Neural Networks for Full Phase-space Reweighting and Parameter Tuning

Anders Andreassen[1, 2, *] and Benjamin Nachman[2, †]
[1] *Department of Physics, University of California, Berkeley, CA 94720, USA*
[2] *Physics Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA*

Precise scientific analysis in collider-based particle physics is possible because of complex simulations that connect fundamental theories to observable quantities. The significant computational cost of these programs limits the scope, precision, and accuracy of Standard Model measurements and searches for new phenomena. We therefore introduce *Deep neural networks using Classification for Tuning and Reweighting* (Dctr), a neural network-based approach to reweight and fit simulations using all kinematic and flavor information – the full phase space. Dctr can perform tasks that are currently not possible with existing methods, such as estimating non-perturbative fragmentation uncertainties. The core idea behind the new approach is to exploit powerful high-dimensional classifiers to reweight phase space as well as to identify the best parameters for describing data. Numerical examples from $e^+e^- \to$ jets demonstrate the fidelity of these methods for simulation parameters that have a big and broad impact on phase space as well as those that have a minimal and/or localized impact. The high fidelity of the full phase-space reweighting enables a new paradigm for simulations, parameter tuning, and model systematic uncertainties across particle physics and possibly beyond.

In collider-based high-energy physics, parton-, particle-, and detector-level Monte Carlo (MC) simulation programs enable scientific inference by connecting fundamental theories to observable quantities. However, these tools are often computationally slow and emulate probability distributions that are analytically intractable. This has resulted in three key simulation challenges for particle physics: (1) an insufficient number of simulated events, (2) unaccounted for biases from simulation parameters, and (3) the inability to utilize the full phase space for parameter tuning.

A variety of approaches have been proposed to address the above challenges. The two existing solutions to (1) are to use more [1–3] and/or faster computers or accelerators [4, 5] or to build fast surrogate models ('fast simulation'). Machine learning tools hold great promise for augmenting [6] or replacing [7–20] current fast detector simulation approaches, but are not yet precise enough to match the full, physics-based detector simulators that are often the limiting factor in the overall software pipeline. Deep learning methods to circumvent expensive simulations for hypothesis testing were studied in the context of effective field theory fits [21–23]; related ideas will be useful also for reweighting. The only solution for (2) aside from generating a large set of simulations or interpolating between bins of low-dimensional histograms [24] is to assign event weights for parameter variations. Currently, this is only possible for a small number of perturbative parameters in parton shower programs [25–27] and for parton distribution functions [28, 29]. Pseudo-automated procedures exist for tuning parton shower models [24, 30], but the format of the existing public data means that these algorithms are restricted to a set of mostly one-dimensional inputs that must be assumed to be independent. The variational method proposed in Ref. [31] has been demonstrated with high-dimensional data, but utilizes a minimax optimization technique and requires running the simulator many times during training.

This letter introduces *Deep neural networks using Classification for Tuning and Reweighting* (Dctr, pronounced "doctor"), a new approach to solve all three computational challenges. In particular, deep neural network-based classifiers are used to (continuously) reweight one particle-level simulation into another and additionally use the full phase space to fit parameters within a given model. When the nominal particle-level sample has a corresponding detector-level simulation, then this procedure produces a new detector-level sample as well. Non-deep machine learning tools have been used in the past for discrete re-weighting [18, 32–34] with a small number of observables. Deep-learning-based discrete weighting was considered in [18, 35] and continuous single observable reweightings were presented in [36]. The re-weighting presented here combines a full-phase space deep learning architecture [37] with parameterization [38, 39] to fully morph one simulation into another. There are no restrictions on the size of the input feature space nor on the number of interpolated parameters. In addition to re-weighting, we show how Dctr can be used with a differentiable re-weighting function (such as the one just mentioned) to optimize simulation parameters. Fitting parameters based on the parameterized classifiers was proposed in Ref. [38]; here, the fitting procedure uses a classifier to construct the loss function, which can readily incorporate all of the information from the (potentially high-dimensional) input features and be optimized using standard deep learning tools.

The first ingredient to the full phase-space reweighting procedure is a prescription to derive event weights. Consider two simulations that describe the same phase space Ω and are described by probability densities $p_0(x)$ and $p_1(x)$, for $x \in \Omega$. Assuming that p_0 and p_1 have the same support[1], the function $w(x) = p_0(x)/p_1(x)$ is the ideal per-event weight to morph the second simulation into the first one. A key observation made by multiple groups in the past is that w can be well-approximated by training a machine learning classifier to distinguish the two simulations. For example, let $f(x)$ be a neural network and trained with the binary cross-entropy loss:

$$\text{loss}(f(x)) = -\sum_{i\in\mathbf{0}} \log f(x_i) - \sum_{i\in\mathbf{1}} \log(1 - f(x_i)), \quad (1)$$

where $\mathbf{0}$ and $\mathbf{1}$ represent sets of examples from the two simulations. Then a well-known result is that[2], $f(x)/(1 - f(x)) \approx p_0(x)/p_1(x)$. The benefit of parameterizing f as a neural network is that deep learning can readily analyze all of Ω, which was not possible with shallow learning attempts with a similar statistical foundation. The closest attempt to a full phase space approach directly tried to learn $p_i(x)$ using the full kinematic (i.e. non-flavor) part of Ω [35, 40], but this is much harder than learning the ratio.

