $pp \to e^-e^+ggd\bar{d}$ process. The training dataset has been normalized to be equal to the test dataset.

Similarly to the previous process, we used the ReLU activation function for the hidden layers, the weights are initialized with the heuristic initialization and the optimizer is

Adam. For the training, we used a patience equal to 30, as before. During the process of model selection of the neural network, we tested different batch sizes, learning rates and architecture. In the end, we chose the parameters which produced a lower validation loss, which correspond to a batch size equal to 500 and a learning rate equal to 10^{-4}. We started from an architecture composed by 4 layers with 64 nodes each one and then tested mainly other 2 architecture composed respectively by: 4 layers with 128 nodes each one and 8 layers with 64 nodes each one. After several tests we observed that, compared to the initial architecture, the last two architecture had in general slightly better results with a lower validation loss. However, the architecture with 8 layers required almost a double time for the prediction respect the architecture with 4 layers. For these reasons, we selected the architecture with 4 hidden layers and 128 nodes each one. About the hyper-parameters, we tested models with learning rates equal to $10^{-3}, 10^{-4}, 10^{-5}$ and batch sizes equal to $64, 128, 256, 500, 1000$. From our tests, we observed the best validation loss with a learning rate equal to 10^{-4} and an improvement in the validation loss with the increase of the batch size up to 500, with a similar or slightly worse value for 1000.

A possible problem for a more complex problem could be the presence of outlier or rare events, which can be difficult to learn for the neural network. In order to study whether these type of events are present and affect the training or not, we tested different loss functions. In particular, we tested as loss function for this process, the mse, the logarithm of the hyperbolic cosine and the Huber function, which behaves as quadratic loss error and linear for large error. However, we did not observe particular differences between the predictions obtained with these loss functions, so we preferred to use the mse.

NN hyper-parameter	value
hidden layers	4
nodes per layer	128
activation function	Relu
output activation function	linear
loss function	mse
optimizer	ADAM
learning rate	0.0001
batch size	500

Table 4.2: Summary of the hyper-parameters used in the neural network model for the $pp \to e^- e^+ g g d \bar{d}$ process.

Chapter 5

Results

In the following chapter we will describe the results obtained studying two different processes with the surrogate unweighting method describe in the section (3.2) and the neural networks described in section (4.2).

We produced weighted events through MadEvent, bypass the unweighting algorithm. Each event is characterized by the four-momenta of each initial and final particles and its weight. Instead of the MadEvent unweighting, we will use the neural network as a light-weight surrogate to perform a first unweighting and, then, compute the true weights of the remaining events to perform a second unweighting in order to compensate for the possible mismatches in the predictions.

5.1 Results on $pp \to t\bar{t}$

As a first test, we studied the scattering process $pp \to t\bar{t}$, which is a relatively simple process. MadGraph has already good results with a fast evaluation of the matrix element and a good unweighting efficiency. In particular, in the s-channel of the scattering mediated by gluons, the average time to perform the unweighting of one event is $< t_{MG} >= 0.00171s$ and the efficiency is about $\varepsilon_{st} = 10\%$. $< t_{MG} >$ has been measured unweighting a sample of 4000 events, however, we observed that the average unweighting time decreases with the time increasing of the sample size to unweight. The surrogate method has a good performance in this case, but it could give more benefits for more complex processes where the time spent in the unweighting is much longer. For this reason, we used this process only as a first example without taking care of the tuning details to improve the surrogate unweighting, as we have done for the second process.

We used a dataset composed by 12000 events for the training and 4000 events for the validation of the neural network. The simplicity of the process, as we can see from the weights distribution of the events, fig. (5.1), allowed us to use a relatively small dataset, even tough it has some statistical fluctuations. Using a small dataset allowed us to see

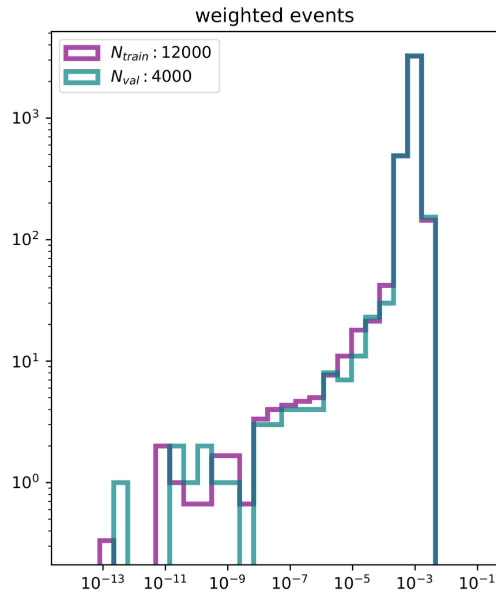


Figure 5.1: Weights distributions of the training and validation datasets generated by MadGraph for the s-channel of the process $pp \rightarrow e^- e^+ g g d \bar{d}$ at $\sqrt{s} = 13 TeV$. The training dataset has been normalized to be equal to the validation dataset.

that for a relatively simple process, the neural network can be trained with dataset of about the same order of the samples of events that usually MadGraph has to unweight.

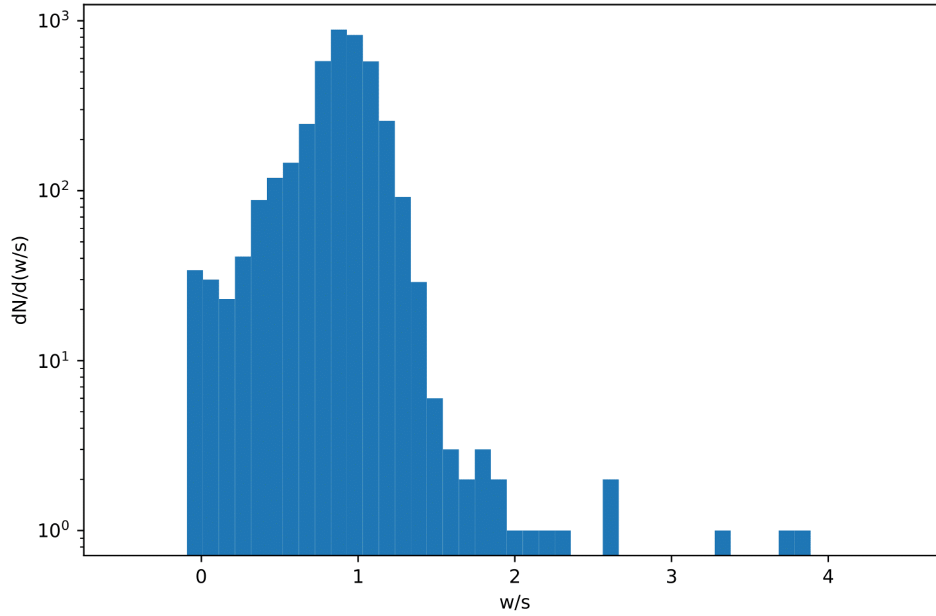
In the histograms of fig. (5.2) we can observe that the predictions are quite in agreement with the true weights having an almost narrow peaks in correspondence of $w/s = 1$. From the left most bin, we see the presence of some events (about 10) with a $w/s \lesssim 0.1$, while only few events have $w/s \gtrsim 2$. In the 2-d histograms of fig. (5.2) we can notice these behaviours with the peak of events centered around $w/s \simeq 1$ and the tail for small weights with an overestimation of the predictions.

In order to improve the precision for small weights and reduce the left-tail of the distribution we tried to use a chi square loss function, but the resulting neural network was too much sensible to small events, losing accuracy for the peak of events.

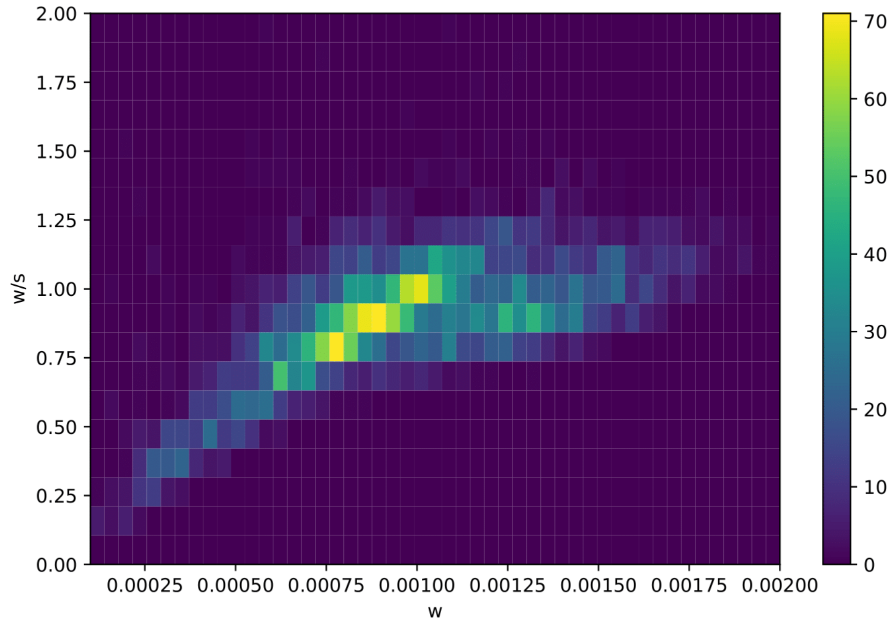
The histograms of fig. (5.2) is useful to understand the performances of the unweighting efficiencies. In fact an overestimation of the large weights can lead to the definition of a very large s_{thres}, that is the value respect which the first unweighting is performed. Such maximum value would correspond to a lowered first efficiency ε_1. It is a similar case for the second unweighting, that is performed over x_{thres}, where $x = w/s$. A long tale to the right of the peak in the histogram (5.2a), that is produced by a large underestimation of the weights (independently of their order of magnitude), leads to a very large x_{thres} and so, to a lowered second efficiency. In the opposite case, a populated tail to the left of the peak in the same plot, would imply the presence of a large number of events with a small x value respect x_{thres}, that is usually larger than 1. Thus, these events would have a small probability to be accepted in the second unweigthing, contributing to a lowered second efficiency. For these reasons, it is important to obtain a narrow distribution centered around 1 in the histograms (5.2).

The fig. (5.3, 5.4) shows the first and second unweighting efficiency of the model discussed before. The 2 efficiencies have good performances with values between about 60% and 80% depending on the percentage of allowed overweight. Combining the two efficiencies, we obtain the total efficiency that is, on average, larger than the MadGraph one for this process (that is about 10%). Moving from the most left point of a curve to the right points, we can notice that with the decreasing of the allowed overweight, the maximum increases and the efficiency decreases, as expected from the acceptance-rejection sampling method.

Each first unweighting is performed separately over the same initial sample. Starting from the same sample, we produce a sample for each overweight parameter r_{s}. These parameters are used to find the corresponding s_{thres}, which is used to perform the unweighting as described by the algorithm described in the section (3.2). The parameters selected for this first test are $r_{\text{s}} = 20\%, 10\%, 5\%, 1\%$. At the end of this step, we generate $N_{r_{\text{s}}} = 4$ partially unweighted samples, where $N_{r_{\text{s}}}$ is the number of r_{s} parameters to test. Then, for each partially unweighted sample, the second unweighting is performed separately for each overweight parameter r_{x}, producing $N_{r_{\text{s}}} \cdot N_{r_{\text{x}}}$ unweighted samples. For the second unweighting, we chose the overweight parameters r_{x} equal to r_{s}, so we



(a)



(b)

Figure 5.2: Distribution of the surrogate weights s respect the true weights w for the $pp \to t\bar{t}$ process. Fig. (5.2a) represents a one-dimensional histogram of the ratio $x = w/s$. Fig. (5.2b) represents a two-dimensional histogram between the ratio $x = w/s$ and w.

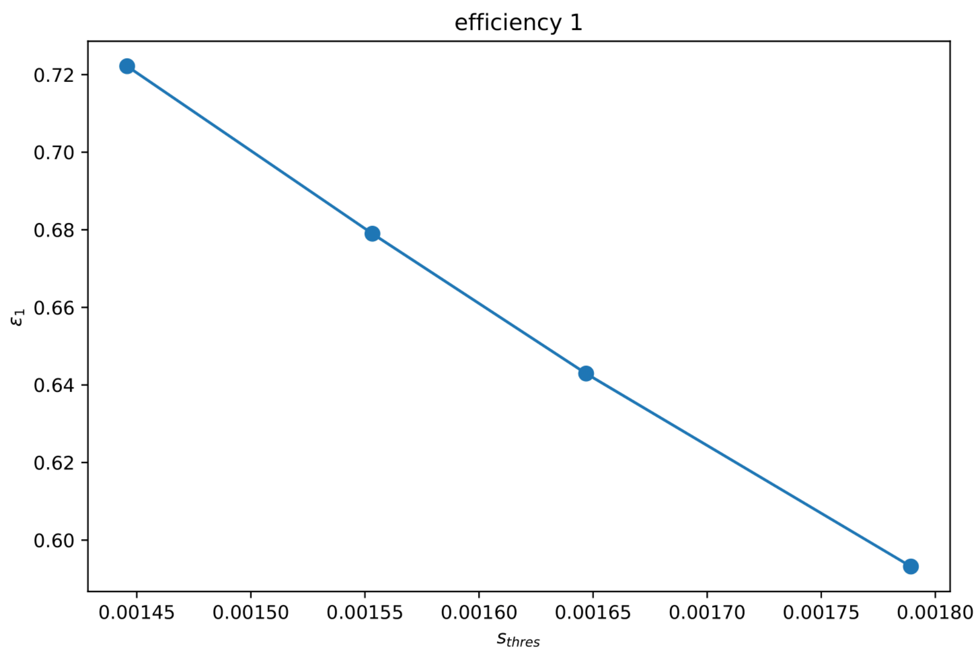


Figure 5.3: Efficiencies of the first unweighting respect the maximum used to compute the unweighting for the $pp \to t\bar{t}$ process. Each point corresponds to an unweighting performed respect a different overweight parameter r_s. From left to right, each point corresponds respectively to $20\%, 10\%, 5\%, 1\%$ of allowed overweight in the first unweighting.

