Chapter 5

Tune procedure and MCNNTUNES

The study of the underlying event, or more in general of soft QCD processes require the use of Monte Carlo generators that often are based on phenomenological models with lots of free parameters. These free parameters must be tuned on experimental data in order to obtain meaningful results from the simulations. The procedure of estimate the best parameters values in order to best reproduce the observables under analysis is called *tune*.
The tune procedure can be really computational expensive, in fact it requires to run the generator a very large amount of times, and usually these MC generators are really expansive in term of computational time required for just a single job.
To tune a certain number of parameters the number of jobs you have to run increase with this number of parameters. In fact, the parameters space dimension increase with the number of the sampling we want to perform and the granularity we want to achieve. So, in the past, different approaches have been developed and used to tune these MC generators. A brief description is reported for each approach in the following list:

1) **Manual tunes**: this approach is based on an optimization of the parameters made by eye. This is absolutely not the best way to tune some parameters, usually it requires a very large time for even semi-reasonable results since the process require a very large number of iterations .

2) **Brute force tunes**: a better way would be to perform a very dense sampling in parameters space, run the generator with every configuration and then chose the configuration corresponding to the output that better describes the experimental data. This is very computational expensive and a scan in a 5 parameters space with 10 division each requires $10^5 = 100000$ Monte Carlo runs this with a rising number of parameters becomes really impractical, also using computers batch systems as an example CondorHT (CERN).

3) **Parametrization-based tunes**: an even more better approach is to find a surrogate function to parameterize the response of the MC generator at different values of the parameters to tune, and try to study (minimize) this surrogate function instead of the real response of the generator. This is the right approach that is used in the high energy physics tune and that we are going to use in the follow for our tune.

Let us discuss this last approach with more details in the next section.

5.1 Parametrization-based approach

The parametrization-based approach is the most used method. The current state-of-art in the tune procedure is to use a polynomial function to fit the response of the generator. Once the parametrization is performed the tuned parameters are given by the minimization of this parameterized response function. This approach based on the polynomial parametrization have been implemented in the software PROFESSOR [41].
So the first step in the procedure is to fit the response of the generator using a surrogate function simpler to study than the real one (e.g. Professor instead of the arbitrary complex real function use a polynomial fitted to well describe the real function).

$$h(p) \xrightarrow{\text{parametrization}} \overline{h}(p) \quad , \tag{5.1}$$

where the real function have been substituted by the surrogate one.
After that, a *loss function* $\mathcal{L}(\overline{h}(p), h_{\text{data}})$ is defined between the surrogate function and the experimental data. A common choice for it is the χ^2 function defined as:

$$\mathcal{L}(\overline{h}(p), h_{\text{data}}) \equiv \chi^2 = \frac{(\overline{h}(p) - h_{\text{data}})^2}{\sigma^2} \quad . \tag{5.2}$$

In the end to find the best parameters estimation, this loss function need to be minimized. The set of parameters p_{best} that do this are the best evaluation that our generator can provide for the real values and we are going to call this set of best parameters: *tune*.

$$p_{\text{best}} = \arg\min_{p} \ \mathcal{L}(\overline{h}(p), h_{\text{data}}) \quad . \tag{5.3}$$

In our study instead of the common software PROFESSOR based on the polynomial parameterization we use the machine learning approach implemented in MCNNTUNES software [42] using Feed Forward Neural Networks. MCNNTUNES is a software developed by S. Carazza, S. Aioli and M. Lazzarin presented in [43] based on machine learning library TensorFlow [44]. MCNNTUNES is writen in python and it uses neural networks (NNs) that are trained in order to learn the generator behavior to the parameters variations. This remove the polynomial constraint in the fit of the generator respond function in fact one of the main feature of the Neural Network is that they are universal function approximators.
Let's make a brief introduction on machine learning and in particular on neural networks in order to understand why this choice.

5.2 Machine Learning and Neural Networks

Machine learning (ML) is a particular type of Artificial Intelligence it consists in systems that learn automatically by the data that are fed into it and not by the explicit programming of the algorithm. It is clear that ML requires the training of the algorithm in order to have a predictive output related to the problem under analysis. The training is the most important step in the ML approach in fact is this step that gives to the ML the ability to return a predictive output without it the output we get is not able to give us a meaningful result.

A particular type of ML is Deep Learning that uses neural networks with more than one layer organized in a hierarchical structure to solve the problem. This is the type of ML we are interested in. But before we start with the explanation of the simple possible neural network that can be created in order to understand the basis in the logic of each unit that is going to compose the neural network.

5.2.1 Neural Networks - Perceptron

The concept of the Neural Network (NN) was developed in 1958 by Frank Rosenblatt. He introduce the simpler example of NN: the perceptron [45]. A representation of a percepton is shown in Fig. 5.1 the input values are weighted and summed, an additional offset b can be introduced, then the weighted-sum is passed to an activation function (step function). So, the output that the perceptron returns is:

$$h(x) = \text{step}\left(\sum_j w^j x_j + b\right) \tag{5.4}$$

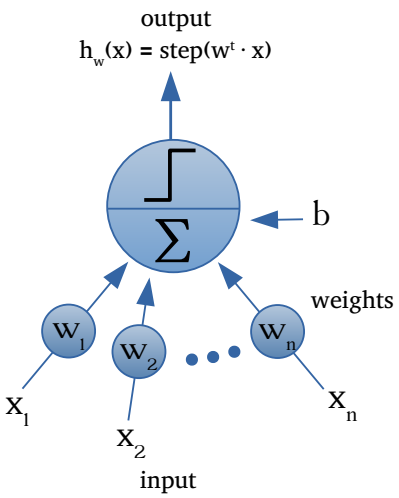

Figure 5.1: A schematic representation of a perceptron.

The revolutionary feature of the perceptron was the ability of learning by an adjustment of the weights. But a single perceptron is not enough this kind of logical units have lots of limitations. An example of limitation for the perceptron is shown in Fig. 5.2 where the impossibility of implement a XOR operation using a perceptron is shown with a graphical explanation. The perceptron is a linear classification algorithm and in the image is represented as a blue line that set a boundary for the acceptation of the hypothesis.

The structure is very simple with a single unit but is not enough it have a lot of limitation so we have to introduce the concept of NN with more than one unit if we want eliminate these limitations and approximate every type of functions.