An important reweighting scenario is when the two simulations are from the same simulation program, but with different model parameters, θ. For example, when model uncertainties are evaluated, one may want to transform $p_\theta(x)$ into $p_{\theta+\delta_\theta}(x)$. When these uncertainties are profiled in a fit, it is important that the transformation procedure be able to continuously interpolate between model parameters. The neural network reweighting approximation can be extended to this continuous case by adding θ as a feature [38, 39]: $f(x, \theta)$. In the examples presented below, the training data are generated with a uniform distribution in θ, but this probability density can be optimized per application and can even be discrete.

Even though generators have many parameters that must be fit to data, gradient methods cannot be used directly with the models as the phase space they produce is not usually differentiable (or at least the derivative is intractable) with respect to their model parameters. Surrogate generative models built from neural networks can be used for gradient-based parameter fitting, but may not have sufficient quality to be reliable. Reweighting is a robust alternative to surrogate generative models. A neural network-based continuous reweighting function is essentially a differentiable (in model parameters) version of the original simulator and can be used to perform inference on the parameters themselves. This is especially powerful for particle-level parameter tuning to data where one sample with a computational expensive full detector simulation can be continuously reweighted to other parameter points with the same detector model at no extra simulation cost.

An ideal loss function used to fit model parameters makes use of the full observable phase space. Typical metrics such as the χ^2 between histogram approximations to probability densities become impractical when Ω is high dimensional. As described above, classifiers are powerful tools for accessing all of the available information. Therefore, one can use a classifier *for the loss*. When a classifier trained to distinguish some $\boldsymbol{\theta_0}$ from a $\boldsymbol{\theta_1}$ performs poorly, then the two samples are close. While using classification to quantify differences between event samples has been used for anomaly detection [41–43], we are unaware of an example where it is used for parameter fitting. The idea of using the classifier loss as a metric is similar to the minimax strategy in Generative Adversarial Networks [44], only in this context the generative part is a reweighter and is trained independently. A more elegant way of implementing this approach is to fit unknown parameters to the values that minimize the nominal classifier loss. In particular, suppose that a reweighter neural network f is trained as described above. Such a function will satisfy

$$f(x, \theta) = \underset{f'}{\text{argmax}} \sum_{i\in\boldsymbol{\theta_0}} \log f'(x_i, \theta) + \sum_{i\in\boldsymbol{\theta}} \log(1 - f'(x_i, \theta)) \quad (2)$$

for all θ. Note that the f' in the first sum takes the parameter θ and not θ_0, otherwise the discrimination task would be trivial. Now, suppose there is a new sample $\boldsymbol{\theta_1}$ where θ_1 is unknown (for instance, $\boldsymbol{\theta_1}$ are collider data). The claim is that if θ^* is chosen as

$$\theta^* = \underset{\theta'}{\text{argmax}} \sum_{i\in\boldsymbol{\theta_0}} \log f(x_i, \theta') + \sum_{i\in\boldsymbol{\theta_1}} \log(1 - f(x_i, \theta')) \quad (3)$$

then $\theta^* = \theta_1$. As f minimizes the cross-entropy loss for any θ (Eq. 2),

$$\begin{aligned}&\sum_{i\in\boldsymbol{\theta_0}} \log f(x_i, \theta_1) + \sum_{i\in\boldsymbol{\theta_1}} \log(1 - f(x_i, \theta_1)) \\ &\quad \geq \sum_{i\in\boldsymbol{\theta_0}} \log f(x_i, \theta^*) + \sum_{i\in\boldsymbol{\theta_1}} \log(1 - f(x_i, \theta^*)) \quad (4)\end{aligned}$$

must hold. However, the converse must also be true since θ^* minimizes the cross-entropy loss as well and therefore, $\theta^* = \theta_1$. Since f is differentiable, Eq. 3 can be solved using standard gradient-based methods. While Eq. 3 performs the fit on the same particle-level phase space as the reweighting, it can be readily extended to do the fitting (via the classification loss) at detector-level while the

[1] In most physical applications, this is always the case. If there are regions where $p_0(x)/p_1(x)$ is far from unity, one can add a regularization parameter to the training to mitigate large weights, which may significantly reduce the statistical power of the reweighted dataset. We found that this works well, but was unnecessary for the examples presented in this paper.

[2] See Appendix A for the derivation.

reweighting can be performed at particle-level using one fully detector-simulated event sample (see Appendix B).