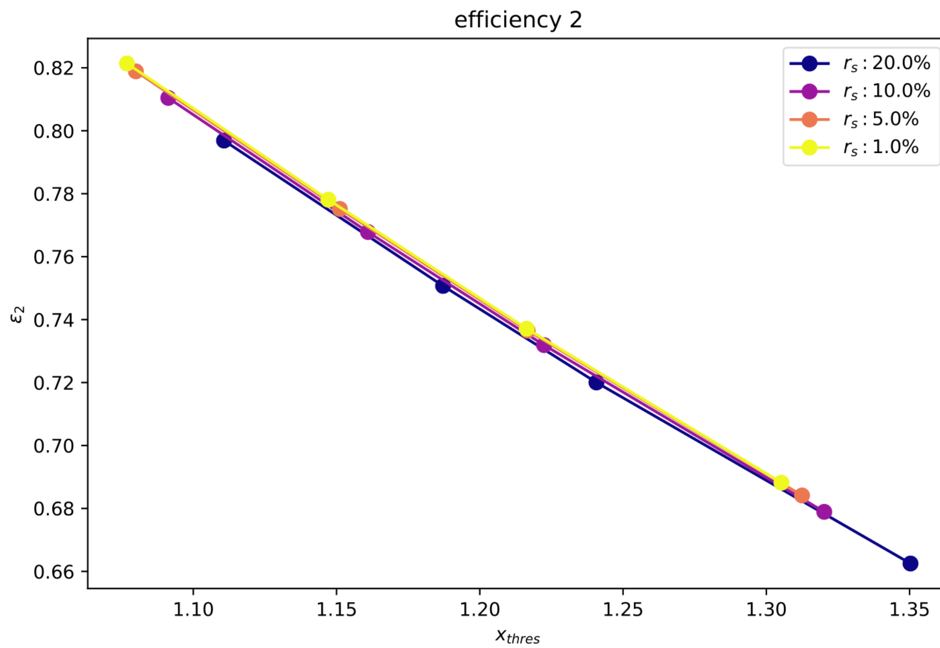


Figure 5.4: Efficiencies of the second unweighting respect the maximum used to compute the unweighting for the $pp \to t\bar{t}$ process. Each line corresponds to different unweightings performed over the same dataset, which is produced by a given r_s parameter. From left to right, each point corresponds respectively to $20\%, 10\%, 5\%, 1\%$ of allowed overweight in the second unweighting.

generate $4 \cdot 4 = 16$ unweighted samples.

In the fig. (5.4) we can observe that the maxima x_{thres} corresponding to the same value of r_{x}, are not perfectly coincident, but they are slightly shifted (especially in the case of $r_{\text{s}} = 20\%$). This is due to the fact that the maxima x_{thres} are computed after the first unweighting and, so, the samples will depend also on r_{s}. Thus, the value of x_{thres} and ε_2 slightly depend also on the parameter r_{s}. However, even if in this case those differences are small, in the case of more complex processes those differences can be relevant, in particularly, for small overweights parameters r_{x} when the maxima are more sensible to rare large outliers.

The choice of the r_{s}, r_{x} parameters determines the efficiencies ε_1, ε_2, but also the number and the magnitude of the overweighted events. The distribution of overweighted events defines the Kish factor, as described in eq. (3.2). The fig. (5.5) shows the distribution of the correction weights $\tilde{w}$ of the unweighted events respect the parameters r_{s} and r_{x}. N_0^{ow} represents the number of accepted events with no overweight in the first and in the second unweighting ($\tilde{w}^{(1)} = 1$ and $\tilde{w}^{(2)} = 1$); N_1^{ow} represents the number of accepted events with overweight only in the first unweighting ($\tilde{w}^{(1)} > 1$ and $\tilde{w}^{(2)} = 1$); N_2^{ow} represents the number of accepted events with overweight only in the second unweighting ($\tilde{w}^{(1)} = 1$ and $\tilde{w}^{(2)} > 1$); N_{12}^{ow} represents the number of accepted events with overweight in both the first and the second unweighting ($\tilde{w}^{(1)} > 1$ and $\tilde{w}^{(2)} > 1$).

From these plots it is possible to observe that, for this process, the distribution of the overweighted $\tilde{w}^{(2)}$ depends almost only on the overweight parameter of the second unweighting r_{x}. The overweight in the first unweighting is determined by the parameter r_{s} and it is almost completely reabsorbed in the second unweighting. We can see this behavior from the fact that the number of events with an overweight in both the unweightings is much smaller than the number of of events with an overweight only in the first unweighting and not in the second one. Thanks to this fact, also the Kish factor is mainly determined by r_{x} parameter, allowing us to have a larger overweight in the first unweighting in respect to the second unweighting, in order to have a larger ε_1 and a Kish factor α closer to 1. Notice that the number of unit-weight in the final sample is given by $N_0^{ow} + N_1^{ow}$, while the number of final overweighted events is given by $N_2^{ow} + N_{12}^{ow}$.

To evaluate the overall performance of the surrogate method we use the effective gain factor to estimate the gain of time in respect to the classical method. The fig. (5.6) illustrate the plots of the effective gain factor of the model discussed before. The values contained in these plots are computed through the effective gain factor formula (3.12) defined in the section (3.2.2). The first thing that we notice is that the surrogate unweighting method has better result for larger allowed overweight r_{x}, as expected. Even tough the Kish factor is smaller for larger allowed overweight r_{x}, the gain in efficiency is dominant and produces a larger effective gain factor. In agreement with the almost non-dependence of ε_2 and α on r_{s}, also, the effective gain factor is weakly sensible to the allowed overweight in the first unweighting. We underline that for all the combinations of r_{s} and r_{x} tested in this process, the surrogate unweighting method has always better

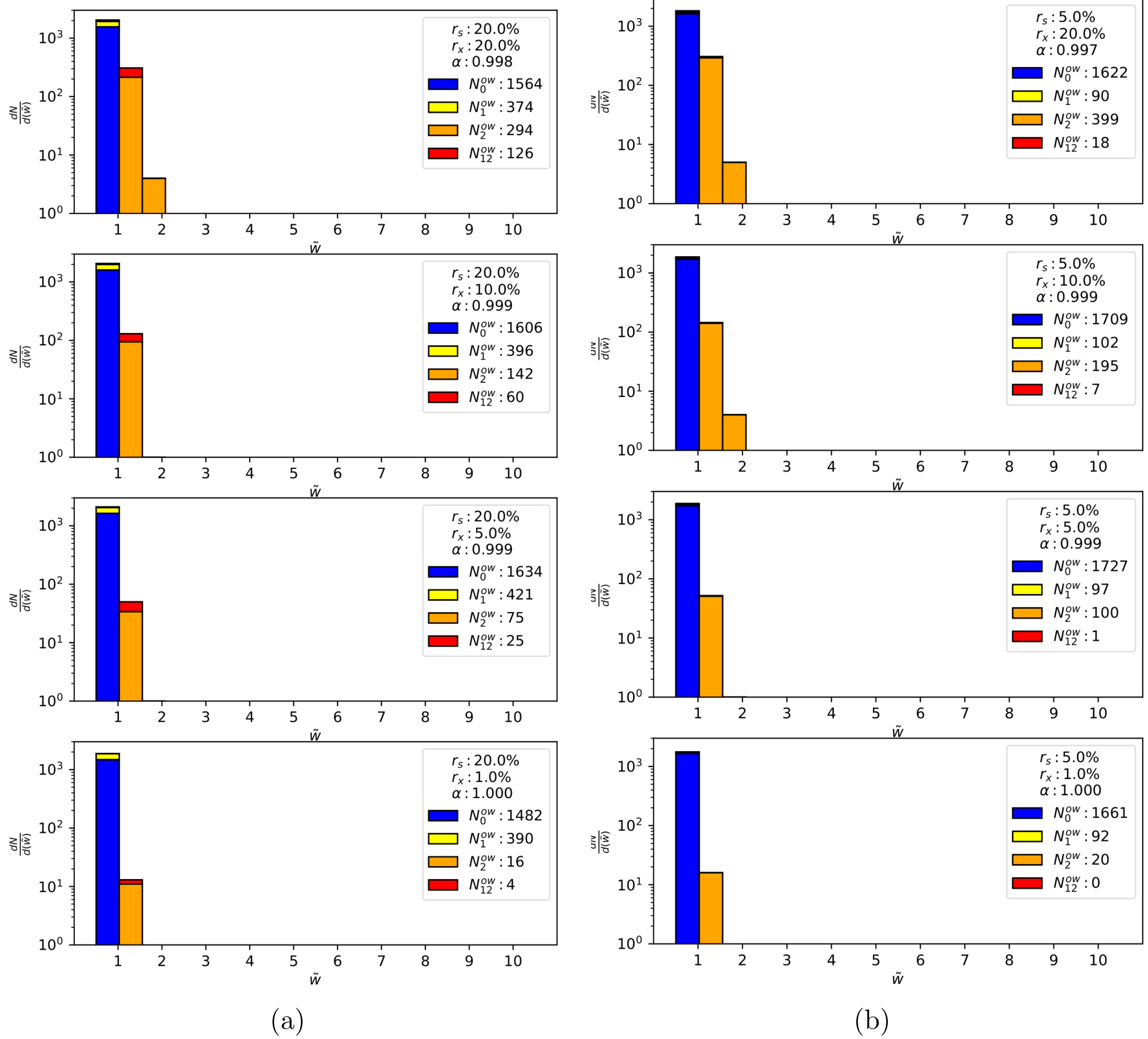


Figure 5.5: Comparison of the final event weights $\tilde{w}$ obtained by the unweighting of 4000 weighted events for the $pp \to t\bar{t}$ process. Fig. (5.5a) represents the unweightings with different r_x parameters respect the same $r_\text{s} = 20\%$ parameter. Fig. (5.5b) represents the unweightings with different r_x parameters respect the same $r_\text{s} = 5\%$ parameter.

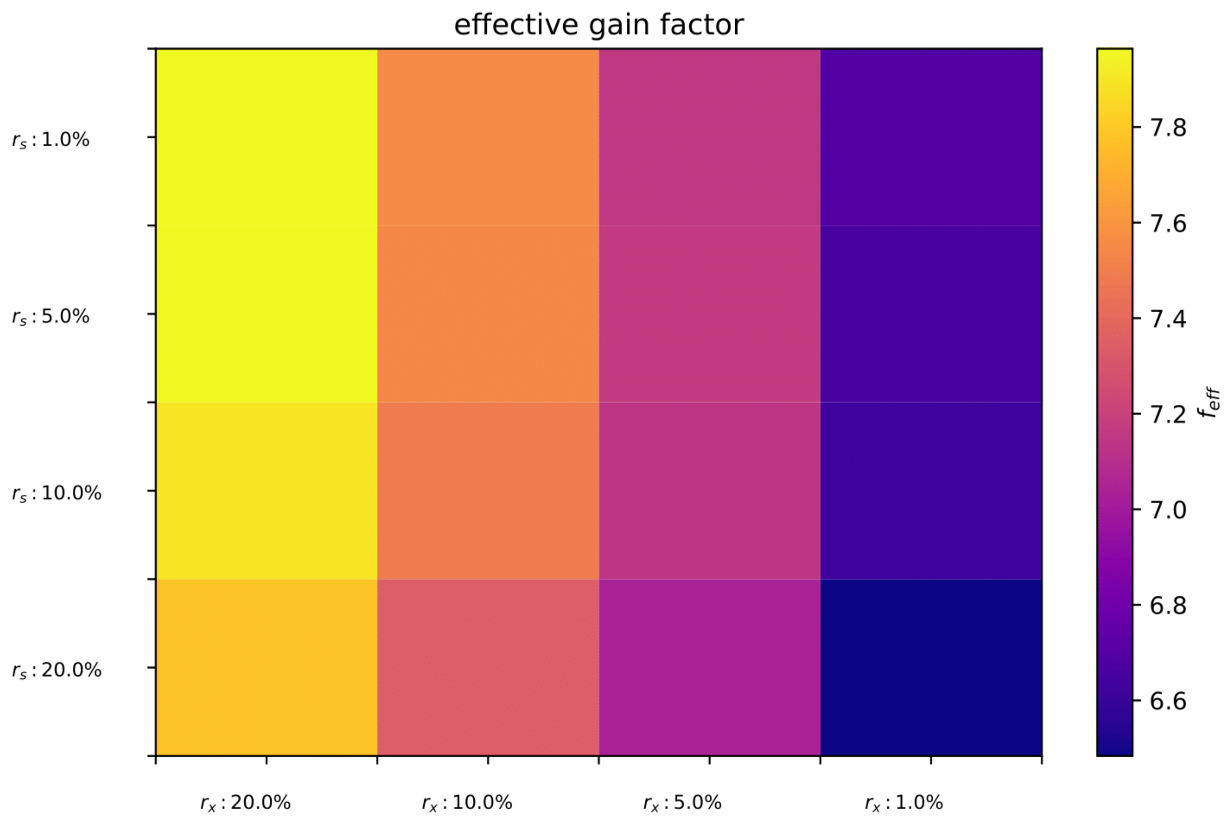


Figure 5.6: Effective gain factors achieved by different combinations of the r_s and r_x parameters for the $pp \to t\bar{t}$ process. On the y-axis there is the allowed overweight in the first unweighting r_s and on the x-axis the allowed overweight in the second unweighting r_x. The magnitude of the effective gain factor is shown by the color-bar in the right.

performances than the standard unweighting method. This is due to the high efficiencies ε_1 and ε_2, which are always larger than standard efficiency ε_{st}.