5.2.2 Feed-Forward Neural Networks

In a Neural Network different units called "neurons" are linked together. Different type of NNs exist and they are classified according to the various ways the neurons

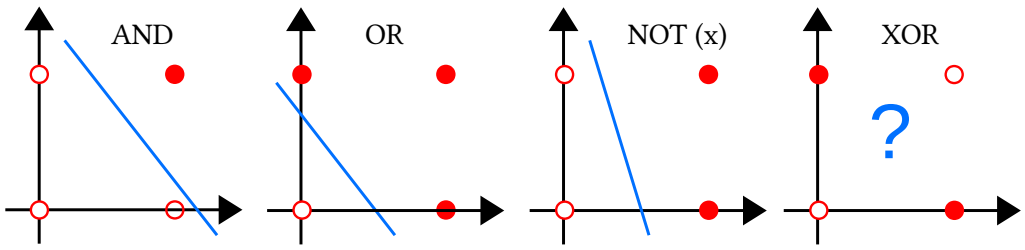

Figure 5.2: The figure shows one of the limitations of the perceptron. The XOR operation is not possible with a linear cut.

are linked each other. We are interested in *fully-connected Feed-Forward NNs*, that are the ones used in MCNNTUNES.
In fully-connected NNs each neurons from a layer are connected to every neurons in the next layer. While, the Feed Forward attribute refers to the fact that the NN have not internal recursions (loops) between neurons but all the neurons from a layer are connected forward to the ones of the next layer.
Fig. 5.3 shows a schematic view of a fully-connected feed-forward multi-layer NN the basic idea is that the neurons can get some value in input and return a value as output. The output of neuron is then sent to all the neurons in the next layer and weighted differently for each one. To be more specific: each unit j in the hidden layer (i) takes a vector of values x^k, that come from all the k neurons of the previous layer, and an offset ϑ_j as input, compute the weighted sum $\sum_k w_{kj}x^k$ and apply the activation function ϕ to the result, common choices for this function are *tanh* or *sigmoid* functions. So, the total output of the (i)-th layer in the network is the function:

$$f^{(i)}(x) = \sum_{j=1}^{N^{(i)}} \varphi\left(\sum_k w_{kj}^{(i)} x^k + \vartheta_j^{(i)}\right) \quad , \tag{5.5}$$

where $N^{(i)}$ is the number of hidden units in the (i)-th layer.
One of the biggest feature of the NNs is that the they are universal function approximators [46, 47] the only request for this is that a sufficient number of hidden layers is available. This is the main reason why we want use a NN based approach instead of the polynomial one

As mentioned before the main feature of all the types of NNs is the ability to learn from data without being directly programmed. But to to this and so get some predictive results from the NN a learning algorithm have to be defined.
A common training algorithm for the NN is the *back-propagation*. Where a set of Monte Carlo simulations is used to train the NN. The back-propagation procedure is based on the idea of change the weights, w_{jk}, and the offsets, ϑ_j, in order to minimize a loss function usually defined as the mean squared error (E):

$$E = \frac{1}{2}\sum_i (h_i(x^j, w_{jk}) - d_i)^2 \quad , \tag{5.6}$$

where the h_i are the value in output from the NN and the d_i the real value known from the Monte Carlo truth.

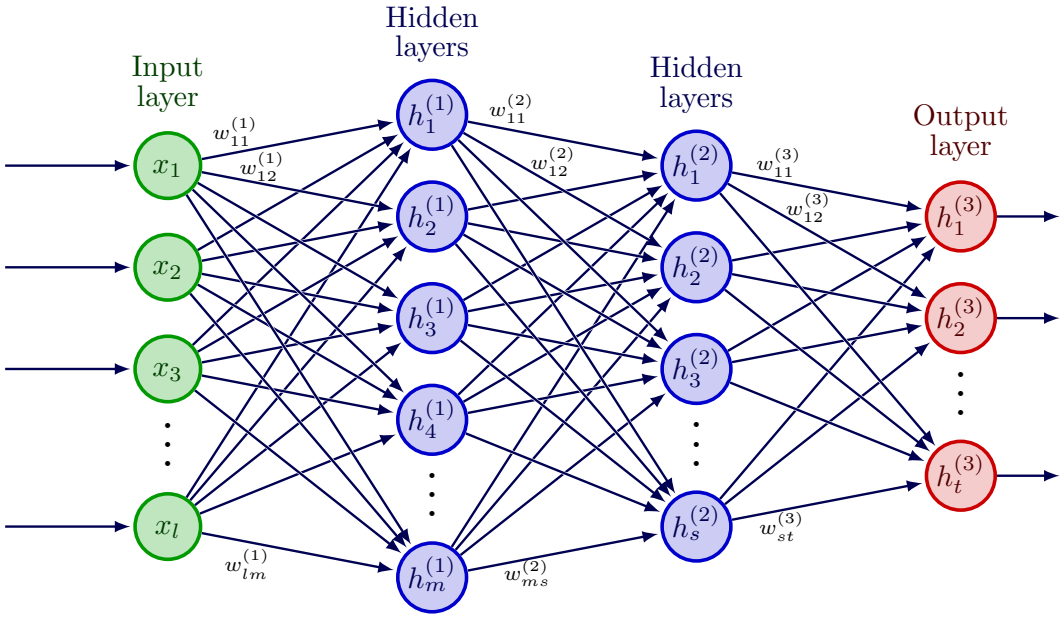

Figure 5.3: A fully-connected feed-forward neural network with more than one hidden layer, the green one is the input layer and takes the value in input and usually scale them in the range $[0, 1]$ and then the data sequentially go trough the neurons in the hidden layers (blue) where all the steps described above are performed for each layer. At the end the results of the computation are collected by the output layer (red).