The last ingredient to DCTR is a suitable neural network architecture that can effectively capture all the salient features of Ω. A natural tool for this task is the Particle Flow Network (PFN) [37], built on the Deep Sets framework [45]. While many deep learning architectures incorporate the symmetries and structure of high energy physics events [8, 35, 46–56], PFNs are particularly effective because they can operate on variable-length sets of particles and respect the quantum-mechanically induced permutation invariance of particle labels. These networks can also readily incorporate non-kinematic information such as particle flavor. A particle flow network is a composition of two neural networks F and Φ: $f(\{p_i\}) = F(\sum_{i=1}^{n} \Phi(p_i))$, where p_i is the set of features belong to particle i (momentum and flavor) as well as θ. The function Φ embeds the input particles into an ℓ-dimensional *latent space* and F is a simple $\mathbb{R}^\ell \mapsto \mathbb{R}$ neural network. References [37, 45] proved that this structure is sufficiently flexible to approximate any function and in practice, $\ell \sim \mathcal{O}(10)$.

To illustrate the potential of DCTR, full phase-space reweighting and parameter tuning is performed on a sample of generated events from the PYTHIA 8.230 [57, 58] event generator. Particle-level $e^+e^- \to Z \to$ dijet events with about 100 particles in each event are clustered into jets using the anti-k_t clustering algorithm [59] ($R = 0.8$) with Fastjet 3.0.3 [60, 61]. The jets are presented to the neural network for training, with each jet constituent represented by $(p_T, \eta, \phi, \text{particle type}, \theta)$, where θ is the parameter in Eq. (2). One million events were generated for each set of PYTHIA parameters. In addition to a default parameter set using the Monash tune [62], three separate samples were generated with uniformly sampled `TimeShower:alphaSvalue`, `StringZ:aLund` and `StringFlav:probStoUD` in the ranges $[0.10, 0.18]$, $[0.50, 0.90]$ and $[0.10, 0.30]$, respectively. We also generated one sample where all three parameters were simultaneously uniformly sampled. These parameters were chosen because they represent both perturbative and non-perturbative physical effects and the ranges are similar to those studied in Ref. [30]. The Monash values of the three parameters are 0.1365, 0.68, and 0.217, respectively.

The reweighting and fitting was found to work well without any hyperparameter modifications from Ref. [37]. In particular, Φ has two hidden layers with $\ell = 128$ and F is composed of three hidden layers and two output nodes for binary classification, and all the hidden layers have 100 nodes. The activation function used for all layers is ReLu with the exception of the classification output which uses softmax. All models were implemented in Keras [63] with the Tensorflow backend [64] and trained using the crossentropy loss with the Adam [65] optimizer for 50 epochs, using early stopping with patience 10, with batch size 1000. Each training set contained $8 \cdot 10^5$ training and 10^5 validation jets of each class. Training time was 10-15 min for each model (20 seconds per epoch) on an NVIDIA GeForce GTX 1080.

As a first test of DCTR, a single parameter (`TimeShower:alphaSvalue`) is reweighted using the full phase space of the generated jets. Results for discrete and continuous reweighting from a varied parameter to the nominal sample are presented in Fig. 1. The entire phase-space is reweighted, but is too high dimensional to visualize. Instead, three histograms of physically relevant one-dimensional observables are presented: the number of particles inside the jet (multiplicity), an n-subjettiness ratio τ_{32} [66, 67], and a four-point Energy Correlation Function [68] ECF($\text{N} = 4, \beta = 4$). By definition, $\tau_{32} = \tau_3/\tau_2$ where $\tau_n = \sum_{i\in\text{jet}} p_{T,i} \min_{j=1\ldots n}\{\Delta R(i,j)\}$ for axis j; likewise, ECF(N, β) is the sum over all quadruples inside the jet weighted by the product of the momenta and the product of all opening angles raised to the power β. The large values of n, N, and β are used to expose complex features with a nontrivial dependence on all particles inside the jet. Many more observables were studied, but these ones are representative.

The reweighted distributions are in excellent agreement with the target nominal distribution. Samples used to make the histograms shown for `TimeShower:alphaSvalue` values 0.1365 and 0.1600 were not used during training or validation. The fidelity of a continuous reweighting is quantified in the lower right plot of Fig. 1, which presents the χ^2/ndf as a function of the initial α_s parameter value.

Variations in `TimeShower:alphaSvalue` modify many aspects of jet fragmentation and therefore it may be an easy parameter for the reweighting network to learn. In contrast, the hadronization parameters `StringZ:aLund` and `StringFlav:probStoUD` may be more difficult because the size of their effects on the phase space is small and/or localized. Figure 2 shows that despite these potential challenges, the reweighting procedure is able to effectively capture subtle and isolated modifications to the phase space. Variations in the `StringZ:aLund` parameter result in mostly percent-level differences in the presented distributions, which are corrected in the reweighted model. Modifying the `StringFlav:probStoUD` parameter only changes strange particles such as kaons, which highlights the importance of including flavor in the full phase space network. Importantly, this model learns to only change the distributions related to strange particles, leaving other observables untouched. Simultaneously reweighting all three parameters also works well, but is more difficult to visualize. We also verified that DCTR works for *pp* MC simulations by reweighting from PYTHIA to HERWIG for both the quark and gluon samples taken from [37, 69, 70].