The average time to unweight an event for the surrogate method, $\langle t_{nn} \rangle$ is defined as:

$$\langle t_{nn} \rangle = \frac{t_{init} + t_{pred} + t_{unwgt1} + t_{unwgt2}}{N} \quad , \tag{5.1}$$

where N is the initial number of weighted events, t_{init} is the time to initialize the inputs of the sample for the neural network, t_{unwgt1} is the time to perform the first unweighting and t_{unwgt2} is the time to perform the second unweighting. For this process, we measured $\langle t_{nn} \rangle = 0.000043s$. Notice that $\langle t_{nn} \rangle$ slightly depends on the overweight parameters $r_{\mathrm{s}}, r_{\mathrm{x}}$, due to the smaller size of the samples to unweight with the decreasing of the allowed overweights $r_{\mathrm{s}}, r_{\mathrm{x}}$ (and so with the decreasing of the efficiencies). However, we preferred to use a fixed time ratio in order to avoid statistical fluctuations in the unweighting time and neglecting those small differences of the order of $\lesssim 1\%$. Compared to the standard unweighting, the surrogate method has an unweighting average time per even about $\langle t_{MG} \rangle / \langle t_{nn} \rangle \simeq 40$ times faster for this process. For more complex processes we expect smaller surrogate efficiencies, but a larger gain in time in respect to the standard method.

In order to show the results independently of the ratio $\langle t_{nn} \rangle / \langle t_{st} \rangle$, which can be more easily improved in respect to the efficiencies, we define the plot in fig. (5.8). This plot represents the minimum time performances, $\langle t_{st} \rangle / \langle t_{nn} \rangle$, required to have a positive effective gain factor. In particularly, the dots represent the time ratio required to obtain a null gain, $f_{eff} = 1$, and the crosses represent the time ratio to obtain a large gain, $f_{eff} = 5$. This means that each time performance above the corresponding dot (or cross) has an effective gain factor larger than 1 (or larger than 5). Thus, if we look to two dots, the lower one requires a lower time performance to achieve the same result.

We notice that, also time ratio lower 1 are allowed, which means that the surrogate efficiencies are so better respect the standard efficiency that also for $\langle t_{st} \rangle < \langle t_{nn} \rangle$, we could have a positive effective gain factor. Moreover, each point has a color that indicates the Kish factor for those parameters. This allows us to select the best models in respect the efficiencies performances and the Kish factor in the same plot. We can see that, in agreement with the previous plots, both the efficiencies performance and the Kish factor depend mainly on the parameter r_{x} of the allowed overweight in the second unweighting.

5.2 Results on $e^- e^+ g g d \bar{d}$

The second process that we studied is the scattering $pp \to e^- e^+ g g d \bar{d}$ mediated by two gluons. For the generation of the events we used the default phase-space cuts of MadGraph. We used the integration channel corresponding to the Feynman diagram in fig. (4.3) to optimize the neural network performance in terms of timing and accuracy.

r_s r_x	ε_1 ε_2	α	f_{eff}
20.0% 20.0%	0.722 0.797	0.998	7.783
20.0% 10.0%	0.722 0.751	0.999	7.341
20.0% 5.0%	0.722 0.720	0.999	7.044
20.0% 1.0%	0.722 0.663	1.000	6.484
10.0% 20.0%	0.679 0.810	0.997	7.897
10.0% 10.0%	0.679 0.768	0.999	7.491
10.0% 5.0%	0.679 0.732	0.999	7.145
10.0% 1.0%	0.679 0.679	1.000	6.630
5.0% 20.0%	0.643 0.819	0.997	7.961
5.0% 10.0%	0.643 0.775	0.999	7.547
5.0% 5.0%	0.643 0.736	0.999	7.175
5.0% 1.0%	0.643 0.684	1.000	6.667
1.0% 20.0%	0.593 0.821	0.998	7.964
1.0% 10.0%	0.593 0.778	0.999	7.554
1.0% 5.0%	0.593 0.737	0.999	7.160
1.0% 1.0%	0.593 0.688	1.000	6.687

Figure 5.7: Table containing the results of the unweighting in respect to all the combination of the overweight parameters r_s, r_x for the $pp \to t\bar{t}$ process.

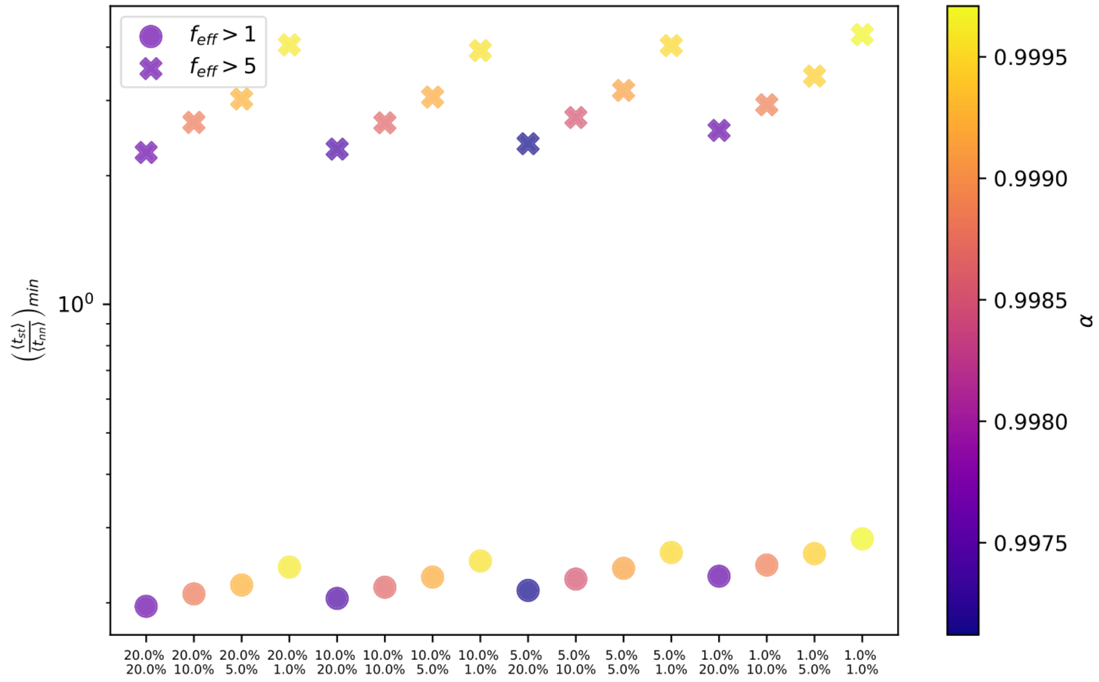


Figure 5.8: Comparison of the minimum ratio $\langle t_{st}\rangle/\langle t_{nn}\rangle$ between the ratios of the standard method and the surrogate method to achieve a given effective gain factor for the $pp \to t\bar{t}$ process. Each column (composed by a vertical pair of a dot and a cross) corresponds to a given combination between r_{s} (above) and r_{x} (below). The color of each combination represents the Kish factor α associated to the corresponding final events sample of that combination.

The higher multiplicity of this process, respect to the previous one, implies a longer evaluation of the cross-section and a more difficult unweighting. These additional difficulties cause a longer average time to unweight an event and a lower unweighting efficiency. In particular, for this channel, we measured an average time to unweight an event $\langle t_{st} \rangle = 0.00248s$ and an unweighting efficiency $\varepsilon_{st} = 1.4\%$. These values has been measured unweighting a sample of 16000 events, which is larger than the sample of the previous process (4000) due to the fact that usually for more complex processes MadGraph requires a larger sample to achieve a good accuracy in the cross-section evaluation. As before, we observed a decrease of the average unweighting time per event with the increase of the sample size to unweight.

As we said before, the $t\bar{t}$ process is relatively simple and also for the neural network is relatively simple to learn the prediction of the weights. This allowed us to use a quite small dataset to train the neural network with only 16000 events. Due to the higher complexity of this process and to the larger number of inputs, the neural network required a much larger dataset for the training respect to the previous process. For the training and the validation of the neural network we used 2 datasets composed respectively by 5000000 and 2000000 weighted events. For this process we used also a test dataset of 1000000 events and the results showed below are the results obtained over this test dataset. In fig. (5.9) it is shown the distributions of the training and validation datasets described above. We can notice that the weights span several order of magnitude between them, up to 25.

In the histograms of fig. (5.10) we can observe the distributions of the ratio of the true weights respect the predictions w/s. They present a peak centered around $w/s = 1$, showing that about half of the events have a precise prediction with $s \simeq w$. However, fig. (5.10a) shows quite long tails for the distribution of w/s, reaching respectively about 10^{-7} and 10^{3} for the predictions with the largest errors. In 2-d histogram of fig. (5.10b) we observe that the peak of events corresponds to an almost flat region centered around $w/s = 1$, in agreement with the 1-d histogram. We can notice also that the neural network achieved the best accuracy for the most populated region of training sample, $w \sim 10^{-7}$, as shown in fig. (5.9). In the less populated region, we can observe that the predictions tend to overestimate the events with $w \lesssim 10^{-7}$, while they tend to underestimate the events with $w \gtrsim 10^{-7}$.

The fig. (5.11) the distributions of the partially unweighted samples considering also their overweight $\tilde{w}^{(1)}$. These samples are obtained from the first unweighting, using different allowed overweight parameters r_{s}. In particular, we can notice that decreasing r_{s}, also the number of accepted events decreases, due to the increase of the maximum s_{thres}. These samples produce different maxima x_{thres} for the second unweighting, which so slightly affects also the efficiency of the second unweighting.

In the fig. (5.12, 5.13), the efficiencies of the first and second unweighting are represented. The efficiencies for the first unweighting are included between about 2% and 20%, while for the second one are included between 1% and 9%.

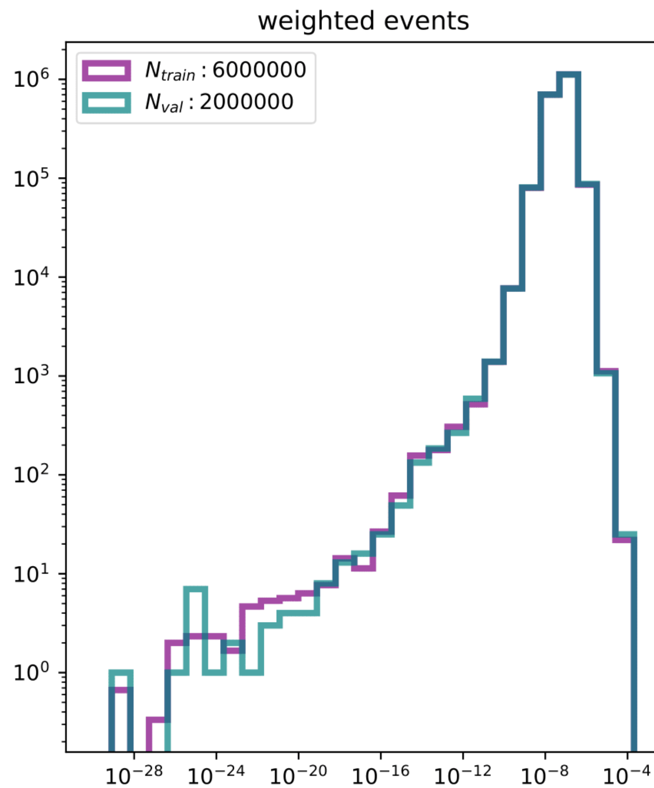
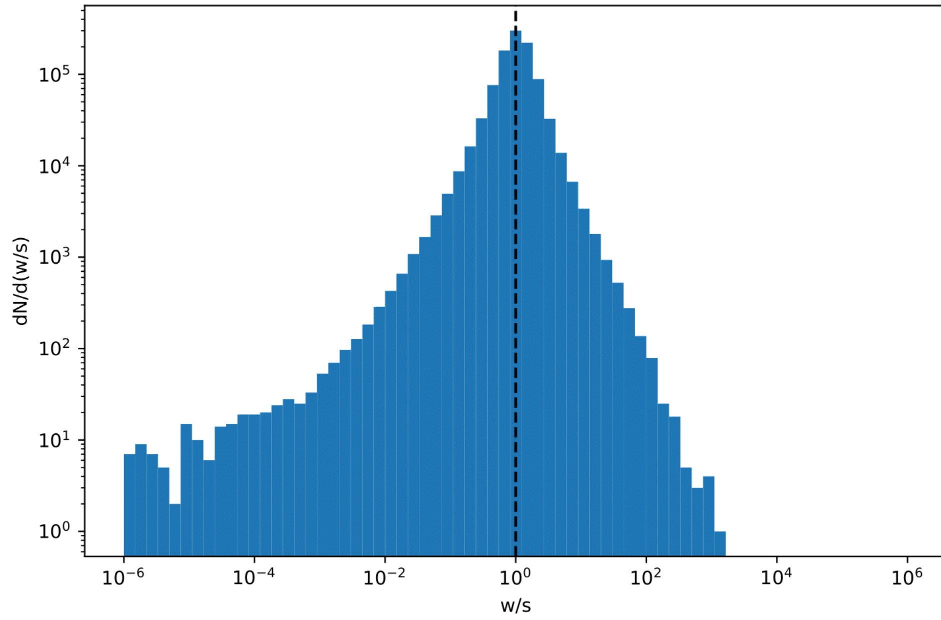
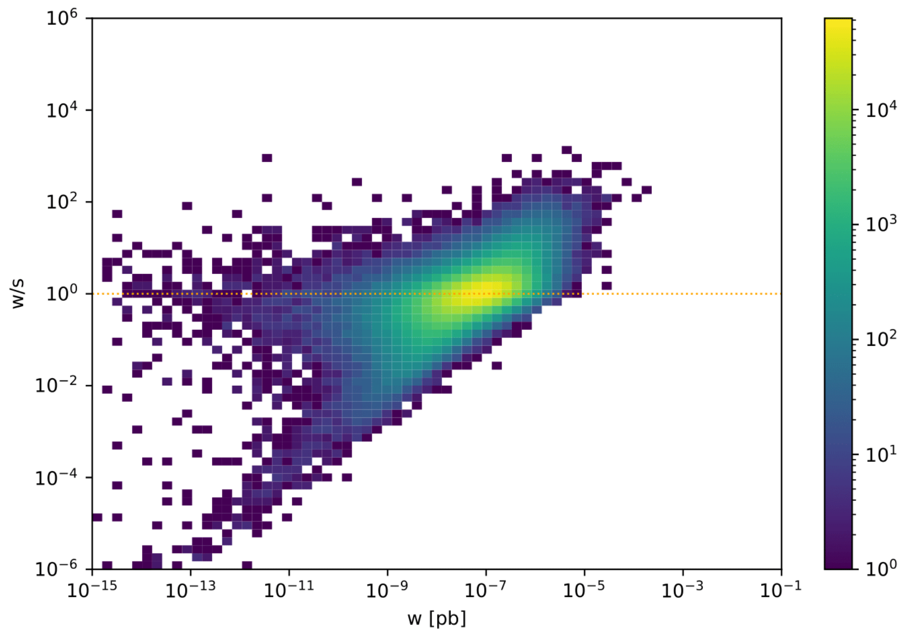


