Computer Physics Communications 263 (2021) 107908

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

MCNNTUNES: Tuning Shower Monte Carlo generators with machine learning☆,☆☆

Marco Lazzarin [a], Simone Alioli [b], Stefano Carrazza [a,*]

[a] TIF Lab, Dipartimento di Fisica, Università degli Studi di Milano and INFN Sezione di Milano, Milan, Italy
[b] Dipartimento di Fisica, Università degli Studi di Milano Bicocca and INFN Sezione di Milano Bicocca, Milan, Italy

ARTICLE INFO

Article history:
Received 16 October 2020
Received in revised form 22 January 2021
Accepted 3 February 2021
Available online 20 February 2021

Keywords:
Event generator tuning
Machine learning

ABSTRACT

The parameters tuning of event generators is a research topic characterized by complex choices: the generator response to parameter variations is difficult to obtain on a theoretical basis, and numerical methods are hardly tractable due to the long computational times required by generators. Event generator tuning has been tackled by parametrization-based techniques, with the most successful one being a polynomial parametrization. In this work, an implementation of tuning procedures based on artificial neural networks is proposed. The implementation was tested with closure testing and experimental measurements from the ATLAS experiment at the Large Hadron Collider.

Program summary

Program Title: MCNNTUNES
CPC Library link to program files: https://doi.org/10.17632/dmkydsxgd3.1
Developer's repository link: https://github.com/N3PDF/mcnntunes
Licensing provisions: GPLv3
Programming language: Python
Nature of problem: Shower Monte Carlo generators introduce many parameters that must be tuned to reproduce the experimental measurements. The dependence of the generator output on these parameters is difficult to obtain on a theoretical basis.
Solution method: Implementation of a tuning method using supervised machine learning algorithms based on neural networks, which are universal approximators.

1. Introduction

Shower Monte Carlo (SMC) event generators are tools that apply shower algorithms to simulated collisions of particles at high energies. They introduce many parameters, mainly due to the usage of phenomenological models, like the hadronization model or the underlying event model, needed to describe the low-energy limit of quantum chromodynamics (QCD) which is not easily calculable from first principles. These parameters are difficult to obtain on a theoretical basis, so they must be carefully tuned in order to make the generators reproduce the experimental measurements. The procedure of estimating the best value for each parameter is called event generator tuning.

This tuning procedure is made more difficult by the high computational cost of running a generator, so it requires methods to study the dependence between a generator output and its parameters. Moreover, since the observables considered while analysing the generator output play a pivotal role in determining the tuning, one needs to model this dependence for different observables at the same time.

The current state-of-the-art tuning procedure is based on a polynomial parametrization of the generator response to parameter variations, followed by a numerical fit of the parametrized behaviour to experimental data. This is the procedure which is implemented in `Professor` [1], the primary tool for SMC event generator tuning at the Large Hadron Collider (LHC). However, the assumption that the dependence of the generator output on its parameters is polynomial is not always justified.

This paper investigates new tuning procedures based on artificial neural networks. Artificial neural networks are universal function approximators [2–4], providing accurate predictive models with a finite number of parameters, as shown in early attempts [5]. Two different tuning procedures are presented, called *Per Bin* and *Inverse* from now on. The former follows the same approach of `Professor`, but with a different parametrization model made of fully-connected neural networks and a different

☆ The review of this paper was arranged by Prof. Z. Was.
☆☆ This paper and its associated computer program are available via the Computer Physics Communication homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).

* Corresponding author.
E-mail address: stefano.carrazza@unimi.it (S. Carrazza).

https://doi.org/10.1016/j.cpc.2021.107908
0010-4655/

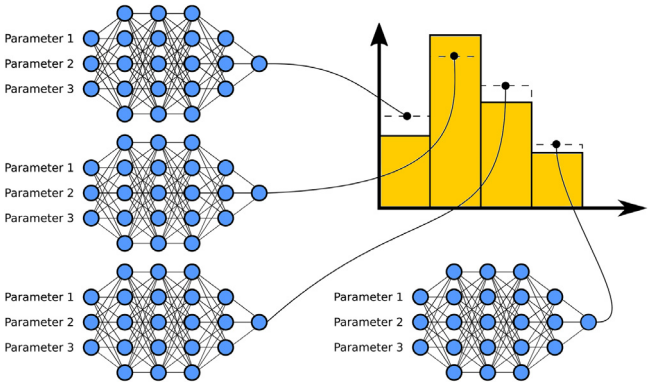

Fig. 1. An illustration of the parametrization of the generator response as implemented in the Per Bin model.

minimization algorithm: the evolutionary algorithm CMA-ES [6]. The latter takes a completely different approach: by using a fully-connected neural network, it learns to predict directly the parameters that the generator needs to output a given result. These two procedures were implemented in the `Python` package `MCNNTUNES` [7] and then tested with the event generator `PYTHIA8` [8]. Two different datasets of Monte Carlo runs were generated, with three and four tunable parameters respectively. The procedures were tested with closure tests and with real experimental data taken from the ATLAS experiment [9,10].