The well-trained DCTR model can now be used to demonstrate the potential for parameter tuning follow-

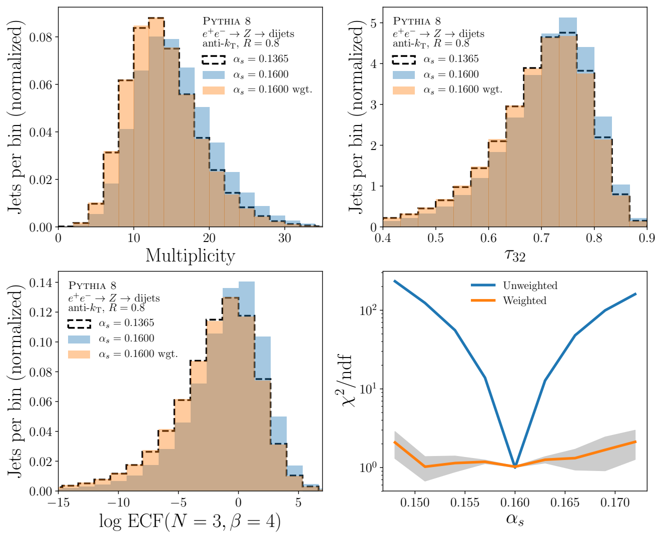

FIG. 1. The three histograms shows the result before and after reweighting between two values of `TimeShower:alphaSvalue` $= \alpha_s$ on different 1D observables. To quantify the quality of the reweighing, and to illustrate one trained model can continuously reweight for any parameter, we show the χ^2/ndf for multiplicity as a function of α_s in the lower right plot for reweighting to $\alpha_s = 0.1600$. For each value, we compare the χ^2 relative to $\alpha_s = 0.1600$ before and after reweighting. Each χ^2 value is averaged over 10 runs and the grey band marks the standard deviation, which is consistent with $\chi^2/\text{ndf} \approx 1$.

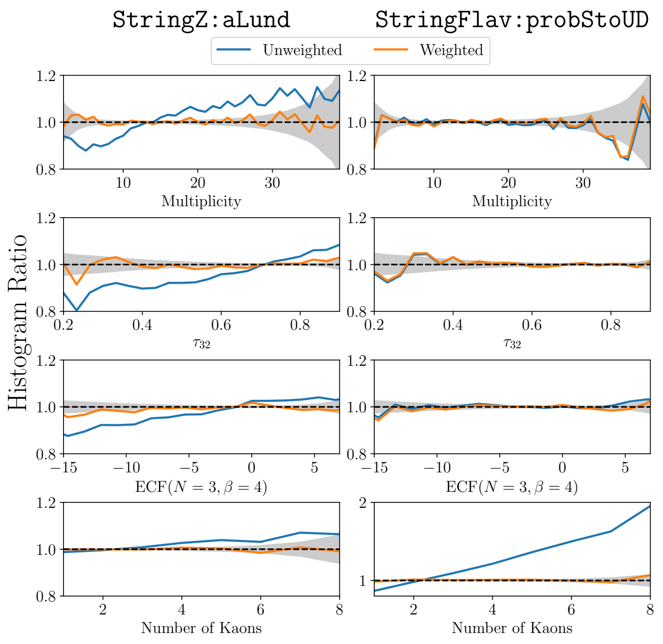

FIG. 2. Ratio of histograms from nominal distribution to sample generated with `StringZ:aLund` = 0.8 on the left and `StringFlav:probStoUD` = 0.275 on the right. Both unweighted and weighted histograms ratios are shown. The gray band indicates the statistical uncertainty from both the nominal and variation sample. After reweighting, the ratio only differs from 1 within the statistical uncertainty.

ing Eq. 3. As a first step, Table I presents the result of a fit where the 'data' are the same as the nominal, but with each parameter changed one at a time (each row is a separate fit). To illustrate the sensitivity to the randomness in the model initialization, each fit is performed ten times. This variation could be reduced with a more sophisticated neural network and/or more training data. For each of these one-dimensional fits, the fitted value is consistent with the target value within these statistical fluctuations from initialization, which are $1-3\%$. As `TimeShower:alphaSvalue` has a bigger impact on the phase space, it is less sensitive to the initialization statistical fluctuations. For a fit with data, the statistical and systematic uncertainty could be determined with toys and even profiled, as is standard for parameter fitting.

TABLE I. Independent fit for simulation where one parameter was changed at a time. The reported numbers are the mean and standard deviation over 10 runs with different model initializations.

Parameter	**Target value**	**Fit value**
`TimeShower:alphaSvalue`	0.1600	0.1601 ± 0.0018
`StringZ:aLund`	0.8000	0.7980 ± 0.0257
`StringFlav:probStoUD`	0.2750	0.2754 ± 0.0065

As a next step, the top part of Table II shows the result of a simultaneous fit to the three parameters. As with the one-dimensional fit, the fitted values are all statistically consistent with the target values. Interestingly, the sensitivity to the initialization statistical fluctuations is about the same for the three-dimensional fit as for the one-dimensional fits, providing confidence in the scaling to more parameters. In practice, the fitting procedure would be validated on a variety of simulations with known parameters, as just described. An illustration of the fit itself is shown in Fig. 3, where a two-dimensional slice through the likelihood landscape is presented and the fit execution demonstrated with markers and dashed lines. The broadness of the loss in the `StringZ:aLund` direction relative to the `TimeShower:alphaSvalue` one is a reflection of the significantly smaller impact of fragmentation function variations on the observable phase space compared with modifications to the final state shower strong coupling. After the validation, the model can be deployed on data, where the parameters are unknown. The lower part of Table II replicates this scenario, where the Pythia parameters were blinded during the fit. This closure test indicates that the method is robust to user-bias.