In the back-propagation algorithm the weight and the coefficients are update using the *steepest-descent minimization*:

$$w_{jk}^{(i+1)} = w_{jk}^{(i)} - \lambda \left(\frac{\partial E}{\partial w_{jk}} \right)^{(i)} \quad ; \qquad \vartheta_j^{(i+1)} = \vartheta_j^{(i)} - \lambda \left(\frac{\partial E}{\partial \vartheta_j} \right)^{(i)} \tag{5.7}$$

where λ is the learning rate and is a user-tunable free parameter and it controls how much change the weights each time the weights are update. It is one of the main parameter in the neural networks a to small value for it can lead to a failure in the training procedure while a to large one can lead to unstable results.
The mini-batch are subset of our training set that contains the Monte Carlo simulations that are used to calculate the gradient. The size of the mini-batches used to train the NN is also a free parameters: smaller batches are faster to compute but the gradient direction is not the real one just an approximation; while bigger batch size give a good approximation of the direction of the steepest-descent but can be computational expensive.
The number of evaluation of the entire training set is called epochs and it is tunable by the user.
Note that the batch size and the number of epochs are strictly related. A training set of 50 MC runs with a batch size of 10 and a number of epochs of 1000 requires 5000 iterations while if we use a batch size of 25 it requires only 2000 iterations to run over all the training set.
A schematic representation of the training process is displayed in Fig. 5.4 where the training set is subdivided in the mini-batches that one by one are fed to the NN. Then, the Monte Carlo truth and the output of the network are used to compute the

loss function E, using this information the back-propagation, based on the gradient descend algorithm, updates the weights in the descent direction of the gradient calculated using the mini-batch. All the procedure is repeated with every mini-batch and for a user tunable number of epochs.

Note that the prediction capability of the Neural Network is dependent on the selection of all these user-tunable parameters that control the Neural Network architecture. All these parameters are usually referred as *hyperparameters*.

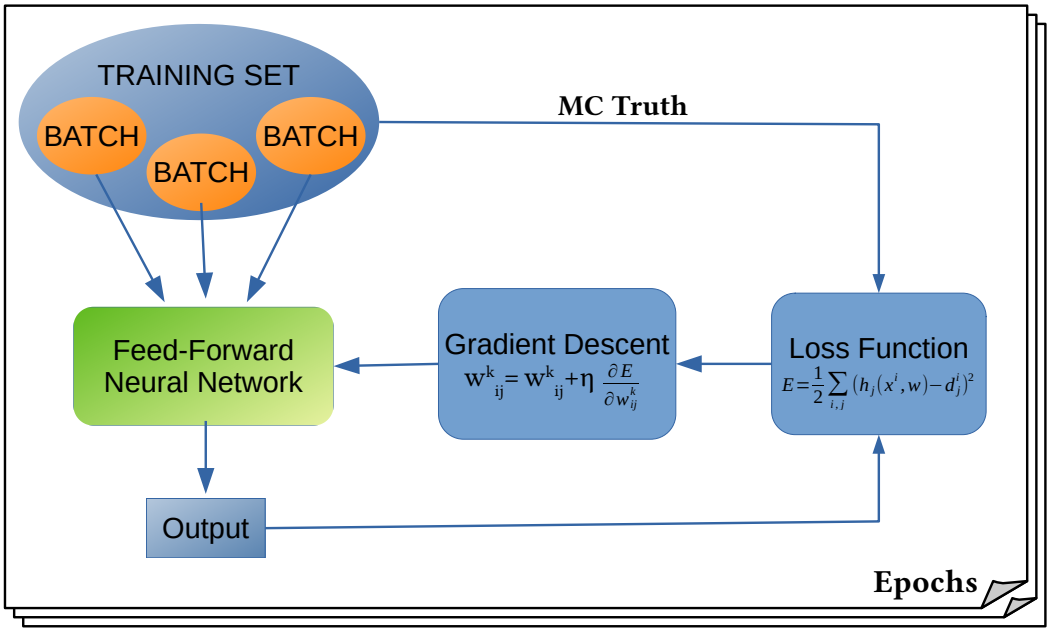

Figure 5.4: The training set used to train the NN is divided in mini-batch (the mini-batch size is a user tunable parameter) than one by one the batches are feed to the NN the output of the network together with the Monte Carlo truth are used to calculate the loss function. Then the weights and the offset are updated as described in Eq. 5.7 with the back-propagation algorithm. This is done with each mini-batch in the training set and for a number of epochs defined from the user.

In the next section `mcnntunes` is introduced and all its working modes are explained in detail.

5.3 MCNNTUNES

`mcnntunes` [43] is a Shower Monte Carlo generators tuning tool that implements a tune procedure based on the use of Feed Forward Neural Networks (FFNNs). The advantage of using FFNNs has been described above and it is that they are universal function approximators at the simple cost of have a sufficient number of hidden units in the hidden layers. This feature allows us to remove the polynomial bias present in `professor` tool.

`mcnntunes` offers two different main operation modes: *PerBin Model* and *Inverse Model*. The first one is based on approach similar to the one in `professor` but where the response of the generator is parameterized using these FFNNs, one for each bin; the later is a totally new approach where the NN (only one in this case)

is trained to learn the inverse function of the generator response and then used to try to infers the parameters value starting from the experimental values of the bins in the distributions used.

5.3.1 Sampling Phase and Training Set Generation

The two approach have the same starting point that is a sampling of the parameter space (e.g. for the UE analysis we use the parameters space shown in Table 4.1), than the generator is run with every sampled configuration. All these MC runs are going to build our dataset that is called *training set*.
A schematic explanation of how the sampling phase and the training set generation were performed in CMSSW environment (lxplus CERN) is shown in Fig. 5.5. In more detail the sampling start by mean of MCNNTUNES script called MCNNTEMPLATE with a sampling of the defined parameter space. The sampling generates N different configuration for the parameters to tune, this are then encapsulated in a runcard for PYTHIA that contains all the necessary information to make PYTHIA work properly (beams energies, type of event to simulate etc.). Once these runcard are saved the PYTHIA generator is run with every configuration generated and this is performed not locally but on a computers batch (e.g. the CERN one CondorHT) in order to gets all the results in few days. The runcards we pass to Condor contain also the information on the analysis we want to perform and on how to fill the various histograms for the observables. Once the generators runs end the outputs are saved in the YODA format, that is a standard type of data file for these analysis. The set of these YODA files, containing the information on the MC runs, composes our training set that then can be used for the tuning procedure.