A description of the procedures and their technical implementation is presented in Section 2, while Section 3 contains the details of the testing phase. Finally, the conclusion of this work and future development directions are presented in Section 4.

2. Implementation

`MCNNTUNES` implements two different strategies for generator tuning, both based on feedforward neural networks. In this section these two strategies are presented in detail, along with a description of their technical implementation.

2.1. Per Bin model

The first strategy is a parametrization-based method similar to `Professor` [1]. The work cycle is divided in three consecutive steps: dataset generation, parametrization of the generator output, and the actual tuning step.

At first, a dataset of Monte Carlo runs is generated by sampling parameter configurations from the parameter space, and then running the generator with each configuration. This step is identical to the one in `Professor`. A standard choice for the distributions used to sample the parameter configurations are uniform distributions within some intervals chosen by the user. The boundaries reflect the user's prior belief about the best configuration.

Then, the generator response to parameter variations is parametrized one bin at a time, using the previously created dataset. The parametrization is tackled by feedforward neural networks, which take the parameters as input and return the value of a single bin. An independent neural network is used for each bin[1] (see Fig. 1 for an illustration). The models are trained with a gradient-based algorithm, as usual for feedforward neural networks, with mean squared error as loss. The details of the architecture, the choice of the optimization algorithm and its settings are all configurable by the user.

[1] The hyperparameters are the same for each bin, only the parameters of the biases and the kernels are different.

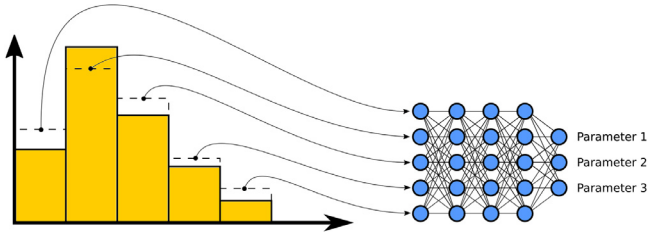

Fig. 2. An illustration of the Inverse model strategy.

Finally, the tuning step exploits the parametric model of the generator to define a surrogate loss function for the tuning problem. In fact, the parametrization step creates a model $h^{(i)}(\mathbf{p})$ of the generator, where $\mathbf{p}$ is the vector of parameters and $h^{(i)}$ the value of the ith bin of the output. It enables the prediction of the generator output given a generic parameter configuration. The quality of the prediction will depend on the quality of the parametrization. The optimization problem underlying the tuning can then be solved (approximately) by minimizing a surrogate loss function, e.g.

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

The possibility of weighting each bin differently in the χ^2 is also implemented. Then, the minimization of this χ^2 is performed with the CMA-ES algorithm, which is a stochastic optimization method for non-linear non-convex functions. The values of the parameters that minimize the χ^2 determine the best tune. The minimization task can be carried out also with gradient-based optimizers. This possibility is implemented as an optional feature. A first test showed it performed worse than CMA-ES, but additional tests with a careful tuning of the hyperparameters may change this result.

2.2. Inverse model

The previous strategy involved models with parameters as input and histograms as output: given a set of parameters, it returns the histograms related to some observables. This is the input/output structure of a generator. In contrast, the Inverse model tries to learn the inverted model of a generator, using a feedforward neural network with the bin values as input layer and with the generator parameters as output layer (see Fig. 2 for an illustration). In case of success, the model is able to predict the parameters used for the generator given its results. Then, tuning the generator consists in feeding the experimental data into the model and inferring the parameters that the generator needs to reproduce them. The uncertainties of the predictions are computed in three steps:

1. At first, the experimental data are resampled many times by using a multivariate Gaussian centred around the actual measurement, with a diagonal covariance matrix that includes the data uncertainties:
$$\text{norm} \cdot \exp\left(-\frac{1}{2}\sum_{j=1}^{N_{\text{bins}}}\left((x_j - h_{j,\text{exp}})^2/\sigma^2_{j,\text{exp}}\right)\right)$$
2. This set of histograms is fed into the neural network.
3. The output of the network is a distribution of predictions for each parameter, and the uncertainties are computed as the standard deviations of these distributions.

2.3. Data augmentation

During the training of the Inverse model the output variables (the parameters) are exact, but the input variables (the histogram bins) have a known uncertainty. In order to exploit this information, the training with jitter [11] method was implemented as an optional feature: at each training epoch, the entire dataset is resampled following the data uncertainty. More precisely, let $\mathbf{X}$ be the dataset matrix where each row is a Monte Carlo run, and each column is the value of a bin, and let σ_{ij} be the corresponding error for each element X_{ij} of $\mathbf{X}$. Then, the training is done by replacing $\mathbf{X}$ with $\overline{\mathbf{X}}$ such that each element $\overline{X}_{ij}$ is a random variable distributed according to a Gaussian with mean X_{ij} and variance σ_{ij}^2. $\overline{\mathbf{X}}$ is resampled at each epoch. This resembles a Gaussian noise layer applied to the input layer, but here the σ of the Gaussian noise is different for each node of the input layer, and for each element of the training set. This method can be seen as a form of regularization, as proven in [11].