The empirical results demonstrate that DCTR is ready to be deployed. The discrete reweighting could be used

TABLE II. Simultaneous fit for three parameters. The top row shows the results for the validation fit where we knew the target parameters, and the bottom row is the blinded fit. The reported numbers are the mean and standard deviation over 20 runs with different model initializations.

	Parameter	Target value	Fit value
Val.	`TimeShower:alphaSvalue`	0.1200	0.1195 ± 0.0022
	`StringZ:aLund`	0.6000	0.6276 ± 0.0373
	`StringFlav:probStoUD`	0.1200	0.1203 ± 0.0071
Blinded	`TimeShower:alphaSvalue`	0.1700	0.1707 ± 0.0022
	`StringZ:aLund`	0.7500	0.7425 ± 0.0453
	`StringFlav:probStoUD`	0.1400	0.1422 ± 0.0065

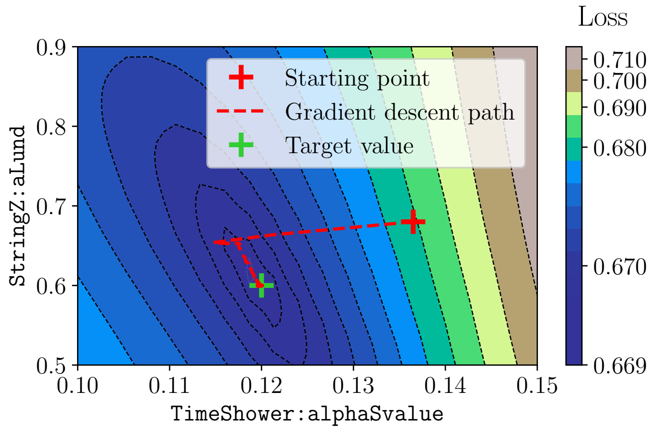

FIG. 3. Two-dimensional slice through the loss surface for the fit described in Table II. Markers indicate the starting point at nominal values, the gradient descent path and the target values. From the starting point, gradient descent using Adam overshoots the minimum in its first two epochs before it converges to the target value.

to generate new full-detector simulated samples with a different particle-level simulation when at least one fully simulated sample exists. This could be particularly useful for systematic uncertainties computed using pairs of simulations (e.g. comparing Pythia and Herwig) and for legacy data analysis in which the original detector simulation is no longer available [71]. Continuous reweighting will enable systematic parameter variations for uncertainty estimation that were not possible before (most parameters). Such variations can even be profiled during any statistical test that fits phase space regions sensitive to the varied nuisance parameters. Finally, the full power of DCTR can be used for parameter tuning. Unlike traditional tuning which use unfolded data that are usually one-dimensional and without observable-observable correlations, a new paradigm is now possible were high-dimensional detector-level data can be used directly. The full power of the data can be utilized and all of the correlations are correctly accounted for in the fit. For the first time, this may allow for proper covariance matrices (and thus correlated uncertainties) to be determined for simulation parameter values. All of these opportunities illustrate the broad applicability of full phase-space reweighting and parameter tuning and the power DCTR to extend the scope, precision, and accuracy of collider-based particle physics analyses.

BN would like to thank Luke de Oliveira, Michela Paganini, Chase Shimmin, and Paul Tipton for collaboration on an early stage prototype of this project. We thank Steve Mrenna and Peter Skands for helpful discussions about automated variations in Pythia. We also thank Kyle Cranmer, Aviv Cukierman, Patrick Komiske, and Eric Metodiev, and Jesse Thaler for lively discussions about deep learning-based reweighting. Additionally, we are grateful to Kyle Cranmer, Phil Ilten, and Jesse Thaler for feedback on the manuscript. This work was supported by the U.S. Department of Energy, Office of Science under contract DE-AC02-05CH11231. Finally, we are grateful for the opportunity to use the Cori supercomputing resources at NERSC.