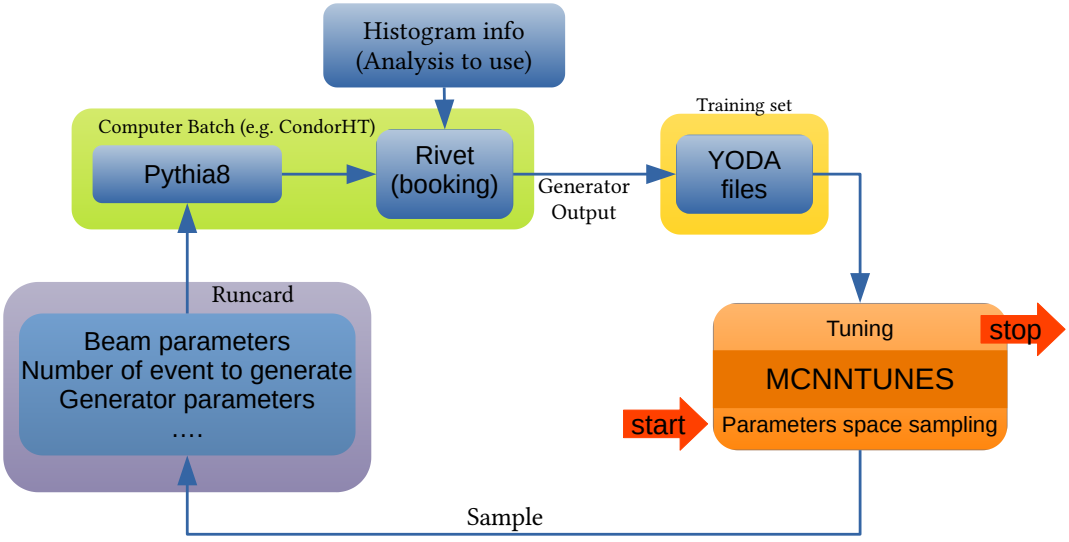

Figure 5.5: A schematic description of the main step from the sampling phase to the generation of the training set used for the tune procedure. These shows how the work was performed in CMSSW environment. The starting point and the final tune procedure are both controlled by MCNNTUNES. In the middle of these two phases the generation of the training set is required, this is performed running the generator many times.

MCNNTUNES offer also the possibility of change the value of the hyperparameters. The value that can be modified for the Neural Network. It is possible to chose the NN architecture: the number of hidden layer, the number of neurons for each layer and the activation function used. It is also possible to set the number of epochs, the batch size and the learning rate in order to have the best train for the architecture selected and other choice are possible. This is really important in order to get the best possible results for the tune. Below we are going to discuss the procedure used for the Inverse Model in order to search for the best hyperparameters configuration.

5.3.2 Per Bin Model

PerBin Model is a parametrisation-based method. The main idea, as shown in Fig. 5.6 is to build a model (i.e. a neural network) for each bin in order to parameterize the generator output. Each NN takes the parameters values as input and returns the bin value as output.
All these NNs are then trained feeding the MC runs from the training set and using a gradient-based algorithm, as usual for feed forward neural networks, with mean squared errors as loss function.
Once, the NN is trained, the last step is the tune in which one actually get the best parameters estimation. This step define a surrogate loss function for the tuning problem. In fact, the parameterization step return a model $h^{(i)}(\mathbf{p})$ for each bin, i, where $\mathbf{p}$ is the vector of the parameters.
Then, this surrogate loss function defined as:

$$\chi^2 = \sum_{i=1}^{N} \frac{\left(h^{(i)}(\mathbf{p}) - h^{(i)}_{exp}\right)^2}{\sigma^2_{(i)}} \tag{5.8}$$

need to be minimized in order to evaluate the best estimation for the parameters. In `mcnntunes` this minimization is performed using the CMA-ES algorithm [48].
So the best estimation for the parameters is the configuration of parameters that minimize this χ^2.

Errors evaluation

MCNNTUNES PerBin model as introduced in [43] did not have a proper error evaluation. In fact it was using as error the final width of the distribution of sampled point in the CMA-ES algorithm. But, this was not working. Thanks to S. Carazza and with the help of M. Lazzarin we had the opportunity of handle the code and add a proper errors estimation.

Now, the errors evaluation for the PerBin Model is given by the definition of a confidence interval using the χ^2 function. In fact, as shown in section 9.6 and 9.7 of [49], for an estimators vector $\hat{h}(\mathbf{p}) = (\hat{h}^{(1)}(\mathbf{p}), \hat{h}^{(2)}(\mathbf{p}), \ldots, \hat{h}^{(n)}(\mathbf{p}))$ for the parameters $\mathbf{p}$ the probability distribution function and the likelihood (the χ^2 in our case) in limit of a large sample[1] are Gaussian distributed. The probability distribution function

[1] In other case this still a good approximation.

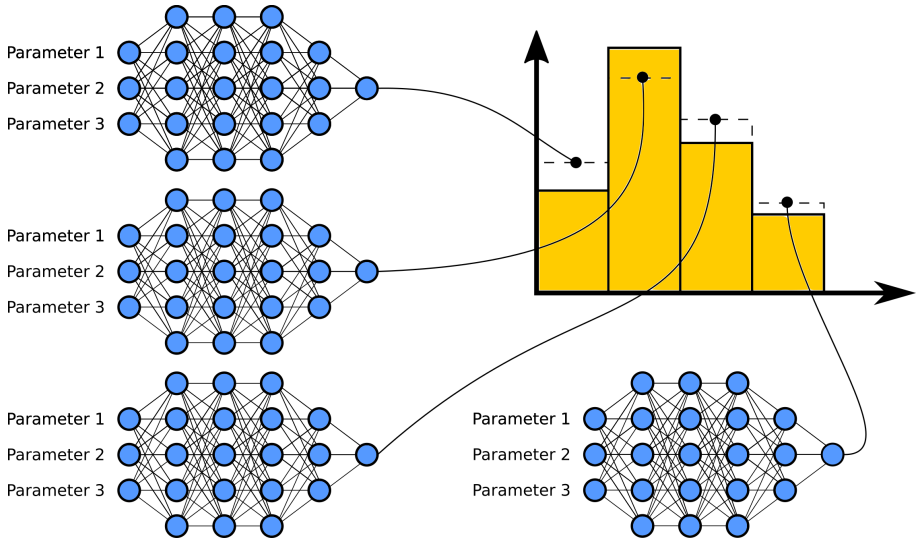

Figure 5.6: Figure from [43]

percentile	Q_γ				
	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$
0.683	1.00	2.30	3.53	4.72	5.89
0.90	2.71	4.61	6.25	7.82	9.24
0.95	3.84	5.99	7.82	9.49	11.1
0.99	6.63	9.21	11.3	13.3	15.1

Table 5.1: The table report the values of the quantile Q_γ for different confidence level 0.683 is the row corresponding to the 1σ definition and is the one of our interest.