2.4. Performance assessment

The program implements a performance assessment procedure based on closure tests. A single closure test consists in using one Monte Carlo run as the experimental data, and then performing the tune; in this way, the obtained tunes can be directly compared with the real parameters used to generate that run. Notice that run must be excluded from the training set, otherwise the result does not measure the ability of the procedure to generalize to new examples. The user can provide two different datasets of Monte Carlo runs: a training set, used to train the model, and a validation set, used to perform closure tests. Once the model has been trained, a closure test is performed for each run in the validation set, and a loss function defined as

$$L = \sum_i \frac{|p_i^{\text{true}} - p_i^{\text{pred}}|}{p_i^{\text{true}}} \tag{2}$$

is computed. The average of the losses of all closure tests is used as validation loss. This loss could be interpreted as an estimator of the accuracy of the tuning procedure, even though it is unsatisfactory: the experimental data and the Monte Carlo runs are not identically distributed nor generated by the same data-generating underlying process. In fact, the generator may be unable to represent the experimental data at all. However, this loss could be used to tune the hyperparameters of the model, or to compare different models.

2.5. Hyperparameter tuning

`MCNNTUNES` features a hyperparameter search procedure implemented with `Hyperopt` [12,13]. `Hyperopt` is a library dedicated to the hyperparameter optimization of machine learning algorithms. In particular, it takes care of scalar-valued functions whose arguments are defined over a search space with a potentially complicated structure: some arguments could be real-valued (e.g. learning rates), others could be discrete (e.g. the choice of the optimization algorithm), and the search space could be tree-structured, i.e. some variables are defined only when other parent variables take on a specific value (e.g. the number of hidden layers and the number of units of each hidden layer).

The implementation of `Hyperopt` requires the definition of a search space and the definition of the function to minimize (the objective function). The search space is provided by the user in the configuration of `MCNNTUNES`, using the format specified in the documentation of `Hyperopt`.

The objective function must receive the sampled hyperparameters as input, create a model with these hyperparameters, train it, and evaluate the validation loss of that model, or at least some sort of performance measurement with the "lower is better" format. `MCNNTUNES` computes the performance measurement presented in Section 2.4 as validation loss, provided that a valid validation set of Monte Carlo runs is available. Specifically, it trains the model on the training set, then performs a closure test for every run in the validation set, computes the loss in Eq. (2) for each of them, and finally returns the average of these losses as validation loss. `Hyperopt` implements two different algorithms: a random search, and a Sequential Model-Based Optimization (SMBO) algorithm called Tree-structured Parzen Estimator [14]. `MCNNTUNES` uses the latter.

Moreover, `MCNNTUNES` supports the parallel search as implemented in `Hyperopt`.

2.6. Technical details

The program is written in `Python`. The Monte Carlo runs (histograms) are loaded with the `YODA` library [15], which is the default histogram format of `Rivet` [16]. The basic operations are implemented with `NumPy` [17], while the machine learning aspects use `Keras` [18], with the `TensorFlow` framework [19] as backend. It uses the `pycma` package [20] for the CMA-ES algorithm. The procedures are implemented in the `mcnntunes` script. The script accepts a configuration runcard in `YAML` format, which contains all program settings. This is the basic work cycle:

1. `mcnntunes preprocess` loads the Monte Carlo runs and the experimental data, transforms the training set so that each input or output has mean 0 and variance 1, computes some useful statistics and saves all the data for future use.
2. `mcnntunes model` trains the model specified in the runcard, and saves it for future use.
3. `mcnntunes tune` performs the tune with the experimental data, and generates an `HTML` report with some information about the whole tuning process.

Some additional features are useful for performance assessment (`mcnntunes benchmark`) and hyperparameter tuning (`mcnntunes optimize`).

3. Results

This section presents the testing phase of `MCNNTUNES`. The choice of the generator, the parameters with their variation ranges, the process and the observables on which performing the tunes were chosen following the AZ tune [9] as reference. This should be considered only as a study of the efficiency and reliability of the `MCNNTUNES` approach for some specific observables and data and not as an attempt to devise a new exhaustive tune for the LHC.

The generation of some datasets of Monte Carlo runs is presented in Section 3.1; a systematic performance assessment is presented in Section 3.2; finally, an AZ-like tune that tries to reproduce some results of [9] is presented in Section 3.3.