* andersja@berkeley.edu
† bpnachman@lbl.gov

[1] S. Ahn, J. Apostolakis, M. Asai, D. Brandt, G. Cooperman, G. Cosmo, A. Dotti, X. Dong, S. Yung Jun, and A. Nowak (2014) p. 04213.
[2] S. Farrell, A. Dotti, M. Asai, P. Calafiura, and R. Monnard, in *Proceedings, 2015 IEEE Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC 2015): San Diego, California, United States* (2016) p. 7581868, arXiv:1605.08371 [physics.comp-ph].
[3] J. T. Childers, T. D. Uram, T. J. LeCompte, M. E. Papka, and D. P. Benjamin, Comput. Phys. Commun. **210**, 54 (2017), arXiv:1511.07312 [hep-ph].
[4] O. Seiskari, J. Kommeri, and T. Niemi, (2012), arXiv:1209.5235 [physics.comp-ph].
[5] P. Canal, D. Elvira, S. Y. Jun, J. Kowalkowski, M. Paterno, and J. Apostolakis, *Proceedings, 20th International Conference on Computing in High Energy and Nuclear Physics (CHEP 2013): Amsterdam, The Netherlands, October 14-18, 2013*, J. Phys. Conf. Ser. **513**, 052013 (2014).
[6] ATLAS Collaboration, ATL-SOFT-PUB-2018-002 (2018).
[7] M. Paganini, L. de Oliveira, and B. Nachman, Phys. Rev. **D97**, 014021 (2018), arXiv:1712.10321 [hep-ex].
[8] L. de Oliveira, M. Paganini, and B. Nachman, Comput. Softw. Big Sci. **1**, 4 (2017), arXiv:1701.05927 [stat.ML].
[9] M. Paganini, L. de Oliveira, and B. Nachman, Phys. Rev. Lett. **120**, 042003 (2018), arXiv:1705.02355 [hep-ex].
[10] L. de Oliveira, M. Paganini, and B. Nachman, *Proceedings, 18th International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT 2017): Seattle, WA, USA, August 21-25, 2017*, J. Phys. Conf. Ser. **1085**, 042017 (2018), arXiv:1711.08813 [hep-

ex].

[11] M. Erdmann, L. Geiger, J. Glombitza, and D. Schmidt, Comput. Softw. Big Sci. **2**, 4 (2018), arXiv:1802.03325 [astro-ph.IM].

[12] P. Musella and F. Pandolfi, Comput. Softw. Big Sci. **2**, 8 (2018), arXiv:1805.00850 [hep-ex].

[13] M. Erdmann, J. Glombitza, and T. Quast, Comput. Softw. Big Sci. **3**, 4 (2019), arXiv:1807.01954 [physics.ins-det].

[14] F. Carminati, A. Gheata, G. Khattak, P. Mendez Lorenzo, S. Sharan, and S. Vallecorsa, *Proceedings, 18th International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT 2017): Seattle, WA, USA, August 21-25, 2017*, J. Phys. Conf. Ser. **1085**, 032016 (2018).

[15] V. Chekalina, E. Orlova, F. Ratnikov, D. Ulyanov, A. Ustyuzhanin, and E. Zakharov, in *23rd International Conference on Computing in High Energy and Nuclear Physics (CHEP 2018) Sofia, Bulgaria, July 9-13, 2018* (2018) arXiv:1812.01319 [physics.data-an].

[16] B. Hashemi, N. Amin, K. Datta, D. Olivito, and M. Pierini, (2019), arXiv:1901.05282 [hep-ex].

[17] R. Di Sipio, M. Faucci Giannelli, S. Ketabchi Haghighat, and S. Palazzo, (2019), arXiv:1903.02433 [hep-ex].

[18] R. Aaij *et al.* (LHCb Collaboration), Phys. Rev. Lett. **119**, 062001 (2017), arXiv:1704.07900 [hep-ex].

[19] J. W. Monk, JHEP **12**, 021 (2018), arXiv:1807.03685 [hep-ph].

[20] ATLAS Collaboration, ATL-SOFT-PUB-2018-001 (2018).

[21] J. Brehmer, K. Cranmer, G. Louppe, and J. Pavez, Phys. Rev. Lett. **121**, 111801 (2018), arXiv:1805.00013 [hep-ph].

[22] J. Brehmer, K. Cranmer, G. Louppe, and J. Pavez, Phys. Rev. **D98**, 052004 (2018), arXiv:1805.00020 [hep-ph].

[23] J. Brehmer, G. Louppe, J. Pavez, and K. Cranmer, (2018), arXiv:1805.12244 [stat.ML].

[24] A. Buckley, H. Hoeth, H. Lacker, H. Schulz, and J. E. von Seggern, The European Physical Journal C **65**, 331 (2009).

[25] S. Mrenna and P. Skands, Phys. Rev. **D94**, 074005 (2016), arXiv:1605.08352 [hep-ph].

[26] J. Bellm, S. Plätzer, P. Richardson, A. Sidmok, and S. Webster, Phys. Rev. **D94**, 034028 (2016), arXiv:1605.08256 [hep-ph].

[27] E. Bothmann, M. Schnherr, and S. Schumann, Eur. Phys. J. **C76**, 590 (2016), arXiv:1606.08753 [hep-ph].

[28] J. Butterworth *et al.*, J. Phys. **G43**, 023001 (2016), arXiv:1510.03865 [hep-ph].

[29] A. Buckley, J. Ferrando, S. Lloyd, K. Nordstrm, B. Page, M. Rfenacht, M. Schnherr, and G. Watt, Eur. Phys. J. **C75**, 132 (2015), arXiv:1412.7420 [hep-ph].

[30] P. Ilten, M. Williams, and Y. Yang, Journal of Instrumentation **12**, P04028 (2017).