for the estimators is than:

$$f(\hat{h}(\mathbf{p})|h(\mathbf{p})) = \frac{1}{(2\pi)^{n/2}|V|^{1/2}} \exp\left[-\frac{1}{2}\left(\hat{h}(\mathbf{p}) - h(\mathbf{p})\right)^T V^{-1}\left(\hat{h}(\mathbf{p}) - h(\mathbf{p})\right)\right], \tag{5.9}$$

where T is the transpose vector and V^{-1} is the inverse covariance matrix. Can be shown that also the likelihood is Gaussian as the probability distribution function. So, we can define our confidence interval using the χ^2 statistic as:

$$\frac{\chi^2(\text{c.i.})}{N_{dof}} = \frac{\chi^2_{min}}{N_{dof}} + \frac{Q_\gamma}{N_{dof}} \tag{5.10}$$

The variation is dependent on the number of parameters and on the chosen confidence level ($1\sigma = 0.683$ in our case) and a list of the values is reported in Table 5.1. Then, the error is defined as the value of the parameters that give a deviation from the minimum value of the χ^2/N_{dof} equal to the Q_γ/N_{dof} value for a confidence level of 0.683, as defined in Eq. 5.10.
An example for the evaluation in `mcnntunes` is shown in Fig. 5.7 where the green line is the deviation from the minimum value of the χ^2 defined in Eq. 5.10 and the errors are given by the points where the χ^2/DoF reach these value. Therefore, the intersections between green and blue line that is the χ^2/N_{dof} for the different values of the parameter.

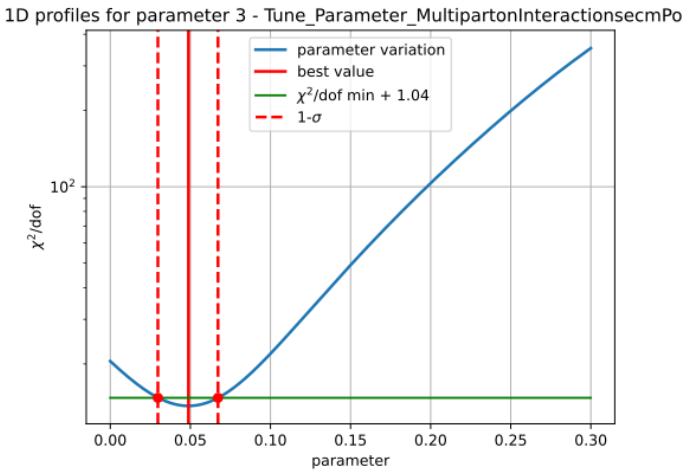

Figure 5.7: This figure shows the error evaluation in `mcnntunes` for the `MultipartonInteractions:ecmPow Pythia8` parameter. The blue line is the value of the $\chi 2/N_{dof}$ evaluated for different parameter values. The best estimation is indicated by the vertical red line, while the green line is the quantity in Eq. 5.10. The error is evaluated from the intersection of blue and green lines.

5.3.3 Negative aspects

One of the negative aspect is that this model is more computational expensive than the below discussed Inverse model. This is due to the large number of NNs built and trained from this model. The high cost in term of time required to get the model work do not give the possibility for a scan in the hyperparameters space in order to obtain the best configuration for the NN architecture.
This can impose some limitation on the model performance that cannot get its maximum performance.

5.3.4 Inverse Model

The Inverse Model is the most innovative tuning procedure introduced by `mcnntunes`. This model contrarily to the PerBin Model takes the histograms bins as input and returns parameters values as output. For the Inverse Model the NN used is only one as shown schematically in Fig. 5.8. What the Inverse Model try to do is to learn the inverted model of the generator. So starting from the observed values the model try to reproduce the parameters values necessary to get the histograms we use as input.
The model is build and then trained with the training set introduced before. Once the model is trained feeding the experimental data to the NN this can try to infer the values of the parameters required to get the output.

Errors evaluation

The errors are evaluated in a different way respect to PerBin Model in fact in the Inverse Model there is not a minimization step, and the error is evaluated by a re-sampling of the experimental data using a *multivariate Gaussian Distribution*,

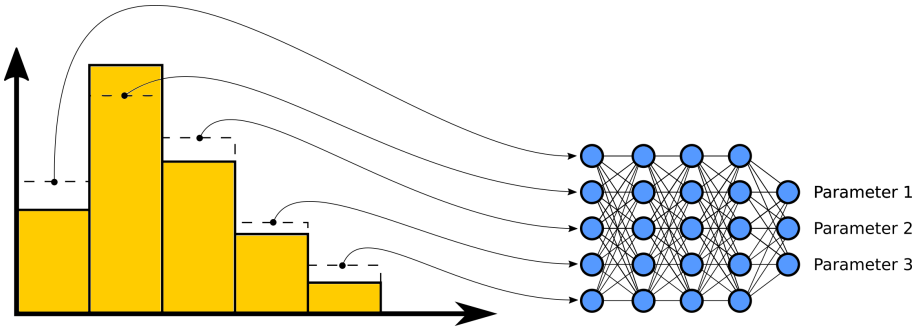

Figure 5.8: Figure from [43]

as the one in Eq. 5.11 with a diagonal covariance matrix that have experimental uncertainties on the main diagonal.

$$f(x_i; h^{(i)}_{\text{exp}}, \sigma^{(i)}_{\text{exp}}) = \mathcal{N} \cdot \exp\left[-\frac{1}{2}\sum_{j=1}^{N_{bins}} \frac{\left(x_i - h^{(i)}_{\text{exp}}\right)^2}{\sigma^{(i)\,2}_{\text{exp}}}\right] \tag{5.11}$$

So, a set of histograms is generated, then this is fed to the NN and a distribution of predictions is generated. An example is shown in Fig. 5.9, from this distribution one can compute the error by evaluating the standard deviation.