3.1. Datasets

The generation of the datasets followed the procedure presented in [9]. The Monte Carlo runs were generated with `PYTHIA` version `8.240` [8], interfaced with the `Rivet` [16] package, version `2.7.0`. Two different analyses were performed: one involved the measurement of the Z/γ^* boson transverse momentum distribution p_T^Z in pp collisions at $\sqrt{s} = 7$ TeV [9] (analysis `ATLAS_2014_I1300647`), the other involved the measurement of angular correlation $\phi_\eta^\star$ [10] (analysis `ATLAS_2012_I1204784`),

Table 1
PYTHIA8 setup and variation ranges.

Parameter	Dataset 3P	Dataset 4P
Primordial k_T [GeV]	1.0–2.5	1.0–2.5
ISR $\alpha_S(m_Z^2)$	0.120–0.140	0.120–0.140
ISR $p_{T,0}^{\text{ref}}$ [GeV]	0.5–2.5	0.5–2.5
MPI $p_{T,0}^{\text{ref}}$ [GeV]	2.18 (fixed)	1.9–2.2
PYTHIA8 base tune	Tune 4C [21]	Tune 4C [21]
Number of events	$4 \cdot 10^6$	$4 \cdot 10^6$
Number of runs	512	1280

which probes the same physics of p_T^Z but with higher experimental resolution. Thus, the activated process was $f\bar{f} \to Z/\gamma^*$. The investigated parameters are the primordial k_T, the parton shower $\alpha_S(m_Z^2)$ and the parton shower damping factor for the lower p_T cut-off (both for the initial state radiation, ISR from now on), and the damping factor for the lower p_T cut-off for the multiparton interaction. Two different datasets were generated: one, the most similar to [9], fixes the multiparton interaction parameter, while the other does not. The former will be the dataset 3P from now on, while the latter will be called 4P. Each parameter is sampled uniformly within a range. The variation ranges and the setup of PYTHIA8 are presented in Table 1.

The tunes in this section rely in the reconstruction of the vector boson properties by combining dressed leptons, as defined in the aforementioned Rivet analyses. We limit ourselves to consider only distributions inclusive in rapidity. The tunes are performed only for $p_T^Z < 26$ GeV and $\phi_\eta^\star < 0.29$, unless "all bins" is specified. Three different sets of measurements are selected: one with only p_T^Z measurements, another with $\phi_\eta^\star$ measurements, and one using only the muon channel p_T^Z measurement and the electron channel $\phi_\eta^\star$ measurement.

3.2. Performance measurements

This subsection presents some performance measurements on Professor and MCNNTUNES. The procedure is described as follows.

At first, the dataset is split in training (80%), validation (10%) and test set (10%).

Then, a hyperparameter optimization of the model is performed by training each hyperparameter configuration on the training set and selecting the configuration with the best loss computed on the validation set. The loss function is the one presented in Section 2.4. For Professor, the hyperparameter search consisted in a grid search for the polynomial order. In practice, the best order was obtained by trying polynomials with degree from one to seven. The other options were kept at their default values, except the options `-s 2 --scan-n=100` for `prof2-tune`. For MCNNTUNES, the hyperparameter search was performed by running Hyperopt, feeding it with the validation loss. The Hyperopt configurations were the one in Table 2 plus another one focused on the architecture only, presented in Table 3. Whether using data augmentation or not, instead, was chosen by performing a grid search on top of the Hyperopt scan.

Finally, the best model is retrained on both the training and the validation set, and its performance is evaluated by closure testing on the test set. A schematic view of the workflow is shown in Fig. 3.

The results are presented in Table 4, for the Inverse model only. The performance of MCNNTUNES turns out to be slightly better than Professor, on average. Results are however limited to this particular benchmark, and may change with different random seeds, losses, datasets, parameters and observables.

Table 2
Hyperopt configuration for the Inverse model - broad search.

Hyperparameter	Variation range
# hidden layers	2–5
Units per layer	2–20
Activation function	tanh, relu, sigmoid
Optimizer	Various Keras optimizers
Epochs	250–10 000 in discrete steps
Batch size	100, 200, 300, 400, 500
Number of trials	1000

Table 3
Hyperopt configuration for the Inverse model - architecture only.

Hyperparameter	Variation range
# hidden layers	3–4
Units per layer	10–50 in step of 2
Activation function	Sigmoid
Optimizer	Adam [22]
Optimizer learning rate	Default value
Initializer	Glorot uniform [23]
Epochs	2500–15 000 in steps of 500
Batch size	128
Number of trials	1000

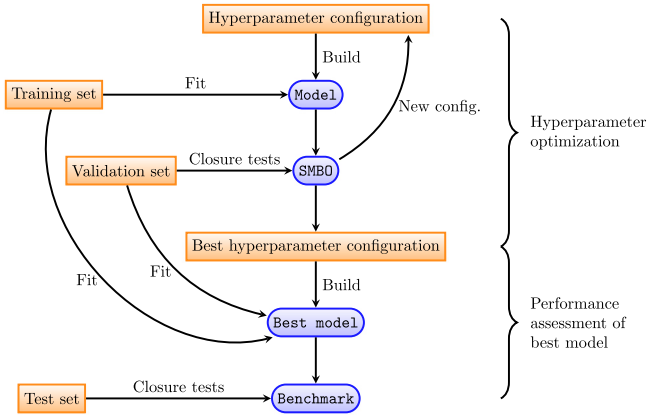

Fig. 3. Illustration of the performance assessment procedure.