Figure 5.9: Weights distributions of the training and validation datasets generated by MadGraph for the channel $G128$ of the process $pp \rightarrow e^- e^+ g g d \bar{d}$ at $\sqrt{s} = 13TeV$. The training dataset has been normalized to be equal to the validation dataset.



(a)



(b)

Figure 5.10: Distribution of the surrogate weights s respect the true weights w for the $pp \to e^- e^+ g g d \bar{d}$ process. Fig. (5.10a) represents a one-dimensional histogram of the ratio $x = w/s$. Fig. (5.10b) represents a two-dimensional histogram between the ratio $x = w/s$ and w.

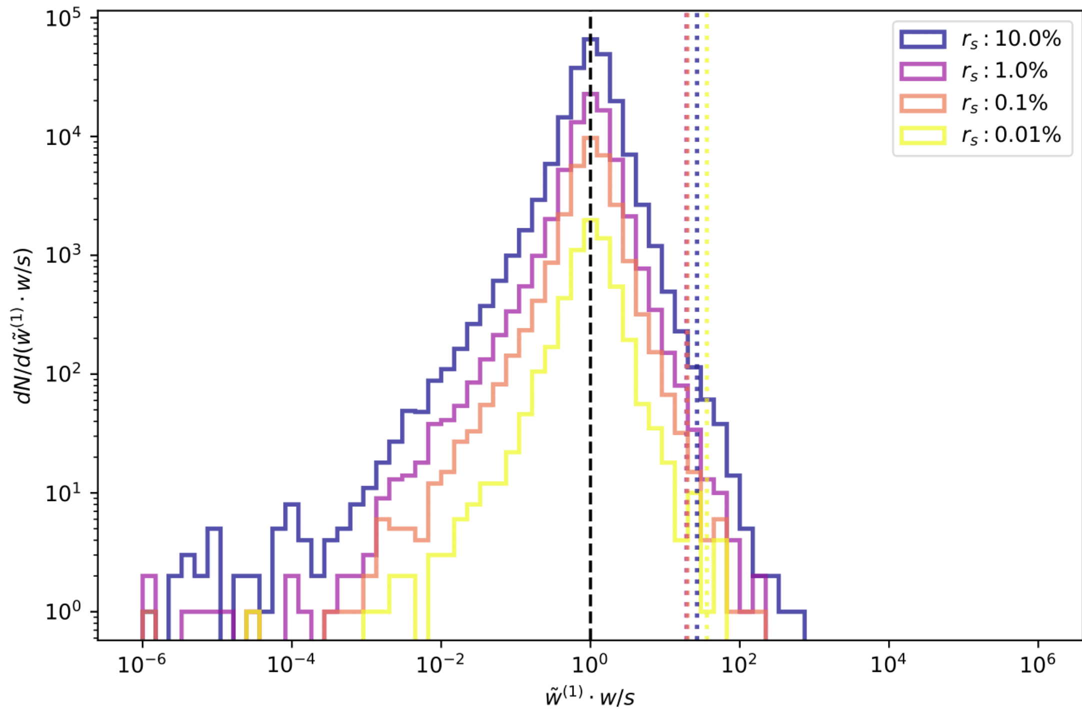


Figure 5.11: Distribution of the accepted events by the first unweighting respect the quantity $\tilde{w}^{(1)} \cdot w/s$ for the $pp \to e^-e^+ggd\bar{d}$ process. Each sample is obtained by a different overweight parameter for the first unweighting r_s. The dotted lines represent the x_thres corresponding to each sample using an overweight parameter for the second unweighting $r_\text{x} = 2\%$.

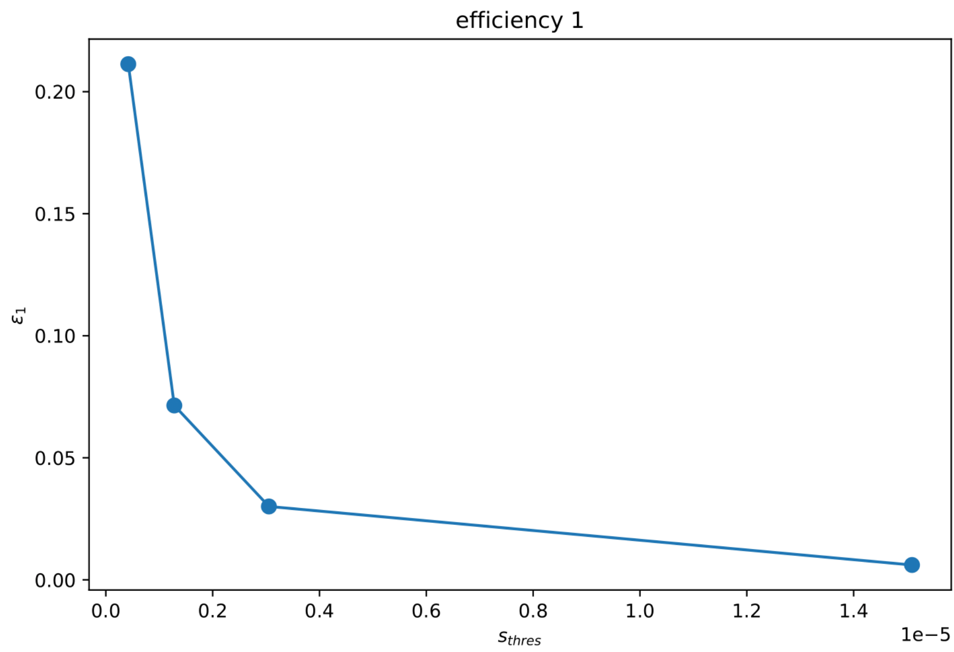


Figure 5.12: Efficiencies of the first unweighting respect the maximum used to compute the unweighting for the $pp \to e^- e^+ ggd\bar{d}$ process. Each point corresponds to an unweighting performed respect a different overweight parameter r_s. From left to right, each point corresponds respectively to $10\%, 1\%, 0.1\%, 0.01\%$ of allowed overweight in the first unweighting.

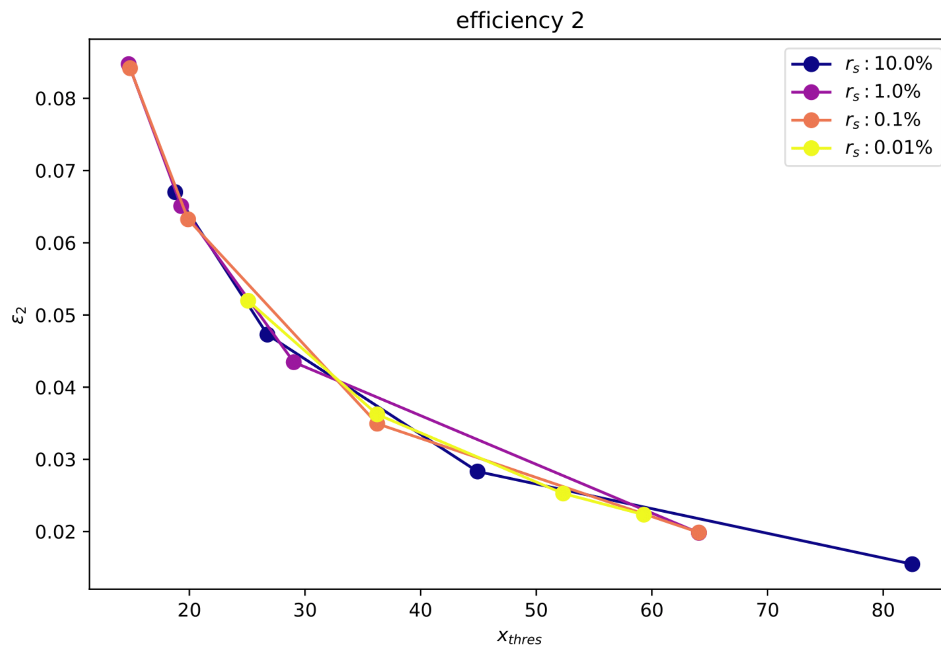


Figure 5.13: Efficiencies of the second unweighting respect the maximum used to compute the unweighting for the $pp \to e^-e^+ggd\bar{d}$ process. Each line corresponds to different unweightings performed over the same dataset, which is produced by a given r_s parameter. From left to right, each point corresponds respectively to $4\%, 3\%, 2\%, 1\%$ of allowed overweight in the second unweighting.

Comparing these results, with the efficiencies of the previous process, we observe a drop in the performance of the first and second unweighting. Some causes of this behaviour are describe here. First of all, we chose different overweight parameters for this process. We required a lower overweight in both the two unweightings, producing a lower efficiency. The choice of smaller overweight parameters $r_{\mathrm{s}}, r_{\mathrm{x}}$ is motivated by the distributions of s and w/s, which now span several orders of magnitude and, so, larger remaining overweights $\tilde{w}^{(1)}, \tilde{w}$ (defined respectively in eq. (3.4) and (3.5)) are produced. The presence of large overweights $\tilde{w}$ tends to drastically decreasing the Kish factor, as we can notice from the table (5.16). In fact, comparing this table with the previous table (5.7), we can observe that the unweighting of the second process is much more sensible to the overweight, producing lower Kish factor α. Secondly, the first unweighting is performed over a sample which have weights which span several orders of magnitude between them. Thus, the acceptance-rejection sampling, described in eq. (3.4), will provide for the smaller weights a probability to be accepted almost null, lowering the first efficiency. Lastly, in a similar way, also the second unweighting is performed over a sample with a wide distribution, as it is shown in fig. (5.11), producing a small second efficiency.

In the case of ε_2, we observe that due to the presence of events with a large $x = w/s$ ratio, the definition of the maximum x_{thres} is really sensible to the second overweight parameters r_{x}.

The fig. (5.14) show the distribution of overweighted events. As discussed before, we observe that to achieve Kish factor $\alpha \simeq 1$ we need to require small allowed overweight $r_{\mathrm{x}} \lesssim 1\%$. Moreover, as in the first process, the overweighted events of the first process are almost reabsorbed all final samples with $r_{\mathrm{s}} \lesssim 1$. For this process we can observe a larger dependence of the Kish factor on the first unweighting parameter r_{s}.

The fig. (5.15) shows the effective gain factor of the samples obtained by the different combinations of r_{s} and r_{x}. The first difference, with the previous process, that we can notice is that here the effective gain factor are not always positive. In fact, for small allowed overweight parameter r_{s} and r_{x} we produce samples with an effective gain factor close or lower than one, implying an almost null or a negative gain in time performances of the surrogate method respect the standard one. This behaviour is caused by the large maxima x_{thres} for small parameters r_{x}, which produce small second unweighting efficiencies ε_2. As discussed in section 3.2, to produce a positive gain in performance, one condition is that the second efficiency must be larger than the standard efficiency, $\varepsilon_2 > \varepsilon_{st}$. However in fig. (5.13), we can observe that for $r_{\mathrm{x}} \lesssim 1\%$, the second unweighting efficiency is close to the standard one, $\varepsilon_{st} = 1.4\%$.

For this process we measured $\langle t_{nn} \rangle = 0.00003s$. Compared to the standard unweighting, the surrogate method has an unweighting average time per even about $\langle t_{MG} \rangle / \langle t_{nn} \rangle \simeq$ 80 times faster for this process. $\langle t_{nn} \rangle$ has been measured unweighting a sample of 16000 events, as for $\langle t_{MG} \rangle$. However we observed an increase of the ratio $\langle t_{MG} \rangle / \langle t_{nn} \rangle$ with the increase of the sample size.