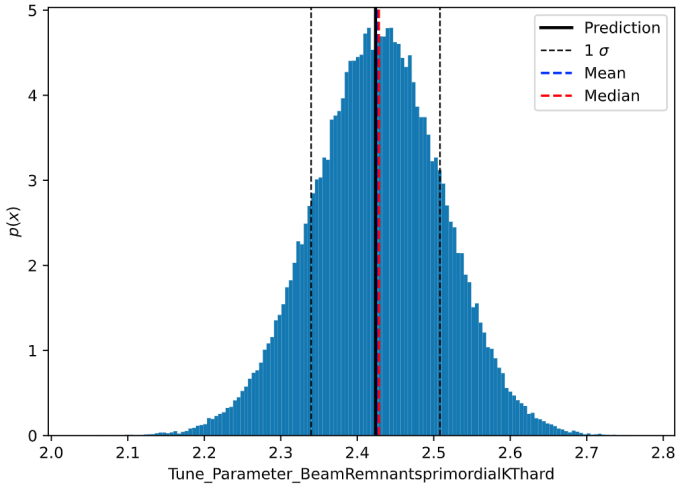

Figure 5.9: Predictions spread for the inverse model after that a Gaussian resample is performed

Note that this is a new method for the tune. As we are going to see this method requires more attentions than the PerBin Model to get it working correctly in the case of an high number of parameters to tune.
Respect to the PerBin Model this method is faster in the training step. The NN trained is only one and is not needed a minimization so a scan in the hyperparameters can be performed in order to search for the best architecture.

Hyperparameters

A really important step in the Inverse model is the *hyperparameter optimization* it is required to get the method working.

The procedure consist in build a *validation set* containing some Monte Carlo simulations as the training set (e.g. 10% of the simulations in the training set) and retrain the model with different choices for the NN architecture. Than a closure test is performed in order to estimate the performance of the NN. The test is performed using these validation set MC runs as input instead of the real experimental data, the closure test is done computing a loss function between the predicted value and the Monte Carlo truth defined as:

$$L = \sum_i \frac{\left| p_i^{\text{true}} - p_i^{\text{pred}} \right|}{p_i^{\text{true}}} \tag{5.12}$$

The configuration that minimize the distance between predicted and real values (MC truth), and so this loss function, is the best architecture in the selected hyperparaemters space.
Once the best model is found, it is retrained using the MC runs contained in both training and validation set. Then the experimental data are fed to it in order to get the best estimation for the parameters to tune. This procedure is schematically summarized in Fig. 5.10.
The hyperparameters scan is performed using the python package `hyperopt`.

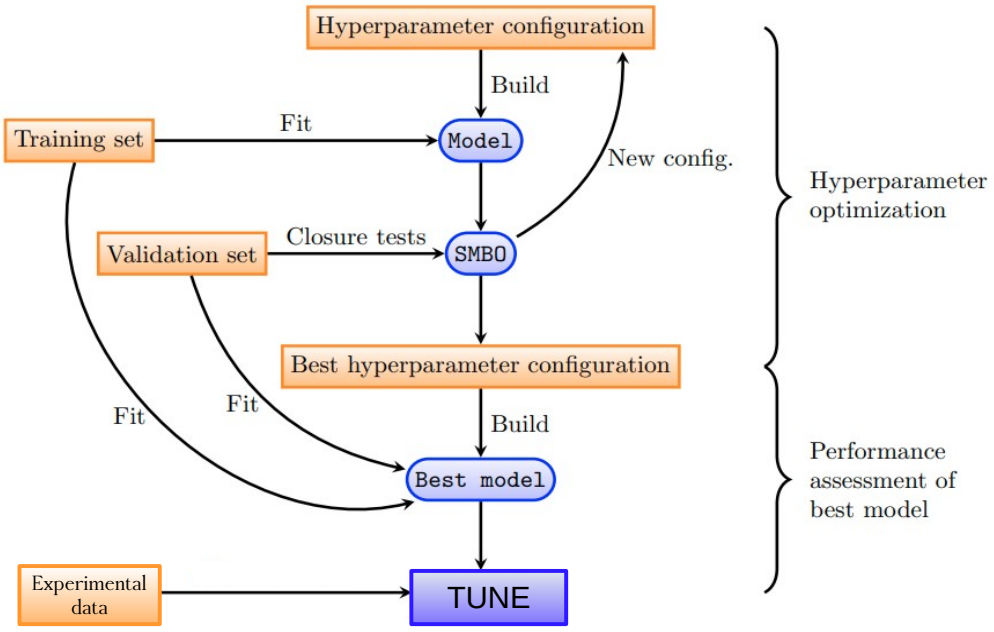

Figure 5.10: The hyperparameter optimization procedure is schematically shown here. The model is trained using a training set than performing a closure test a scan on the hyperparameter is done. Once the best configuration is found the best model is retrained using both the runs in the training set and the runs in the validation set. Then the experimental data are fed the network and the best parameters are estimated. Figure from [43]

.

Problems

As we are going to see the main problem for the Inverse model is the stability in the results. The operation that this model is trying to do is very hard, learn the inverse response of the generator is not an easy task. These difficulties in some case lead to a bad prediction.
As we are going to see in the chapter that describe our tune the Inverse model requires some more attention than the PerBin model to get it work properly and in some case when the number of parameters became larger and maybe the correlation between the parameters are important this model fails.
But if more care are given to this model this can become a very powerful method for future tunes.

5.3.5 Weightrules in MCNNTUNES

MCNNTUNES as PROFESSOR implement the possibility of change the weight of the singles bins in the distributions. This is a really useful feature. As we will discuss later we use these weightrules to get a results more similar to CP5 in our tune. This feature gives to us the possibility to decide which distribution is more important in our tune. Increasing the weight of a bin this bin became more important in the overall χ^2 evaluation and it is better described by the simulations. An example is shown in Fig. 5.11 where the application of the weightrules takes us to a better description of the experimental data.

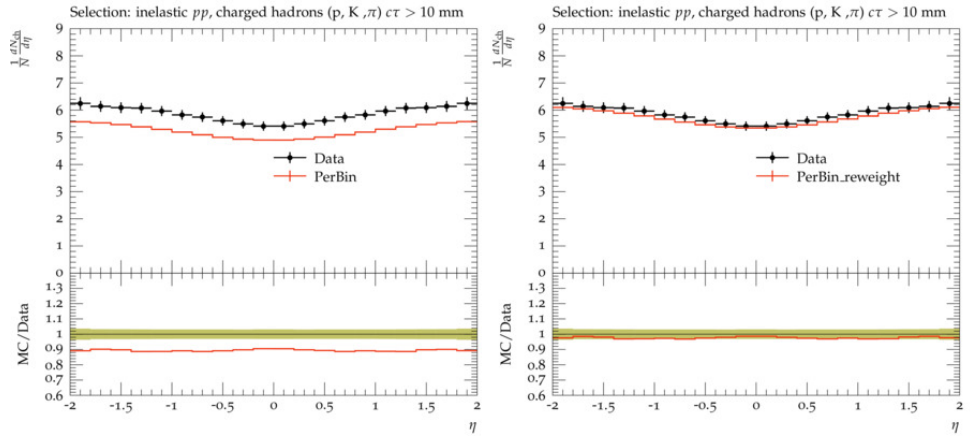

Figure 5.11: An example of weightrules application. Here the black point are the experimental data from the CMS analysis at $\sqrt{s} = 13$ TeV [37] while the colored lines are the simulations, the vertical lines on MC points indicate the statistical uncertainties. It is easy to see that the distribution in the right panel is not well described from the tune, but thanks to the weightrules we can give to this distribution a greater importance in the tune and describes it better as shown in the left panel. This is better explained in the next chapter.