Moreover, this benchmark uses Monte Carlo runs, and not experimental data, so this performance measurement will not estimate the real tuning precision with real experimental data, because experimental data and Monte Carlo runs are not drawn from the same underlying data-generating distribution function. Due to computation time constraints, performance measurements for the Per Bin model were limited to restricted observables and datasets, without validation-test split, but they showed solid results (see Table 5).

3.3. Tunes

Finally, the datasets 3P and 4P were used to perform some final tunes. The hyperparameter configurations are the ones selected in the hyperparameter tuning step of Section 3.2, except for the Per Bin model for which the hyperparameters were chosen manually. The whole datasets were used for training. The results are presented in Tables 6 and 7 for the 3P and 4P dataset respectively. Tunes obtained with the Per Bin model have no errors because no proper error estimation is implemented. A few comments on these tunes may be made:

- Professor and the Per Bin model give similar results, usually compatible with each other.

Table 4
Test errors - Inverse model against `Professor` (polynomial order inside parentheses).

Observables	Inverse (%)	`Professor` (%)
$\phi_\eta^\star$, 3P	3.7 ± 0.5	4.8 ± 0.7 (3)
p_T^Z, 3P	2.6 ± 0.3	3.1 ± 0.5 (3)
$\phi_\eta^\star p_T^Z$, 3P	3.2 ± 0.4	3.6 ± 0.5 (3)
$\phi_\eta^\star$, 3P, all bins	4.4 ± 0.6	4.2 ± 0.7 (3)
p_T^Z, 3P, all bins	2.5 ± 0.3	2.6 ± 0.4 (3)
$\phi_\eta^\star p_T^Z$, 3P, all bins	3.1 ± 0.4	3.1 ± 0.5 (3)
$\phi_\eta^\star$, 4P	3.5 ± 0.2	4.6 ± 0.3 (5)
p_T^Z, 4P	2.71 ± 0.16	3.28 ± 0.18 (5)
$\phi_\eta^\star p_T^Z$, 4P	3.2 ± 0.2	3.8 ± 0.3 (4)
$\phi_\eta^\star$, 4P, all bins	3.7 ± 0.2	4.0 ± 0.3 (4)
p_T^Z, 4P, all bins	2.89 ± 0.15	3.2 ± 0.2 (4)
$\phi_\eta^\star p_T^Z$, 4P, all bins	3.0 ± 0.2	3.4 ± 0.2 (4)

- The Inverse model sometimes gives different results: the agreement with `Professor` is usually good for primordial k_T and $\alpha_S^{ISR}(m_Z^2)$ parameters, with some exceptions. The other parameters are harder to analyse and will be discussed in the next points.
- The estimation of the MPI parameter by the Inverse model does not work: the model predicts always a parameter near the midpoint of the variation range, i.e. it is a trivial predictor. This did not prevent the Inverse model to perform better than `Professor` on closure testing, which means that both algorithms fail, and the trivial prediction is just the way the learning algorithm found to minimize the training loss.
- The choice of the variation ranges used to generate the dataset strongly affects the quality of the procedure, because the extrapolation outside them may be unreliable. Unfortunately, many results suggest that the best value for ISR

Table 5
Validation losses (with smaller datasets). The hyperparameter search space was similar to the one of Table 2 but without configurations with five hidden layers.

Observables	Per bin (%)	`Professor` (%)
$\phi_\eta^\star$, 3P	3.1 ± 0.5	3.6 ± 0.6
$\phi_\eta^\star$, 4P	4.0 ± 0.3	4.0 ± 0.3

Table 6
Tunes using dataset 3P.

`Professor`			
Parameter	p_T^Z	$\phi_\eta^\star$	p_T^Z $\phi_\eta^\star$
Primordial k_T [GeV]	1.77 ± 0.04	1.80 ± 0.04	1.77 ± 0.04
ISR $\alpha_S(m_Z^2)$	0.1232 ± 0.0002	0.1236 ± 0.0002	0.1236 ± 0.0002
ISR $p_{T,0}^{\text{ref}}$ [GeV]	Left bound	Left bound	Left bound
`MCNNTUNES`, Per Bin model			
Parameter	p_T^Z	$\phi_\eta^\star$	p_T^Z $\phi_\eta^\star$
Primordial k_T [GeV]	1.75	1.76	1.74
ISR $\alpha_S(m_Z^2)$	0.1233	0.1236	0.1236
ISR $p_{T,0}^{\text{ref}}$ [GeV]	Left bound	Left bound	Left bound
`MCNNTUNES`, Inverse model			
Parameter	p_T^Z	$\phi_\eta^\star$	p_T^Z $\phi_\eta^\star$
Primordial k_T [GeV]	1.75 ± 0.05	1.81 ± 0.05	1.77 ± 0.04
ISR $\alpha_S(m_Z^2)$	0.1249 ± 0.0006	0.1233 ± 0.0004	0.1241 ± 0.0005
ISR $p_{T,0}^{\text{ref}}$ [GeV]	0.9 ± 0.2	0.24 ± 0.18	0.8 ± 0.2

Table 7
Tunes using dataset 4P.