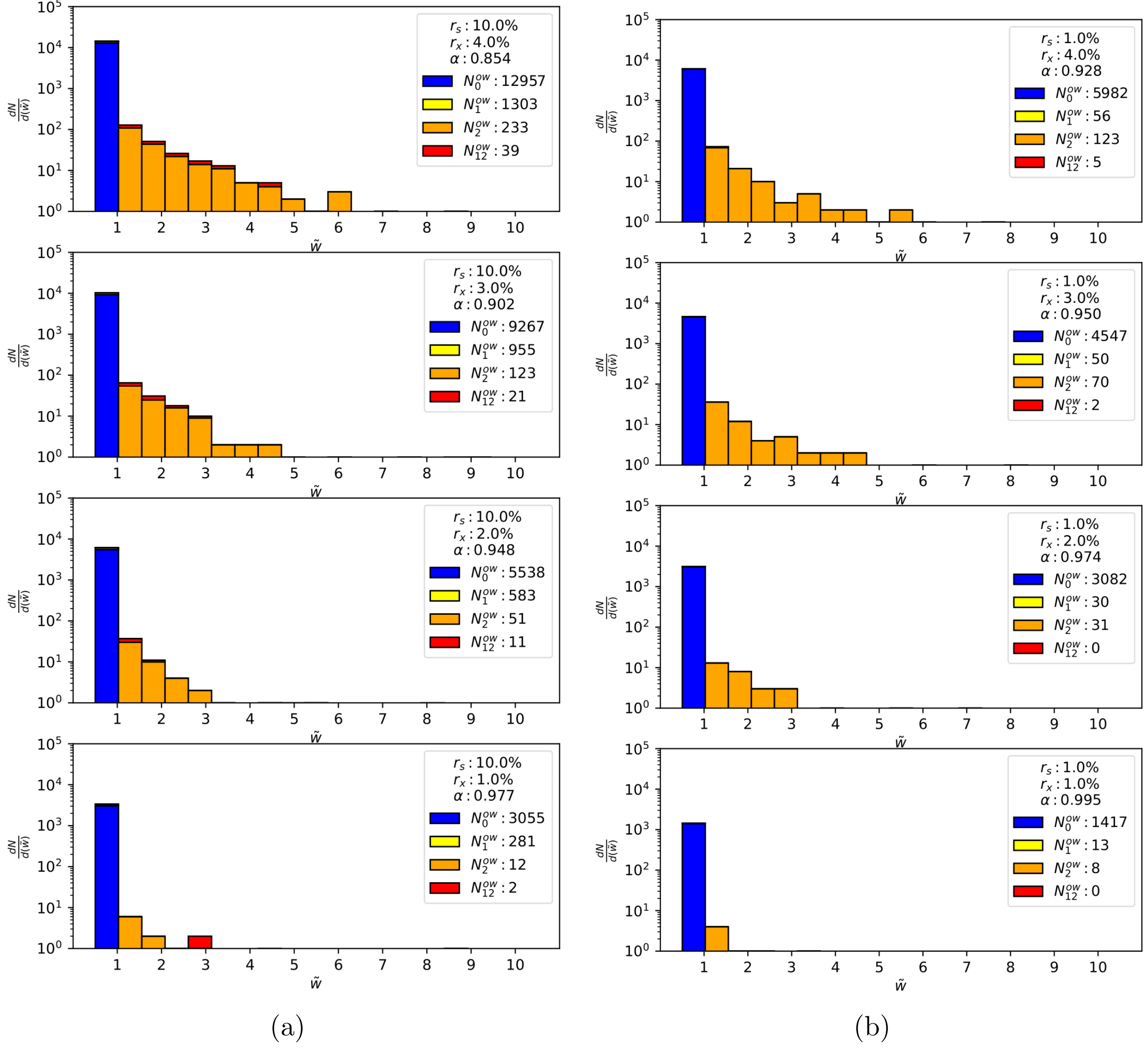


Figure 5.14: Comparison of the final event weights $\tilde{w}$ obtained by the unweighting of 2000000 weighted events for the $pp \to e^- e^+ g g d \bar{d}$ process. Fig. (**??**) represents the unweightings with different r_x parameters respect the same $r_\mathrm{s} = 10\%$ parameter. Fig. (**??**) represents the unweightings with different r_x parameters respect the same $r_\mathrm{s} = 1\%$ parameter.

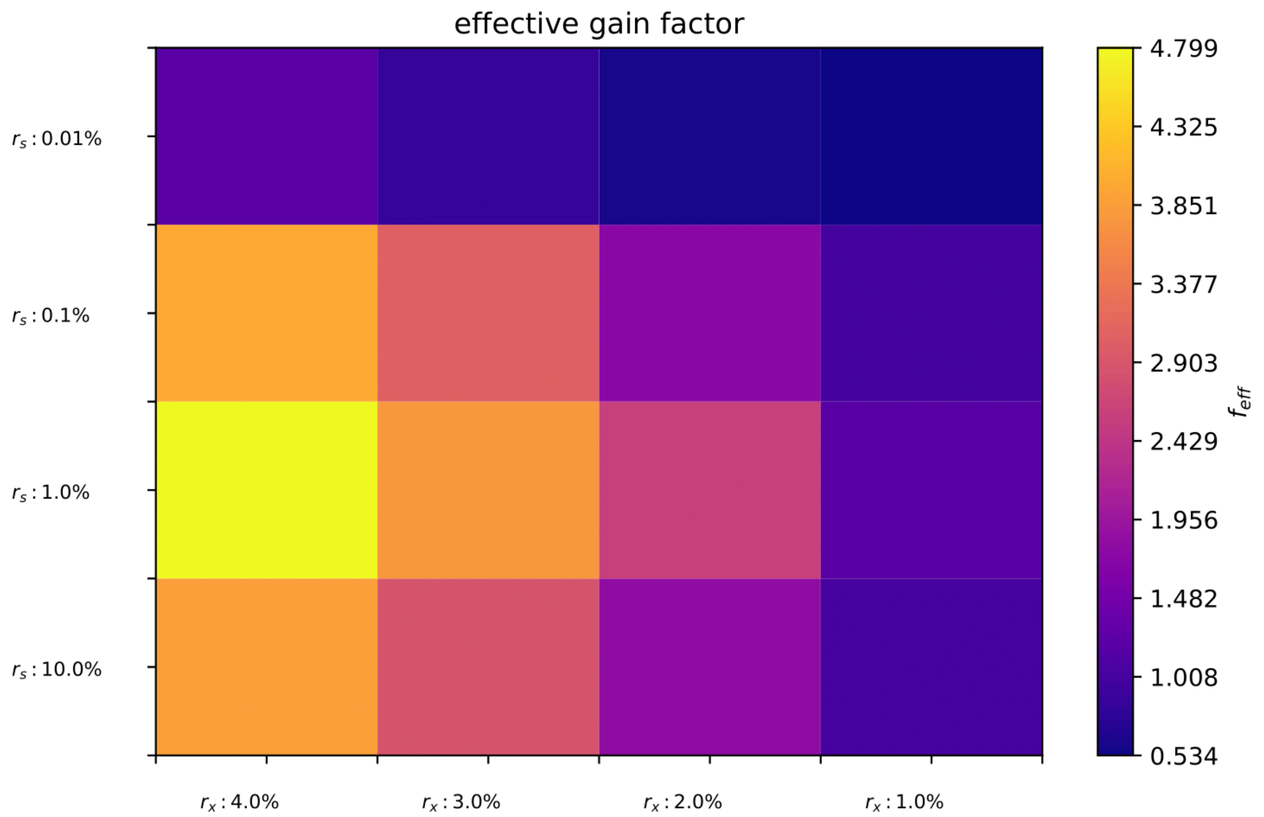


Figure 5.15: Effective gain factors achieved by different combinations of the r_s and r_x parameters for the $pp \to e^- e^+ g g d \bar{d}$ process. On the y-axis there is the allowed overweight in the first unweighting r_s and on the x-axis the allowed overweight in the second unweighting r_x. The magnitude of the effective gain factor is shown by the colorbar in the right.

In the fig. (5.16) we see the the summary of the results of this process. For example, in the case of ($r_\mathrm{s} = 1\%, r_\mathrm{x} = 1\%$), we obtain a quite good result with high Fish factor, $\alpha = 0.988$, and a positive effective gain factor, $f_\mathrm{eff} = 1.471$. While if we allow smaller Kish factor, we achieve larger f_eff.

In the fig. (5.17) we see that the minimum time ratio required by this process to achieve a positive effective gain factor is larger than the previous one in fig. (5.8). Similarly to the previous plots, these worse performances are mainly due to the lower efficiencies of the second process. Moreover we observe that in this case it is generally not possible to reach large effective gain factor, $f_\mathrm{eff} > 5$, just decreasing the average unweighting time per event of the surrogate method, or it would require too large time ratios, like $\langle t_{MG}\rangle/\langle t_{nn}\rangle > 10^4$.

We have noticed that the value of the second reduced maximum x_max increases with the sample size and this tends to decrease the efficiency of the second unweighting. This behaviour compromises the gain of the surrogate method for large sample size. To show this behaviour we can have a look to fig. (5.18). This table represents the summary of the results achieved by the same neural network model used to achieved the results above, but in fig. (5.18) it is used to unweight a sample of 100000 events. Comparing this table, with the one obtained using a sample of 1 million of events in fig. (5.16), we observe that with a smaller dataset, the surrogate method is able to reach much better results, with a Kish factor close to 1 and an effective gain factor of the order of 4. This likely points to an issue of the method that the ratio x between the surrogates and the real weight might not be bounded or achieve it's maximum in the tail of the distribution making difficult to have a correct estimate of that maximum.

5.3 Comparison to the original algorithm

In this subsection, we will compare our work to a similar work [4], done by K. Danziger, T. Janßen, S. Schumann and F. Siegert. Although, the two works have a similar algorithm, they present some major differences which we are going to discuss.

The first major difference is that the study of the paper [4] regards the event generation at LHC using AMEGIC, that is a part of the SHERPA package [19]. In particular, the SHERPA framework contains modules which automatically construct both the transition matrix elements and appropriate multi-channel integrators for a generic tree-level process. AMEGIC is one of the two built-in matrix element generators of SHERPA.

About the algorithm, there are important differences in the unweightings. Firstly they tested 2 different maxima functions, which one is the maximum quantile reduction, but with a fixed overweight contribution, such that overweighted events have a relative contribution of 1‰ to the inclusive cross-section. In our case we only used the maximum quantile reduction function because it has a better control over the final overweight and we tested it for several overweight parameters, paying attention to avoid results with a

r_s r_x	ε_1 ε_2	α	f_{eff}
10.0% 4.0%	0.211 0.067	0.854	3.864
10.0% 3.0%	0.211 0.047	0.902	2.880
10.0% 2.0%	0.211 0.028	0.948	1.812
10.0% 1.0%	0.211 0.015	0.977	1.021
1.0% 4.0%	0.071 0.085	0.928	4.799
1.0% 3.0%	0.071 0.065	0.950	3.774
1.0% 2.0%	0.071 0.043	0.974	2.584
1.0% 1.0%	0.071 0.020	0.995	1.203
0.1% 4.0%	0.030 0.084	0.930	3.987
0.1% 3.0%	0.030 0.063	0.952	3.065
0.1% 2.0%	0.030 0.035	0.984	1.750
0.1% 1.0%	0.030 0.020	0.996	1.007
0.01% 4.0%	0.006 0.052	0.982	1.218
0.01% 3.0%	0.006 0.036	0.995	0.861
0.01% 2.0%	0.006 0.025	1.000	0.604
0.01% 1.0%	0.006 0.022	1.000	0.534

Figure 5.16: Table containing the results of the unweighting respect all the combination of the overweight parameters $r_{\mathrm{s}}, r_{\mathrm{x}}$ for the $pp \to e^- e^+ g g d \bar{d}$ process.

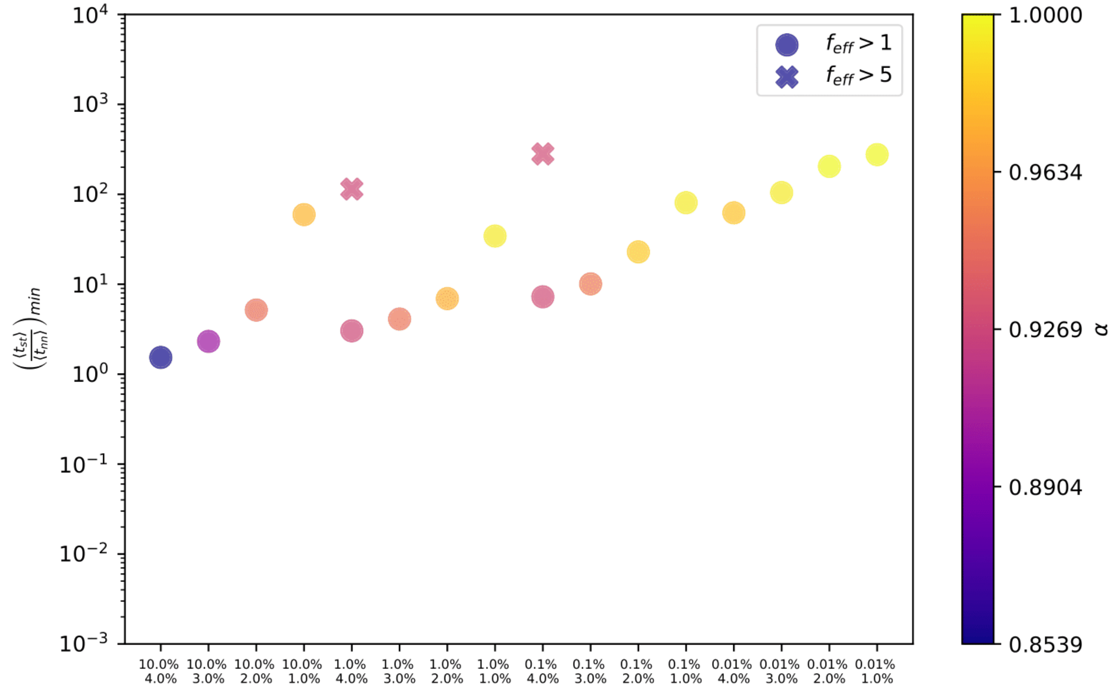


Figure 5.17: Comparison of the minimum ratio $\langle t_{st}\rangle / \langle t_{nn}\rangle$ between the ratios of the standard method and the surrogate method to achieve a given effective gain factor for the $pp \to e^- e^+ g g d \bar{d}$ process. Each column (composed by a vertical pair of a dot and a cross) corresponds to a given combination between r_s (above) and r_x (below). The color of each combination represents the Kish factor α associated to the corresponding final events sample of that combination.