[31] G. Louppe, J. Hermans, and K. Cranmer, in *Proceedings of Machine Learning Research*, Proceedings of Machine Learning Research, Vol. 89, edited by K. Chaudhuri and M. Sugiyama (PMLR, 2019) pp. 1438–1447.

[32] D. Martschei, M. Feindt, S. Honc, and J. Wagner-Kuhr, *Proceedings, 14th International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT 2011): Uxbridge, UK, September 5-9, 2011*, J. Phys. Conf. Ser. **368**, 012028 (2012).

[33] A. Rogozhnikov, *Proceedings, 17th International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT 2016): Valparaiso, Chile, January 18-22, 2016*, J. Phys. Conf. Ser. **762**, 012036 (2016), arXiv:1608.05806 [physics.data-an].

[34] M. Aaboud *et al.* (ATLAS Collaboration), Phys. Rev. **D98**, 092002 (2018), arXiv:1806.04030 [hep-ex].

[35] A. Andreassen, I. Feige, C. Frye, and M. D. Schwartz, (2018), arXiv:1804.09720 [hep-ph].

[36] E. Bothmann and L. Debbio, JHEP **01**, 033 (2019), arXiv:1808.07802 [hep-ph].

[37] P. T. Komiske, E. M. Metodiev, and J. Thaler, JHEP **01**, 121 (2019), arXiv:1810.05165 [hep-ph].

[38] K. Cranmer, J. Pavez, and G. Louppe, (2015), arXiv:1506.02169 [stat.AP].

[39] P. Baldi, K. Cranmer, T. Faucett, P. Sadowski, and D. Whiteson, Eur. Phys. J. **C76**, 235 (2016), arXiv:1601.07913 [hep-ex].

[40] A. Andreassen, I. Feige, C. Frye, and M. D. Schwartz, (2019), arXiv:1906.10137 [hep-ph].

[41] R. T. D'Agnolo and A. Wulzer, Phys. Rev. **D99**, 015014 (2019), arXiv:1806.02350 [hep-ph].

[42] J. H. Collins, K. Howe, and B. Nachman, Phys. Rev. **D99**, 014038 (2019), arXiv:1902.02634 [hep-ph].

[43] J. H. Collins, K. Howe, and B. Nachman, Phys. Rev. Lett. **121**, 241803 (2018), arXiv:1805.02664 [hep-ph].

[44] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14 (MIT Press, Cambridge, MA, USA, 2014) pp. 2672–2680.

[45] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Póczos, R. Salakhutdinov, and A. J. Smola, CoRR **abs/1703.06114** (2017), arXiv:1703.06114.

[46] L. de Oliveira, M. Kagan, L. Mackey, B. Nachman, and A. Schwartzman, JHEP **07**, 069 (2016), arXiv:1511.05190 [hep-ph].

[47] P. Baldi, K. Bauer, C. Eng, P. Sadowski, and D. Whiteson, Phys. Rev. **D93**, 094034 (2016), arXiv:1603.09349 [hep-ex].

[48] J. Barnard, E. N. Dawe, M. J. Dolan, and N. Rajcic, Phys. Rev. **D95**, 014018 (2017), arXiv:1609.00607 [hep-ph].

[49] P. T. Komiske, E. M. Metodiev, and M. D. Schwartz, JHEP **01**, 110 (2017), arXiv:1612.01551 [hep-ph].

[50] G. Louppe, K. Cho, C. Becot, and K. Cranmer, JHEP **01**, 057 (2019), arXiv:1702.00748 [hep-ph].

[51] H. Qu and L. Gouskos, (2019), arXiv:1902.08570 [hep-ph].

[52] A. Butter, G. Kasieczka, T. Plehn, and M. Russell, SciPost Phys. **5**, 028 (2018), arXiv:1707.08966 [hep-ph].

[53] D. Guest, J. Collado, P. Baldi, S.-C. Hsu, G. Urban, and D. Whiteson, Phys. Rev. **D94**, 112002 (2016), arXiv:1607.08633 [hep-ex].

[54] ATLAS Collaboration, ATL-PHYS-PUB-2017-003 (2017).

[55] CMS Collaboration, CMS-DP-2017-049 (2017).

[56] A. M. Sirunyan *et al.* (CMS), JINST **13**, P05011 (2018), arXiv:1712.07158 [physics.ins-det].

[57] T. Sjöstrand, S. Ask, J. R. Christiansen, R. Corke, N. Desai, P. Ilten, S. Mrenna, S. Prestel, C. O. Rasmussen, and P. Z. Skands, Comput. Phys. Commun. **191**, 159 (2015), arXiv:1410.3012 [hep-ph].