`Professor`			
Parameter	p_T^Z	$\phi_\eta^\star$	p_T^Z $\phi_\eta^\star$
Primordial k_T [GeV]	1.76 ± 0.05	1.80 ± 0.05	1.79 ± 0.04
ISR $\alpha_S(m_Z^2)$	0.1233 ± 0.0003	0.1237 ± 0.0002	0.1236 ± 0.0002
ISR $p_{T,0}^{\text{ref}}$ [GeV]	Left bound	Left bound	0.5 ± 1.9
MPI $p_{T,0}^{\text{ref}}$ [GeV]	2.11 ± 0.06	2.13 ± 0.07	Right bound
`MCNNTUNES`, Per Bin model			
Parameter	p_T^Z	$\phi_\eta^\star$	p_T^Z $\phi_\eta^\star$
Primordial k_T [GeV]	1.70	1.76	1.76
ISR $\alpha_S(m_Z^2)$	0.1233	0.1237	0.1236
ISR $p_{T,0}^{\text{ref}}$ [GeV]	Left bound	Left bound	Left bound
MPI $p_{T,0}^{\text{ref}}$ [GeV]	1.95	Right bound	Right bound
`MCNNTUNES`, Inverse model			
Parameter	p_T^Z	$\phi_\eta^\star$	p_T^Z $\phi_\eta^\star$
Primordial k_T [GeV]	1.69 ± 0.07	1.69 ± 0.04	1.60 ± 0.06
ISR $\alpha_S(m_Z^2)$	0.1246 ± 0.0007	0.12345 ± 0.00018	0.1238 ± 0.0007
ISR $p_{T,0}^{\text{ref}}$ [GeV]	0.9 ± 0.2	0.29 ± 0.09	0.6 ± 0.2
MPI $p_{T,0}^{\text{ref}}$ [GeV]	2.0468 ± 0.0011	2.0431 ± 0.0009	2.0450 ± 0.0009

Table 8
Tunes using dataset 3P, all bins.

`Professor`			
Parameter	p_T^Z	$\phi_\eta^\star$	p_T^Z $\phi_\eta^\star$
Primordial k_T [GeV]	1.82 ± 0.05	1.79 ± 0.04	1.74 ± 0.04
ISR $\alpha_S(m_Z^2)$	0.1252 ± 0.0003	0.12370 ± 0.00017	0.1244 ± 0.0002
ISR $p_{T,0}^{\text{ref}}$ [GeV]	1.27 ± 0.16	Left bound	0.80 ± 0.14
`MCNNTUNES`, Per Bin model			
Parameter	p_T^Z	$\phi_\eta^\star$	p_T^Z $\phi_\eta^\star$
Primordial k_T [GeV]	1.79	1.75	1.75
ISR $\alpha_S(m_Z^2)$	0.1251	0.1238	0.1246
ISR $p_{T,0}^{\text{ref}}$ [GeV]	1.18	0.54	0.89
`MCNNTUNES`, Inverse model			
Parameter	p_T^Z	$\phi_\eta^\star$	p_T^Z $\phi_\eta^\star$
Primordial k_T [GeV]	1.87 ± 0.04	1.79 ± 0.03	1.75 ± 0.05
ISR $\alpha_S(m_Z^2)$	0.1256 ± 0.0003	0.12363 ± 0.00016	0.1244 ± 0.0003
ISR $p_{T,0}^{\text{ref}}$ [GeV]	1.36 ± 0.14	0.61 ± 0.08	0.85 ± 0.14

Table 9
Tunes using dataset 4P, all bins.