r_s r_x	ε_1 ε_2	α	f_{eff}
10.0% 4.0%	0.211 0.070	0.946	4.454
10.0% 3.0%	0.211 0.054	0.963	3.511
10.0% 2.0%	0.211 0.033	0.986	2.215
10.0% 1.0%	0.211 0.018	0.998	1.226
1.0% 4.0%	0.075 0.109	0.989	6.657
1.0% 3.0%	0.075 0.102	0.992	6.200
1.0% 2.0%	0.075 0.074	0.998	4.530
1.0% 1.0%	0.075 0.062	0.999	3.813
0.1% 4.0%	0.029 0.107	0.988	5.329
0.1% 3.0%	0.029 0.079	0.996	3.979
0.1% 2.0%	0.029 0.074	0.998	3.726
0.1% 1.0%	0.029 0.053	1.000	2.684
0.01% 4.0%	0.014 0.109	0.991	4.139
0.01% 3.0%	0.014 0.080	0.998	3.088
0.01% 2.0%	0.014 0.076	0.998	2.919
0.01% 1.0%	0.014 0.054	1.000	2.076

Figure 5.18: Table containing the results of the unweighting respect all the combination of the overweight parameters $r_{\mathrm{s}}, r_{\mathrm{x}}$ for the $pp \to e^- e^+ g g d \bar{d}$ process. These results have been obtained over a sample of size 100000 (10 times smaller than the sample used above).

final Kish factor lower than about 0.98.

An other relevant difference regards the first unweighting, they perform it respect w_{thres}, suggesting that is the reduced maximum of the real weights of the events w_i. However, the first unweighting is performed over the surrogate weights s_i, so it seems to be more suitable to define the maximum respect the same surrogate weights s_i. A maximum defined over a different set of events, could lead to large overweights, in particular if $s_{\text{thres}} = max(s_i) > w_{\text{thres}} = max(w_i)$. Moreover, the overweights produced in this way could be more difficult to control due to the non-correspondence between the allowed overweight parameter r_{s} in w_i (which defines w_{thres}), and the resulting overweight contribution in the s_i sample.

About the second unweighting, they define the maximum x_{thres} in a similar way to the first unweighting, but in this case each event x_i is weighted by the surrogate weight s_i. Given N_1 events, with parameters x_i sorted such that $x_i \leq x_{i+1}; \forall i$, they defined the reduced maximum x_{thres} such that:

$$x_{\text{thres}} = min\left(x_j \middle| \sum_{i=j+1}^{N_1} x_i \cdot s_{1_i} \leq r_{\text{x}} \cdot \sum_{i=1}^{N_1} x_i \cdot s_{1_i} \right) \quad , \tag{5.2}$$

where r_{x} is the fraction of the contribution of the overweighted events. Then, they perform the second unweighting over the parameter x_i. Instead, we preferred to implement the second unweighting in order to partially compensate for the first unweighting. As explained in section (3.2.2), we defined x_{thres} in eq. (3.7) in order to include also the remaining overweight $\tilde{w}^{(1)}$ of the first unweighting and, then, perform the second unweighting over the quantity $x_i \cdot \tilde{w}_i^{(1)}$. In this way, if the first overweight $\tilde{w}_i^{(1)}$ does not contribute to the overweight in the second unweighting, the first overweight is reabsorbed.

Our approach in the second unweighting allows us to consider approximately only one final overweight source and so, to consider the final overweight equal to the second overweight $\tilde{w} = \tilde{w}^{(2)}$. In the other study, they have two separate overweight sources and, so, their final overweight for an unweighted event is given by:

$$\tilde{w}_i = \tilde{w}_i^{(1)} \cdot \tilde{w}_i^{(2)} = max\left(1, \frac{s_i}{w_{\text{thres}}}\right) \cdot max\left(1, \frac{x_i}{x_{\text{thres}}}\right) \quad . \tag{5.3}$$

This different approach implies that an initial overweight in the first unweighting, $\tilde{w}_i^{(1)}$, produces always an overweight in the final remaining weight $\tilde{w}_i$, due to the fact that $\tilde{w}^{(2)} \geq 1$.

About the hyper-parameters used in the neural network, they used the parameters described in the table (5.1). The only differences regard the learning rate, the batch size and the output activation function. About this last one, we prefer to use a linear output activation function in order to be able to predict also negative output. In fact, if we standardize the output with mean 0 and standard deviation 1, we obtain negative

NN hyper-parameter	value
hidden layers	4
nodes per layer	
activation function	Relu
output activation function	linear
loss function	mse
optimizer	ADAM
learning rate	0.001
batch size	1000

Table 5.1: Summary of the hyper-parameters used in the neural network model of the work [4] for the $pp \to e^- e^+ ggd\bar{d}$ process.

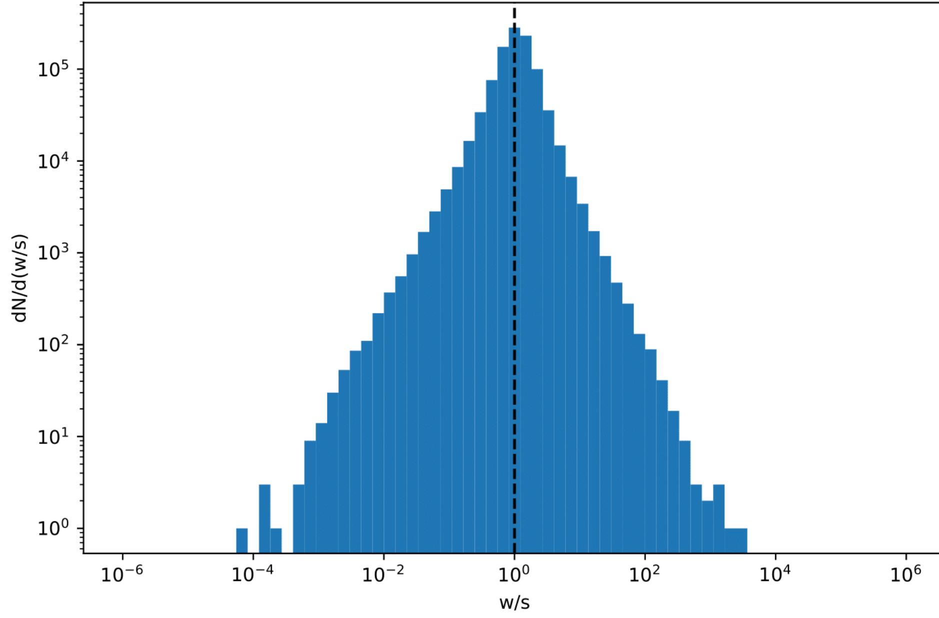
output even if we have only positive weight. We underline that we don't predict directly the weight of the events.

In the study presented in the paper [4], the number of inputs used for the neural network is quite larger than the minimum number required. As we explained in section (4.2), the number of degree for this process is equal to 16, but it can be easily reduced to 15. The study in question used 20 inputs for the neural networks, which are: the momenta along the beam axis of the initial particles and the three-momentum components of the final state particles (assuming the initial state momenta of partonic scattering events to be collinear with the incoming beams).

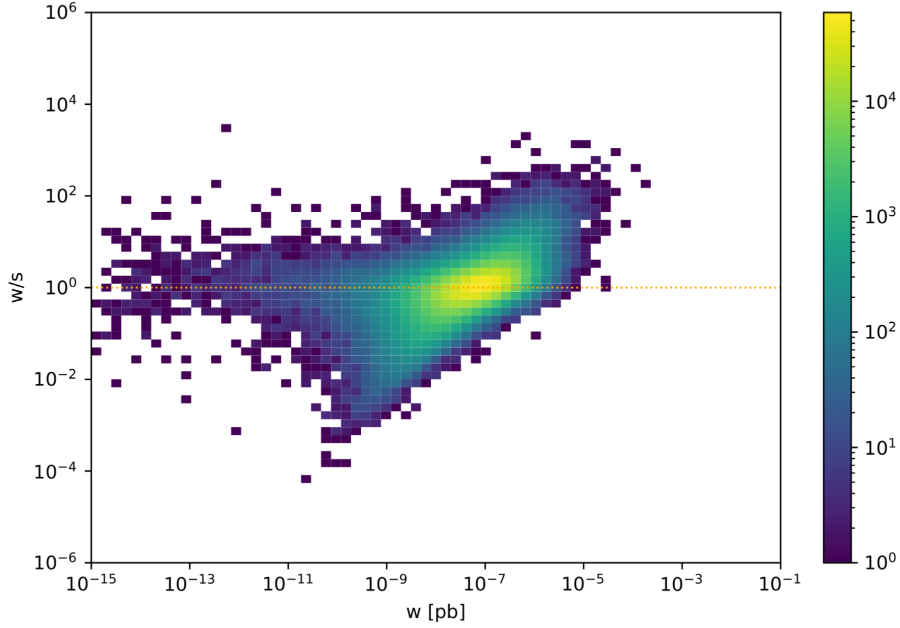
We defined a second algorithm, similar to the one presented in [4], and we applied this algorithm to the unweighting of the same sample used for the $pp \to e^- e^+ ggd\bar{d}$ process to compare the results of the 2 different algorithms. The second algorithm that we defined differs from the proposed one in [4] for the definition of x_{thres}, which in our case is defined respect the sample to unweight (and not in an other sample like in the other work). Moreover it differs also for the fact that we use only the maximum quantile reduction function. In the following part we present the results that we obtained with this different algorithm on the same dataset used in 5.2.

Comparing the fig. (5.19) with the fig (5.10), we see that the second approach has a better accuracy for the rare events with $w \lesssim 10^{-12}$. However, similarly to the first algorithm, it tends to overestimate the events with $w \lesssim 10^{-7}$ and to underestimate the events with $10^{-12} \gtrsim w \gtrsim 10^{-7}$.

In fig. (5.20), we observe that the maximum used for the first unweighting, w_{thres} in this case, it is about one or 2 order larger respect the maximum s_{thres} defined in the first algorithm. This explains the slightly lower efficiency of this second algorithm compared to the first one in fig. (5.12). However we underline that this discrepancy in the definition of the maxima should be at least partially caused by the small bias that we observed in the predictions.



(a)



(b)

Figure 5.19: Distribution of the surrogate weights s respect the true weights w for the $pp \to e^- e^+ ggd\bar{d}$ process obtained with the algorithm presented in [4]. Fig. (5.19a) represents a one-dimensional histogram of the ratio $x = w/s$. Fig. (5.19b) represents a two-dimensional histogram between the ratio $x = w/s$ and w.

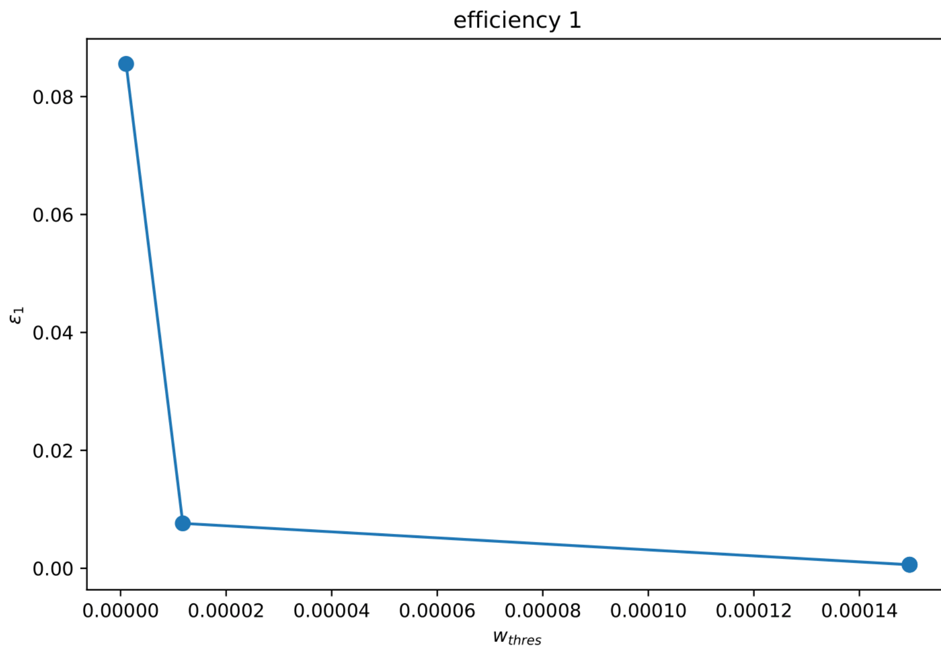


Figure 5.20: Efficiencies of the first unweighting respect the maximum used to compute the unweighting for the $pp \to e^- e^+ g g d \bar{d}$ process obtained with the algorithm presented in [4]. Each point corresponds to an unweighting performed respect a different overweight parameter r_s. From left to right, each point corresponds respectively to $10\%, 1\%, 0.1\%, 0.01\%$ of allowed overweight in the first unweighting.