Bibliography

[1] Steven Weinberg. A model of leptons, Nov 1967. URL https://link.aps.org/doi/10.1103/PhysRevLett.19.1264.

[2] J. D. Bjorken. Asymptotic Sum Rules at Infinite Momentum. *Phys. Rev.*, 179: 1547–1553, 1969. doi: 10.1103/PhysRev.179.1547.

[3] Sidney D Drell and Tung-Mow Yan. Partons and their applications at high energies, 1971. ISSN 0003-4916. URL https://www.sciencedirect.com/science/article/pii/0003491671900716.

[4] J M Campbell, J W Huston, and W J Stirling. Hard interactions of quarks and gluons: a primer for lhc physics, Dec 2006. ISSN 1361-6633. URL http://dx.doi.org/10.1088/0034-4885/70/1/R02.

[5] L N Lipatov. The parton model and perturbation theory, 1975. URL http://cds.cern.ch/record/400357.

[6] Vladimir Naumovich Gribov and L N Lipatov. Deep inelastic ep scattering in perturbation theory, 1972. URL https://cds.cern.ch/record/427157.

[7] G. Altarelli and G. Parisi. Asymptotic freedom in parton language, 1977. ISSN 0550-3213. URL https://www.sciencedirect.com/science/article/pii/0550321377903844.

[8] Yuri L. Dokshitzer. Calculation of the Structure Functions for Deep Inelastic Scattering and e+ e- Annihilation by Perturbation Theory in Quantum Chromodynamics., 1977.

[9] W.J. Stirling. private communication. URL http://www.hep.ph.ic.ac.uk/~wstirlin/plots/plots.html.

[10] F. Bloch and A. Nordsieck. Note on the radiation field of the electron, Jul 1937. URL https://link.aps.org/doi/10.1103/PhysRev.52.54.

[11] Toichiro Kinoshita. Mass singularities of feynman amplitudes, 1962. URL https://doi.org/10.1063/1.1724268.

[12] T. D. Lee and M. Nauenberg. Degenerate systems and mass singularities, Mar 1964. URL https://link.aps.org/doi/10.1103/PhysRev.133.B1549.

[13] A. Kulesza, G. Sterman, and W. Vogelsang. Electroweak vector boson production in joint resummation, 2002. URL https://arxiv.org/abs/hep-ph/0207148.

[14] Torbjörn Sjöstrand, Stefan Ask, Jesper R. Christiansen, Richard Corke, Nishita Desai, Philip Ilten, Stephen Mrenna, Stefan Prestel, Christine O. Rasmussen, and Peter Z. Skands. An introduction to pythia 8.2, Jun 2015. ISSN 0010-4655. URL http://dx.doi.org/10.1016/j.cpc.2015.01.024.

[15] Manuel Bähr, Stefan Gieseke, Martyn A. Gigg, David Grellscheid, Keith Hamilton, Oluseyi Latunde-Dada, Simon Plätzer, Peter Richardson, Michael H. Seymour, Alexander Sherstnev, and Bryan R. Webber. Herwig++ physics and manual, Nov 2008. ISSN 1434-6052. URL http://dx.doi.org/10.1140/epjc/s10052-008-0798-9.

[16] T Gleisberg, S Hoeche, F Krauss, A Schaelicke, S Schumann, and J Winter. Sherpa 1. , a proof-of-concept version, Feb 2004. ISSN 1029-8479. URL http://dx.doi.org/10.1088/1126-6708/2004/02/056.

[17] E. Boos, M. Dobbs, W. Giele, I. Hinchliffe, J. Huston, V. Ilyin, J. Kanzaki, K. Kato, Y. Kurihara, L. Lonnblad, M. Mangano, S. Mrenna, F. Paige, E. Richter-Was, M. Seymour, T. Sjostrand, B. Webber, and D. Zeppenfeld. Generic user process interface for event generators, 2001.

[18] Stefano Catani, Frank Krauss, Bryan R Webber, and Ralf Kuhn. Qcd matrix elements + parton showers. *Journal of High Energy Physics*, 2001(11):063–063, Nov 2001. ISSN 1029-8479. doi: 10.1088/1126-6708/2001/11/063. URL http://dx.doi.org/10.1088/1126-6708/2001/11/063.

[19] Stefano Frixione and Bryan R Webber. Matching nlo qcd computations and parton shower simulations. *Journal of High Energy Physics*, 2002(06):029–029, Jun 2002. ISSN 1029-8479. doi: 10.1088/1126-6708/2002/06/029. URL http://dx.doi.org/10.1088/1126-6708/2002/06/029.

[20] Stefano Frixione, Paolo Nason, and Bryan R Webber. Matching nlo qcd and parton showers in heavy flavour production. *Journal of High Energy Physics*, 2003(08):007–007, Aug 2003. ISSN 1029-8479. doi: 10.1088/1126-6708/2003/08/007. URL http://dx.doi.org/10.1088/1126-6708/2003/08/007.

[21] Stefano Frixione and Bryan R. Webber. The mc@nlo 3.1 event generator, 2005.

[22] Rikkert Frederix and Stefano Frixione. Merging meets matching in MC@NLO. *JHEP*, 12:061, 2012. doi: 10.1007/JHEP12(2012)061.

[23] Victor S. Fadin. BFKL resummation. *Nucl. Phys. A*, 666:155–164, 2000. doi: 10.1016/S0375-9474(00)00022-1.

[24] I. I. Balitsky and L. N. Lipatov. The Pomeranchuk Singularity in Quantum Chromodynamics. *Sov. J. Nucl. Phys.*, 28:822–829, 1978.