`Professor`			
Parameter	p_T^Z	$\phi_\eta^\star$	p_T^Z $\phi_\eta^\star$
Primordial k_T [GeV]	1.80 ± 0.05	1.75 ± 0.04	1.69 ± 0.04
ISR $\alpha_S(m_Z^2)$	0.1253 ± 0.0003	0.12370 ± 0.00018	0.1241 ± 0.0003
ISR $p_{T,0}^{\text{ref}}$ [GeV]	1.33 ± 0.14	Left bound	0.5 ± 0.4
MPI $p_{T,0}^{\text{ref}}$ [GeV]	2.00 ± 0.06	2.01 ± 0.04	2.05 ± 0.04
`MCNNTUNES`, Per Bin model			
Parameter	p_T^Z	$\phi_\eta^\star$	p_T^Z $\phi_\eta^\star$
Primordial k_T [GeV]	1.79	1.73	1.66
ISR $\alpha_S(m_Z^2)$	0.1252	0.1237	0.1241
ISR $p_{T,0}^{\text{ref}}$ [GeV]	1.32	Left bound	Left bound
MPI $p_{T,0}^{\text{ref}}$ [GeV]	1.90	1.99	2.01
`MCNNTUNES`, Inverse model			
Parameter	p_T^Z	$\phi_\eta^\star$	p_T^Z $\phi_\eta^\star$
Primordial k_T [GeV]	1.79 ± 0.05	1.90 ± 0.06	1.72 ± 0.04
ISR $\alpha_S(m_Z^2)$	0.1250 ± 0.0003	0.1231 ± 0.0002	0.1238 ± 0.0003
ISR $p_{T,0}^{\text{ref}}$ [GeV]	1.12 ± 0.15	0.526 ± 0.016	0.83 ± 0.13
MPI $p_{T,0}^{\text{ref}}$ [GeV]	2.0473 ± 0.0005	2.04411 ± 0.00016	2.0460 ± 0.0004

$p_{T,0}^{\text{ref}}$ is somewhere outside the left bound of its variation range. This is easy to observe for two-steps methods like `Professor` and the Per Bin model: they model the generator behaviour in the parameter space, more precisely in the hyperrectangle populated by the dataset, while the tunes are found by a minimization algorithm that explores this hyperrectangle. When the tunes seem outside of the variation ranges the algorithm finds a minimum at the boundary of the parameters hyperrectangle. The minimizers can extrapolate the results outside of the variation ranges, but there the models may be unreliable. For the Inverse model it is more complicated, because the bounds are not hard-coded into the model. Moreover, it is difficult to understand if the experimental data are near some Monte Carlo runs, so that the prediction is reliable: the envelopes are not useful because they show only whether the experimental data are inside the bounding box of the Monte Carlo runs in histograms space, but the runs do not populate this bounding box uniformly. This happens because the user has no direct control over the distribution of the runs in histogram space but only over the distribution of the parameters used to generate the runs, that are the output of the model. A different choice of the latter will affect the predictions, but the link between the distribution of the parameters of the dataset and the reliability of the predictions is less clear. When `Professor` suggests a value outside the variation range, the behaviour of the Inverse model varies: sometimes it directly predicts a value outside the variation range, sometimes a value near the left bound, sometimes a value further away. When `Professor` founds a value inside the variation range, the corresponding value for the Inverse model is compatible with it (this happens in tunes performed over all bins, shown in Tables 8 and 9).

4. Outlook

A deep learning approach to event generator tuning was presented by introducing two different procedures, called Per Bin strategy and Inverse strategy respectively. The former is a variation of the `Professor` tuning procedure, that improves over it by relaxing the assumption of a polynomial dependence of the generator response to variations of the parameters. The latter is a novel and completely different approach. The procedures were tested with closure tests and real experimental data, though in

low dimensional parameter spaces. The Per Bin model closure testing was very limited, due to computational time constraints, but showed solid results. The test with real experimental data showed a behaviour similar to the one of `Professor`. The Inverse model closure testing presented slightly better performances than the ones of `Professor`, while the test with real experimental data showed some differences from the other procedures.

In addition to the fact that the parametrization is not bound to polynomials anymore, already mentioned, another advantage of `MCNNTUNES` is that the models can learn highly non-linear functions with a limited number of parameters thanks to non-linear activation functions, at least in principle. This can be of support to the Inverse model strategy: a two-step method of parametrization and minimization is replaced by a single-step one, which is conceptually simpler, but the function to learn is more complicated.

On the other hand, the procedure brings all the difficulties that are typical of deep learning algorithms: the complexity of the training step, the dependence of the performance on the choice of the hyperparameters, the difficulty in the interpretation of the behaviour of the trained model, the overfitting problem. Moreover, the hyperparameter tuning is computationally expensive, especially for the Per Bin model, and this prevents the models to reach their full potential. Finally, the Inverse strategy introduces some practical problems, e.g. the error estimation and the reliability of the predictions when the experimental measurements have no Monte Carlo runs near them.

The behaviour of the procedures with a wider set of experimental data is still unclear, and requires more in-depth studies. In addition, whether the procedures scale well with the number of parameters is still to be determined. Future investigations may involve studying their performances in high dimensional parameter spaces. Finally, further developments may solve the practical problems highlighted above.

Nevertheless, this is a first attempt to bridge the power of machine learning algorithms into the complexity of SMC tuning. We auspicate that the greater flexibility allowed by this tool will facilitate the tuning efforts inside the experimental collaborations.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility operated under Contract No. DE-AC02-05CH11231. S.C. is supported by the European Research Council under the European Union's Horizon 2020 research and innovation Programme (grant agreement number 740006). The work of S.A. is supported by the ERC Starting Grant REINVENT-714788. He also acknowledges funding from Fondazione Cariplo, Italy and Regione Lombardia, Italy, grant 2017-2070 and by the Italian MIUR through the FARE grant R18ZRBEAFC.