In fig. (5.21) we see that the ε_2 tends to be larger using the second algorithm respect the first one in fig. (5.13). In fact, the definition of x_{thres} in eq. (5.2) allows to define a maximum on the parameter x_i weighted using the the prediction weight s_i. Thanks to this, the events with a larger weight (and usually a larger accuracy) will have a larger relevance on the definition of the maximum.

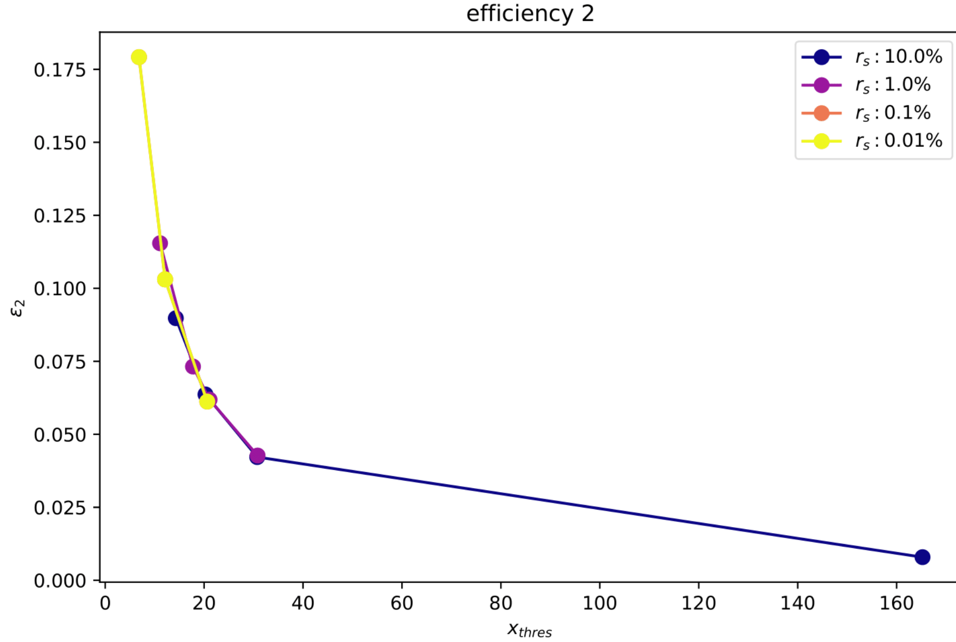


Figure 5.21: Efficiencies of the second unweighting respect the maximum used to compute the unweighting for the $pp \to e^- e^+ g g d \bar{d}$ process obtained with the algorithm presented in [4]. Each line corresponds to different unweightings performed over the same dataset, which is produced by a given r_s parameter. From left to right, each point corresponds respectively to $4\%, 3\%, 2\%, 1\%$ of allowed overweight in the second unweighting.

As discussed before, in fig. (5.22) we see that there is no re-absorption of the overweighted events of the first unweighting.

In fig. (5.23) we can notice that due to the large value of w_{thres} (compared to s_{thres}), the effective gain factor with a parameter $r_\text{s} \lesssim 0.1\%$ is always lower than 1. In the case of $r_\text{s} = 10\%$, we observe a large effective gain factor for the second algorithm, while in the case of $r_\text{s} = 1\%$, we observe a large effective gain factor for the first algorithm. To check these results it is possible to see the fig. (5.24).

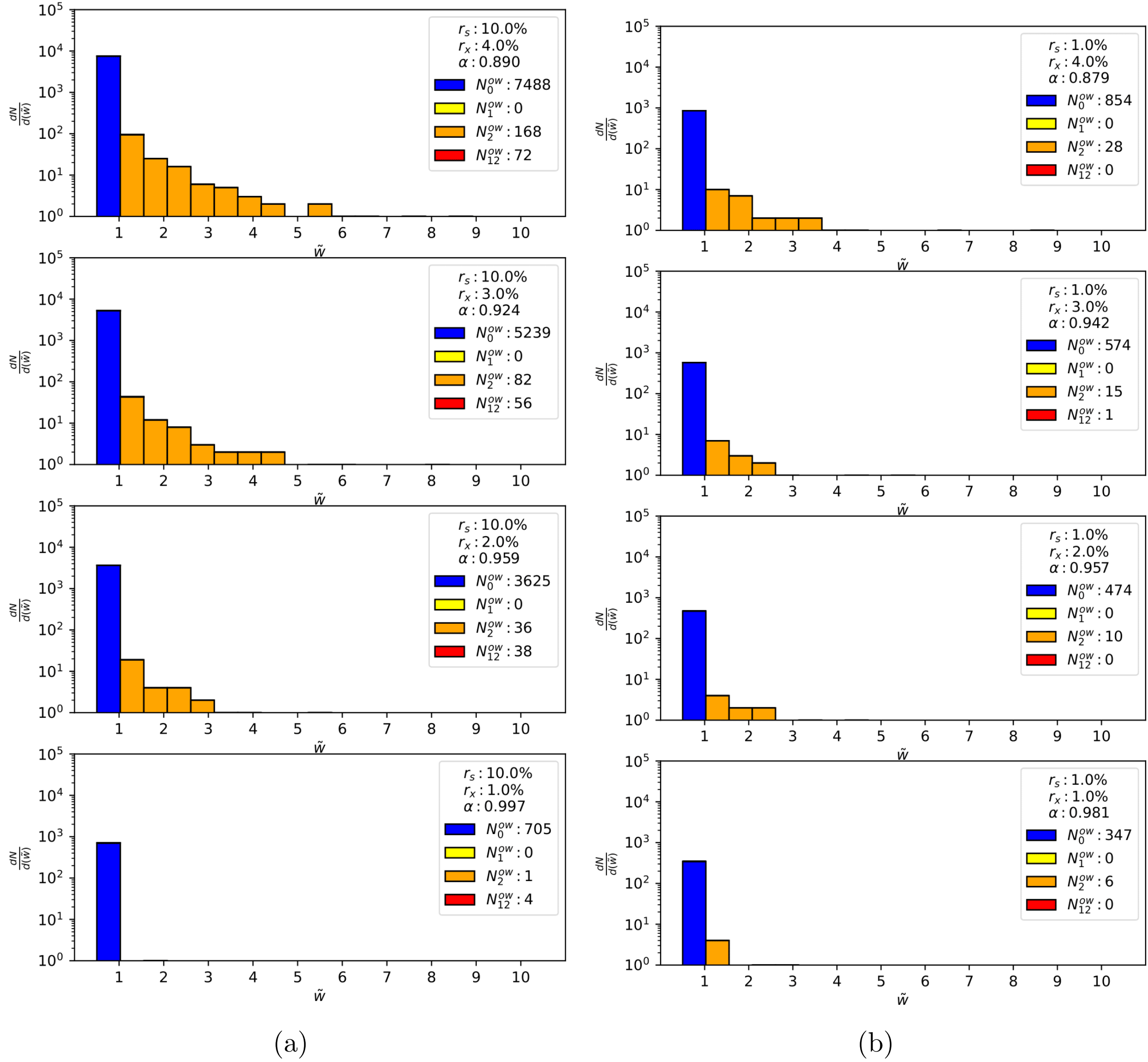


Figure 5.22: Comparison of the final event weights $\tilde{w}$ obtained by the unweighting of 2000000 weighted events for the $pp \to e^- e^+ g g d \bar{d}$ process obtained with the algorithm presented in [4]. Fig. (5.22a) represents the unweightings with different r_x parameters respect the same $r_\mathrm{s} = 10\%$ parameter. Fig. (5.22b) represents the unweightings with different r_x parameters respect the same $r_\mathrm{s} = 1\%$ parameter.

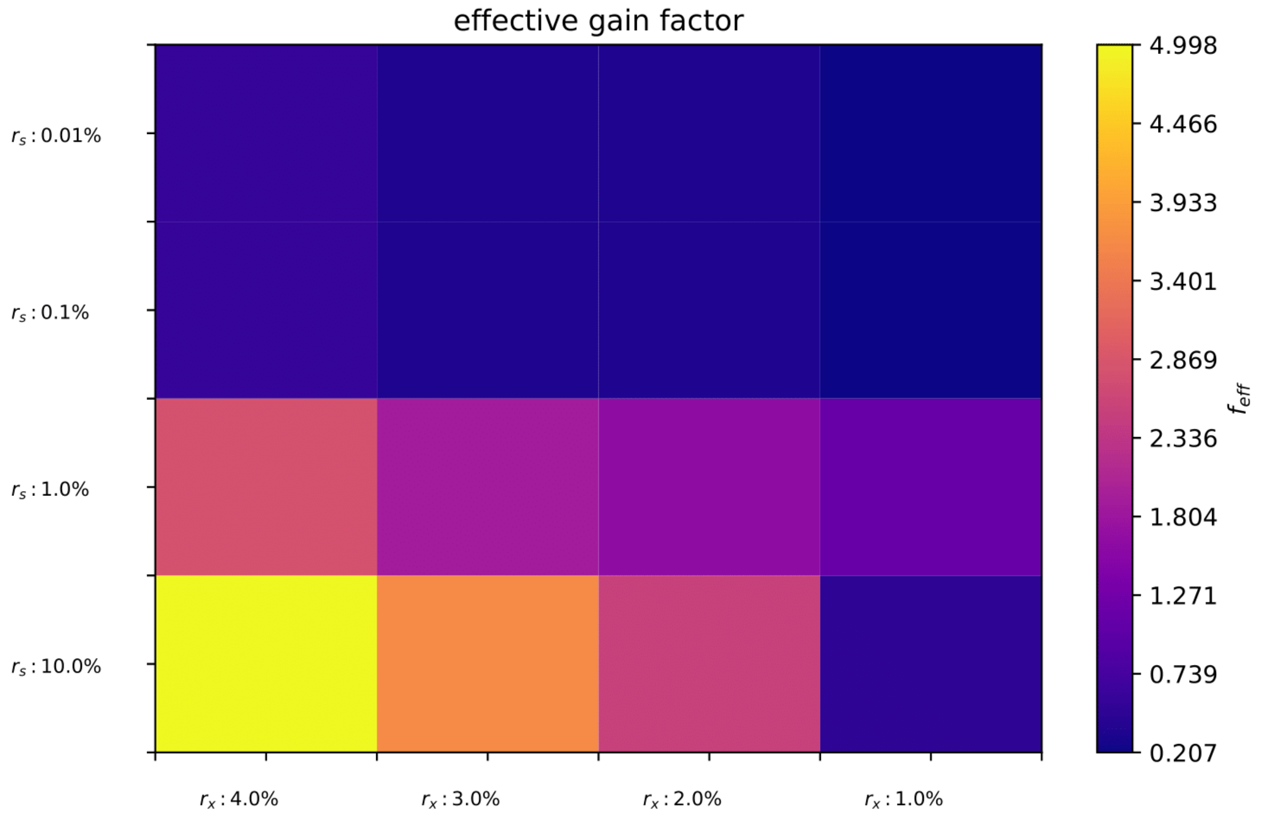


Figure 5.23: Effective gain factors achieved by different combinations of the r_s and r_x parameters for the $pp \to e^- e^+ g g d \bar{d}$ process obtained with the algorithm presented in [4]. On the y-axis there is the allowed overweight in the first unweighting r_s and on the x-axis the allowed overweight in the second unweighting r_x. The magnitude of the effective gain factor is shown by the color-bar in the right.

r_s r_x	ε_1 ε_2	α	f_{eff}
10.0% 4.0%	0.086 0.090	0.890	4.998
10.0% 3.0%	0.086 0.064	0.924	3.679
10.0% 2.0%	0.086 0.042	0.959	2.532
10.0% 1.0%	0.086 0.008	0.997	0.494
1.0% 4.0%	0.008 0.115	0.879	2.798
1.0% 3.0%	0.008 0.073	0.942	1.903
1.0% 2.0%	0.008 0.062	0.957	1.633
1.0% 1.0%	0.008 0.043	0.981	1.157
0.1% 4.0%	0.001 0.179	0.954	0.577
0.1% 3.0%	0.001 0.103	0.992	0.345
0.1% 2.0%	0.001 0.103	0.992	0.345
0.1% 1.0%	0.001 0.061	1.000	0.207
0.01% 4.0%	0.001 0.179	0.954	0.577
0.01% 3.0%	0.001 0.103	0.992	0.345
0.01% 2.0%	0.001 0.103	0.992	0.345
0.01% 1.0%	0.001 0.061	1.000	0.207

Figure 5.24: Table containing the results of the unweighting respect all the combination of the overweight parameters $r_{\mathrm{s}}, r_{\mathrm{x}}$ for the $pp \to e^- e^+ g g d \bar{d}$ process obtained with the algorithm presented in [4].