[25] Richard D. Ball, Valerio Bertone, Stefano Carrazza, Luigi Del Debbio, Stefano Forte, Patrick Groth-Merrild, Alberto Guffanti, Nathan P. Hartland, Zahari Kassabov, José I. Latorre, Emanuele R. Nocera, Juan Rojo, Luca Rottoli,

Emma Slade, and Maria Ubiali. Parton distributions from high-precision collider data. *The European Physical Journal C*, 77(10), Oct 2017. ISSN 1434-6052. doi: 10.1140/epjc/s10052-017-5199-5. URL http://dx.doi.org/10.1140/epjc/s10052-017-5199-5.

[26] FRANK SIEGERT. Monte-carlo event generation for the lhc, 2010. URL http://etheses.dur.ac.uk/484/.

[27] B. Andersson, G. Gustafson, G. Ingelman, and T. Sjöstrand. Parton fragmentation and string dynamics. *Physics Reports*, 97(2):31–145, 1983. ISSN 0370-1573. doi: https://doi.org/10.1016/0370-1573(83)90080-7. URL https://www.sciencedirect.com/science/article/pii/0370157383900807.

[28] Torbjorn Sjostrand. Jet Fragmentation of Nearby Partons. *Nucl. Phys. B*, 248: 469–502, 1984. doi: 10.1016/0550-3213(84)90607-2.

[29] Underlying Event Measurements with Leading Particles and Jets in pp collisions at $\sqrt{s} = 13$ TeV. Technical report, CERN, Geneva, 2015. URL https://cds.cern.ch/record/2104473.

[30] Serguei Chatrchyan et al. Measurement of the underlying event in the Drell-Yan process in proton-proton collisions at $\sqrt{s} = 7$ TeV. *Eur. Phys. J. C*, 72: 2080, 2012. doi: 10.1140/epjc/s10052-012-2080-4.

[31] A. M. Sirunyan et al. Measurement of the underlying event activity in inclusive Z boson production in proton-proton collisions at $\sqrt{s} = 13$ TeV. *JHEP*, 07:032, 2018. doi: 10.1007/JHEP07(2018)032.

[32] Albert M. Sirunyan et al. Study of the underlying event in top quark pair production in pp collisions at 13 TeV. *Eur. Phys. J. C*, 79(2):123, 2019. doi: 10.1140/epjc/s10052-019-6620-z.

[33] Florian Bechtel. *The underlying event in proton-proton collisions*. PhD thesis, Hamburg U., 2009.

[34] Albert M Sirunyan et al. Extraction and validation of a new set of CMS PYTHIA8 tunes from underlying-event measurements. *Eur. Phys. J. C*, 80(1):4, 2020. doi: 10.1140/epjc/s10052-019-7499-4.

[35] A. Banfi, S. Redford, M. Vesterinen, P. Waller, and T. R. Wyatt. Optimisation of variables for studying dilepton transverse momentum distributions at hadron colliders. *Eur. Phys. J. C*, 71:1600, 2011. doi: 10.1140/epjc/s10052-011-1600-y.

[36] Albert M Sirunyan et al. Measurements of differential Z boson production cross sections in proton-proton collisions at $\sqrt{s} = 13$ TeV. *JHEP*, 12:061, 2019. doi: 10.1007/JHEP12(2019)061.

[37] Vardan Khachatryan et al. Pseudorapidity distribution of charged hadrons in proton-proton collisions at $\sqrt{s} = 13$ TeV. *Phys. Lett. B*, 751:143–163, 2015. doi: 10.1016/j.physletb.2015.10.004.

[38] Timo Antero Aaltonen et al. Study of the energy dependence of the underlying event in proton-antiproton collisions. *Phys. Rev. D*, 92(9):092009, 2015. doi: 10.1103/PhysRevD.92.092009.

[39] Measurement of the Underlying Event Activity at the LHC at 7 TeV and Comparison with 0.9 TeV. Technical report, CERN, Geneva, 2012. URL http://cds.cern.ch/record/1478982.

[40] Albert M. Sirunyan et al. Measurement of charged particle spectra in minimum-bias events from proton–proton collisions at $\sqrt{s} = 13\,\text{TeV}$. *Eur. Phys. J. C*, 78 (9):697, 2018. doi: 10.1140/epjc/s10052-018-6144-y.

[41] Andy Buckley, Hendrik Hoeth, Heiko Lacker, Holger Schulz, and Jan Eike von Seggern. Systematic event generator tuning for the LHC. *Eur. Phys. J. C*, 65: 331–357, 2010. doi: 10.1140/epjc/s10052-009-1196-7.

[42] Stefano Carrazza and Marco Lazzarin. N3pdf/mcnntunes: mcnntunes 0.1.0, October 2020. URL https://doi.org/10.5281/zenodo.4071125.

[43] Marco Lazzarin, Simone Alioli, and Stefano Carrazza. MCNNTUNES: Tuning Shower Monte Carlo generators with machine learning. *Comput. Phys. Commun.*, 263:107908, 2021. doi: 10.1016/j.cpc.2021.107908.

[44] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

[45] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. 1958. URL https://doi.org/10.1037/h0042519.

[46] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991. ISSN 0893-6080. doi: https://doi.org/10.1016/0893-6080(91)90009-T. URL https://www.sciencedirect.com/science/article/pii/089360809190009T.

[47] Moshe Leshno, Vladimir Ya. Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6(6):861–867, 1993. ISSN 0893-6080. doi: https://doi.org/10.1016/S0893-6080(05)80131-5. URL https://www.sciencedirect.com/science/article/pii/S0893608005801315.

[48] Nikolaus Hansen. The CMA evolution strategy: A tutorial. *CoRR*, abs/1604.00772, 2016. URL http://arxiv.org/abs/1604.00772.

[49] G. Cowan. *Statistical data analysis*. Oxford University Press, USA, 1998.

[50] Richard D. Ball et al. Parton distributions from high-precision collider data. *Eur. Phys. J. C*, 77(10):663, 2017. doi: 10.1140/epjc/s10052-017-5199-5.

[51] Albert M Sirunyan et al. Measurement of differential cross sections for Z boson production in association with jets in proton-proton collisions at $\sqrt{s} = 13$ TeV. *Eur. Phys. J. C*, 78(11):965, 2018. doi: 10.1140/epjc/s10052-018-6373-0.