References

[1] A. Buckley, H. Hoeth, H. Lacker, H. Schulz, J.E. von Seggern, Eur. Phys. J. C 65 (2010) 331–357, http://dx.doi.org/10.1140/epjc/s10052-009-1196-7.
[2] K. Hornik, Neural Netw. 4 (2) (1991) 251–257, http://dx.doi.org/10.1016/0893-6080(91)90009-T.
[3] M. Leshno, V.Y. Lin, A. Pinkus, S. Schocken, Neural Netw. 6 (6) (1993) 861–867, http://dx.doi.org/10.1016/S0893-6080(05)80131-5.
[4] Z. Lu, H. Pu, F. Wang, Z. Hu, L. Wang, in: I. Guyon, U.V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (Eds.), Advances in Neural Information Processing Systems, Vol. 30, Curran Associates, Inc., 2017, pp. 6231–6239, URL http://papers.nips.cc/paper/7203-the-expressive-power-of-neural-networks-a-view-from-the-width.pdf.
[5] A. Andreassen, B. Nachman, Phys. Rev. D 101 (9) (2020) 091901, http://dx.doi.org/10.1103/PhysRevD.101.091901, arXiv:1907.08209.
[6] N. Hansen, He CMA evolution strategy: A tutorial (2016), 2016, arXiv:1604.00772.
[7] S. Carrazza, M. Lazzarin, N3pdf/mcnntunes: mcnntunes 0.1.0, 2020, http://dx.doi.org/10.5281/zenodo.4071125.
[8] T. Sjöstrand, S. Ask, J.R. Christiansen, R. Corke, N. Desai, P. Ilten, S. Mrenna, S. Prestel, C.O. Rasmussen, P.Z. Skands, Comput. Phys. Comm. 191 (2015) 159–177, http://dx.doi.org/10.1016/j.cpc.2015.01.024.
[9] ATLAS collaboration, J. High Energy Phys. 09 (2014) 145, http://dx.doi.org/10.1007/JHEP09(2014)145.
[10] ATLAS collaboration, Phys. Lett. B 720 (1–3) (2013) 32–51, http://dx.doi.org/10.1016/j.physletb.2013.01.054.
[11] W.M. Czarnecki, I.T. Podolak, in: K. Saeed, R. Chaki, A. Cortesi, S. Wierzchoń (Eds.), Computer Information Systems and Industrial Management, CISIM 2013, in: Lecture Notes in Computer Science, vol. 8104, Springer, Berlin, Heidelberg, 2013, http://dx.doi.org/10.1007/978-3-642-40925-7_35.
[12] J. Bergstra, D. Yamins, D. Cox, Proceedings of the 30th International Conference on Machine Learning, in: PMLR, 28, 2013, pp. 115–123, (1).
[13] J. Bergstra, B. Komer, C. Eliasmith, D. Yamins, D.D. Cox, Comput. Sci. Discov. 8 (1) (2015) 014008, http://dx.doi.org/10.1088/1749-4699/8/1/014008.
[14] J.S. Bergstra, R. Bardenet, Y. Bengio, B. Kégl, in: J. Shawe-Taylor, R.S. Zemel, P.L. Bartlett, F. Pereira, K.Q. Weinberger (Eds.), Advances in Neural Information Processing Systems, Vol. 24, Curran Associates, Inc., 2011, pp. 2546–2554, URL http://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf.
[15] YODA, https://yoda.hepforge.org/.
[16] A. Buckley, J. Butterworth, D. Grellscheid, H. Hoeth, L. Lönnblad, J. Monk, H. Schulz, F. Siegert, Comput. Phys. Comm. 184 (12) (2013) 2803–2819, http://dx.doi.org/10.1016/j.cpc.2013.05.021.
[17] T.E. Oliphant, A Guide to NumPy, Trelgol Publishing USA, 2006.
[18] F. Chollet, et al., Keras, 2015, https://keras.io.
[19] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, Tensorflow: Large-scale machine learning on heterogeneous systems, 2015, software available from tensorflow.org. URL https://www.tensorflow.org/.
[20] N. Hansen, Y. Akimoto, P. Baudis, CMA-ES/Pycma on Github, Zenodo, 2019, http://dx.doi.org/10.5281/zenodo.2559634.
[21] R. Corke, T. Sjöstrand, J. High Energy Phys. (3) (2011) 32, http://dx.doi.org/10.1007/JHEP03(2011)032.
[22] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization (2014), 2014, arXiv:1412.6980.
[23] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in: Proceedings of Machine Learning Research, JMLR Workshop and Conference Proceedings, Chia Laguna Resort, vol . 9, Sardinia, Italy, 2010, pp. 249–256, URL http://proceedings.mlr.press/v9/glorot10a.html.