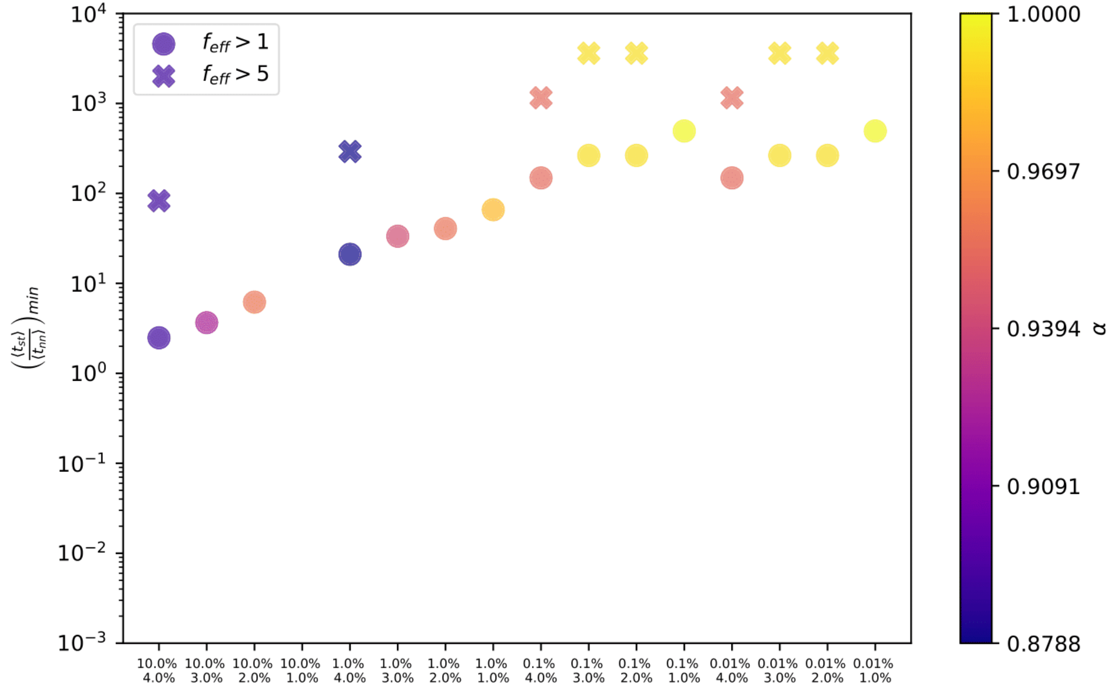


Figure 5.25: Comparison of the minimum ratio $\langle t_{st}\rangle/\langle t_{nn}\rangle$ between the ratios of the standard method and the surrogate method to achieve a given effective gain factor for the $pp \to e^-e^+ggd\bar{d}$ process obtained with the algorithm presented in [4]. Each column (composed by a vertical pair of a dot and a cross) corresponds to a given combination between r_s (above) and r_x (below). The color of each combination represents the Kish factor α associated to the corresponding final events sample of that combination.

Chapter 6

Conclusions

In this work we presented an alternative unweighting algorithm which could substitute the standard one in order to decrease the time and computational cost of the unweighting step during the generation of events. This algorithm takes advantage of a neural network surrogate which well-approximate the weight of the events and its evaluation is much cheaper than the evaluation of the true weights. The neural network surrogate is used in a first unweighting step, where the initial sample is unweighted with an acceptance-rejection sampling respect the predicted weights. After this step, the true weights are evaluated for all the accepted events. The true weights are used to perform a second unweighting step in order to address the error committed by the neural network. In order to increase the efficiencies of the two unweighting steps, we defined a reduced maximum with the maximum quantile reduction function. The definition of a reduced maximum allowed also to protect the maxima against rare large outliers, which could heavily affect them. However, the definition of a reduced maximum implied a partial unweighting of the events, with the presence of some overweighted events. The overweight of the overall final sample has been evaluated through the Kish factor. To avoid final samples with large overweight we tested different overweight parameters combination, in order to choose only those parameters with a relatively good Kish factor for a given process. To obtain a more efficient algorithm than the standard one, we need a fast surrogate neural network and good predictions of the true weights. To measure the gain in timing of the surrogate algorithm respect the standard algorithm we used the effective gain factor, that is defined as the ratio between the total times spent to perform the unweight.

In this work, we presented two processes. For the first process (a simple top pair production), we have seen that the surrogate method reached high efficiencies in both the unweighting efficiencies and the in speed of the surrogate unweighting (about 40 times faster than the standard unweighting), give a resulting effective gain factor about 7. Moreover, this result has been obtained with a Kish factor very close or equal to 1. This means that for simple processes, like $pp \to t\bar{t}$, the surrogate unweighting method can be up to about 7 times faster than the standard unweighting, without the need to

compensate for overweighted events. We underline that this result has been achieved without a detailed study of the optimization of the process. Thus, it could be possible to further improve the results just with a deeper study of the process. For example, it would be interesting to properly study the input of the neural network, in order to find the optimal number of inputs and which parameter pass to the neural network in order to help the learning of the neural network. It could be done also a deeper study of the model selection, in order to find the optimal neural network architecture.

For more complex processes, like the second process that we presented, $pp \to e^- e^+ ggd\bar{d}$, we observed that the training of the neural network requires a large dataset and that for the neural network it is more difficult to learn the weight of the events, in particular for less common events. In fact, the ratio between the true weights and the predicted one can span several orders of magnitude. For this reason, the unweighting is much more sensible to outlier events and this can affect the final result. Thus, in the case of complex events, we need to select more carefully the overweight parameters.

We underline that we observed a bias in the prediction of the logarithm of the weight that was absent in the prediction of the weight. The prediction of the neural network are not heavily affected by this bias and it should be canceled during the second unweighting, when we consider the error of the predictions respect their true value. However, this bias could have affected the result of the second algorithm, the one similar to [4], in section (5.3).

We observed that with the increase of the sample size of the events to unweight, for the surrogate method it is more difficult to define a reduced maximum for the second unweighting which can provide a good effective gain factor. While for sample size of the order of 100000 events, it is easier for the surrogate method to obtain Kish factor close to 1 and effective gain factor about 3 or larger. On the other side, the average time to unweight an event decrease with the increase of the sample size. As we can see from this example, the definition of the maximum of the second unweighting represents a serious problem for the performance of the algorithm presented in this work. This maximum tends to be affected by outlier events, in particularly for large sample, affecting the effective gain factor, and it is problematic for large simulations, where we need to unweight a very large sample of events. A solution could be to use another the definition of the maximum of the second unweighting, such that it would be not heavily penalized by outlier events. Otherwise, another solution, or a complementary one, could be to avoid the events which would heavily penalize the maximum. This can be achieved by using a particular loss which tends to penalize more the under-estimations of the true weight in respect to the over-estimations or using a dedicated trick for the tail. This trick could be, for example, to fix a minimum value for the prediction and all the values of the surrogates lower than such threshold would use that value ($s = max(s, s_{min})$). In this way, we would be sure to correctly estimate the maximum $x = w/s$ ratio for a relatively small sample without being impacted by the tail, which will be over-populated by construction.

About the comparison between the two surrogate unweighting algorithm, the one proposed in this work and the one proposed in the paper [4], we observed that our algorithm has a slightly better control over the overweight of the final sample. While the other algorithm presents a better effective gain factor, sometimes almost 2 time larger. This better result is mainly due to the definition of the maximum for the second unweighting x_{thres}. In fact, the maximum proposed in [4] allows defining the maximum avoiding the effect of outlier events and so obtain a smaller maximum, which guarantees a higher efficiency in the second unweighting.

This work could be extended with the generalization to non-positive event weights, which would allow also the study of higher-order perturbative calculations. In fact, negative event weights are introduced by definition for this type of calculation based on local subtractions methods like Catani–Seymour [31]. To extend the presented algorithm to non-negative weights, we should generalize the output, avoiding predicting the logarithm of the weight of the events. However, this should result in worse performance of the algorithm, thus an optimization of the algorithm and of the neural network would be required. In order to keep the advantages of reducing the order of magnitudes in the weights while preserving the sign, a suitable preprocessing for the output would be the hyperbolic arcsine, instead of the logarithm.

Another interesting extension of this work would be the substitution of the input used in this algorithm with the random number used in event generation to produce the phase-space points. This approach would have the advantage to avoid the computation of the phase-space points for the rejected events by the first unweighting. Moreover, this approach could help the neural network training to learn directly the useful information needed for the evaluation of the surrogate weight. However, the computation of the phase-space points is not computational expensive as the evaluation of the weight of the events, so this would have a smaller impact on the effective gain factor.

Bibliography

[1] J. Alwall, et al., *The automated computation of tree-level and next-to-leading order differential cross sections, and their matching to parton shower simulations*, *JHEP* **07** (2014) 079, [arXiv:1405.0301].

[2] G. A. Stewart, et al., *Hl-lhc computing review: Common tools and community software*, tech. rep., 2020.

[3] A. Collaboration, *ATLAS Software and Computing HL-LHC Roadmap*, tech. rep., CERN, Geneva, 2022.

[4] K. Danziger, T. Janßen, S. Schumann, and F. Siegert, *Accelerating Monte Carlo event generation – rejection sampling using neural network event-weight estimates*, *SciPost Phys.* **12** (2022) 164.

[5] c. d. Buffon, Georges Louis Leclerc, *Histoire naturelle, generale et particuliere ... Supplement*, vol. t.1 (1774). Paris, Impr. royale, 1774-1789, 1774. https://www.biodiversitylibrary.org/bibliography/169168.

[6] L. M. Brown, *Events of physics: From x-rays to quarks*, *Science* **212** (1981) 782–783.

[7] N. Metropolis, *The beginning of the monte carlo method*, *Los Alamos Science* **15** (1987) 125–130.

[8] N. Metropolis and S. Ulam, *The monte carlo method*, *Journal of the American Statistical Association* **44** (1949) 335–341.

[9] T. Haigh, M. Priestley, and C. Rope, *Los alamos bets on eniac: Nuclear monte carlo simulations, 1947-1948*, *IEEE Annals of the History of Computing* **36** (2014) 42–63.

[10] K. E. Atkinson, *An introduction to numerical analysis / Kendall E. Atkinson.* John Wiley Sons, New York, 2. ed ed., 1989.

[11] S. Weinzierl, *Introduction to monte carlo methods*, 2000.

[12] M. Kalos and P. Whitlock, *Monte Carlo Methods*, pp. I – IX. 12, 2007.

[13] E. Veach, *Robust Monte Carlo Methods for Light Transport Simulation*. PhD thesis, Stanford, CA, USA, 1998. AAI9837162.

[14] R. Y. Rubinstein, *Simulation and the Monte Carlo Method*. John Wiley , Sons, Inc., USA, 1st ed., 1981.

[15] G. Peter Lepage, *A new algorithm for adaptive multidimensional integration*, *Journal of Computational Physics* **27** (1978) 192–203.

[16] G. P. Lepage, *VEGAS - an adaptive multi-dimensional integration program*, tech. rep., Cornell Univ. Lab. Nucl. Stud., Ithaca, NY, 1980.

[17] R. Kleiss and R. Pittau, *Weight optimization in multichannel monte carlo*, *Computer Physics Communications* **83** (1994) 141–146.

[18] K. Ostrolenk and O. Mattelaer, *Speeding up MadGraph5_aMC@NLO*, 2021.

[19] **Sherpa** Collaboration, E. Bothmann et al., *Event Generation with Sherpa 2.2*, *SciPost Phys.* **7** (2019) 034, [arXiv:1905.09127].

[20] M. T. Heath, *Hypercube multiprocessors 1986*, .

[21] J. von Neumann, *Various techniques used in connection with random digits*, in *Monte Carlo Method* (A. Householder, G. Forsythe, and H. Germond, eds.), pp. 36–38. National Bureau of Standards Applied Mathematics Series, 12, Washington, D.C.: U.S. Government Printing Office, 1951.

[22] E. Byckling and K. Kajantie, *Particle Kinematics: (Chapters I-VI, X)*. University of Jyvaskyla, Jyvaskyla, Finland, 1971.

[23] R. H. Kleiss, W. J. Stirling, and S. D. Ellis, *A new Monte Carlo treatment of multiparticle phase space at high energies*, *Comput. Phys. Commun.* **40** (1986) 359–373.

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

[25] G. E. Forsythe, *Von neumann's comparison method for random sampling from the normal and other distributions*, *Mathematics of Computation* **26** (1972) 817–826.

[26] H. Wiegand, *Kish, l.: Survey sampling. john wiley & sons, inc., new york, london 1965, ix + 643 s., 31 abb., 56 tab., preis 83 s.*, *Biometrische Zeitschrift* **10** (1968) 88–89.

[27] S. Ganesh, “What’s the role of weights and bias in a neural network?.” https://towardsdatascience.com/whats-the-role-of-weights-and-bias-in-a-neural-network-4cf7e9888a0f.

[28] T. Developers, *Tensorflow*, Mar., 2023. Specific TensorFlow versions can be found in the ”Versions” list on the right side of this page.¡br¿See the full list of authors ¡a href=”htt ps://github.com/tensorflow/tensorflow/graphs/contr ibutors”¿on GitHub¡/a¿.

[29] K. He, X. Zhang, S. Ren, and J. Sun, *Delving deep into rectifiers: Surpassing human-level performance on imagenet classification*, 2015.

[30] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2014.

[31] S. Catani and M. H. Seymour, *A General algorithm for calculating jet cross-sections in NLO QCD*, *Nucl. Phys. B* **485** (1997) 291–419, [hep-ph/9605323]. [Erratum: Nucl.Phys.B 510, 503–504 (1998)].

Acknowledgements

I would like to thank my supervisors, Dr. Olivier Mattelaer and Prof. Fabio Maltoni, for their support over the preparation of this thesis. Their guidance was crucial to successfully complete this project. It was an incredible opportunity to work with them and to be able to explore these topics.

I would also like to express my gratitude to Dr. Ramon Winterhalder, for helping better understand the concepts of neural network and for his availability.

To Ivan and Malko, thank you for being the best hosts and for making me feel welcomed and at home.

To my friends from university, I was lucky enough to meet people that shared the same passions as me and with whom I lived the best years of my life, both in via Libia and via Mazzini.

To my friends from home for always being by my side, even when we were in different cities (and different countries) and for all the unforgettable years we spent growing up together.

I would like to give a special thanks to my family for being by my side and for giving me the chance to pursue my academic goals.

And finally, a special thanks goes to Maria Giulia, for supporting me every day and for making each place feel special, like for Bologna.