[58] T. Sjöstrand, S. Mrenna, and P. Z. Skands, JHEP **05**,

026 (2006), arXiv:hep-ph/0603175 [hep-ph].
[59] M. Cacciari, G. P. Salam, and G. Soyez, JHEP **04**, 063 (2008), arXiv:0802.1189 [hep-ph].
[60] M. Cacciari, G. P. Salam, and G. Soyez, Eur. Phys. J. **C72**, 1896 (2012), arXiv:1111.6097 [hep-ph].
[61] M. Cacciari and G. P. Salam, Phys. Lett. **B641**, 57 (2006), arXiv:hep-ph/0512210 [hep-ph].
[62] P. Skands, S. Carrazza, and J. Rojo, Eur. Phys. J. **C74**, 3024 (2014), arXiv:1404.5630 [hep-ph].
[63] F. Chollet, "Keras," `https://github.com/fchollet/keras` (2017).
[64] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, in *OSDI*, Vol. 16 (2016) pp. 265–283.
[65] D. Kingma and J. Ba, (2014), arXiv:1412.6980 [cs].
[66] J. Thaler and K. Van Tilburg, JHEP **02**, 093 (2012), arXiv:1108.2701 [hep-ph].
[67] J. Thaler and K. Van Tilburg, JHEP **03**, 015 (2011), arXiv:1011.2268 [hep-ph].
[68] A. J. Larkoski, G. P. Salam, and J. Thaler, JHEP **06**, 108 (2013), arXiv:1305.0007 [hep-ph].
[69] A. Pathak, P. Komiske, E. Metodiev, and M. Schwartz, "Herwig7.1 quark and gluon jets," (2019).
[70] P. Komiske, E. Metodiev, and J. Thaler, "Pythia8 quark and gluon jets for energy flow," (2019).
[71] A. Badea, A. Baty, P. Chang, G. M. Innocenti, M. Maggi, C. Mcginn, M. Peters, T.-A. Sheng, J. Thaler, and Y.-J. Lee, (2019), arXiv:1906.00489 [hep-ex].
[72] A. Andreassen and B. Nachman, `https://github.com/bnachman/DCTR`.

Appendix A: Optimal Functions

The results presented here can be found (as exercises) in textbooks, but are repeated here for easy access. Let X be some discriminating features and $Y \in \{0,1\}$ is another random variable representing class membership. Consider the general problem of minimizing some average loss for the function $f(x)$:

$$f = \text{argmin}_{f'}\mathbb{E}[\text{loss}(f'(X), Y)], \tag{A1}$$

where $\mathbb{E}$ means 'expected value', i.e. average value or mean (sometimes represented as $\langle\cdot\rangle$). The expectation values are performed over the joint probability density of (X, Y). One can rewrite Eq. A1 as

$$f = \text{argmin}_{f'}\mathbb{E}[\mathbb{E}[\text{loss}(f'(X), Y)|X]]. \tag{A2}$$

The advantage[3] of writing the loss as in Eq. A2 is that one can see that it is sufficient to minimize the function (and not functional) $\mathbb{E}[\text{loss}(f'(x), Y)|X = x]$ for all x. To see this, let $g(x) = \text{argmin}_{f'}\mathbb{E}[\text{loss}(f'(x), Y)|X = x]$ and suppose that $h(x)$ is a function with a strictly smaller loss in Eq. A1 than g. Since the average loss for h is below that of g, by the intermediate value theorem, there must be an x for which the average loss for h is below that of g, contradicting the construction of g.

[3] The derivation below for the mean-squared error was partially inspired by Appendix A in Ref. [38].

Now, consider the case where the loss is cross-entropy:

$$\max_z \mathbb{E}[Y\log(z) + (1-Y)\log(1-z)|X] \tag{A3}$$
$$= \max_z \left(\mathbb{E}[Y|X]\log(z) + (1-\mathbb{E}[Y|X])\log(1-z)\right), \tag{A4}$$

where $z = f'(x)$ is fixed. Equation A3 is maximized for $g(x) = \mathbb{E}[Y|X = x]$. Coincidentally, the exact same result holds if using mean squared error loss. When using either loss function with two outputs and the softmax activation for the last neural network layer, the first output will asymptotically approach $g(x)$ and the other by construction will be $1 - g(x)$. The ratio of these two outputs is then:

$$\frac{g(x)}{1-g(x)} = \frac{\mathbb{E}[Y|X=x]}{\mathbb{E}[1-Y|X=x]} \tag{A5}$$
$$= \frac{\Pr(Y=1|X=x)}{\Pr(Y=0|X=x)} \tag{A6}$$
$$= \frac{p(X|Y=1)\Pr(Y=1)}{p(X|Y=0)\Pr(Y=0)} \tag{A7}$$
$$= \text{Likelihood ratio} \times \frac{\Pr(Y=1)}{\Pr(Y=0)}. \tag{A8}$$

Therefore, the output is proportional to the likelihood ratio. The proportionality constant is the ratio of fractions of the two classes used during the training. In the paper, the two classes always have the same number of examples and thus this factor is unity.

Appendix B: Alternative Fitting Method

In the main body, it was shown how a continuously parameterized NN used for reweighting:

$$f(x,\theta) = \underset{f'}{\text{argmax}} \sum_{i\in\theta_0}\log(f'(x_i,\theta)) + \sum_{i\in\theta}\log(1-f'(x_i,\theta)) \tag{B1}$$

can also be used for fitting:

$$\theta^* = \underset{\theta'}{\text{argmax}} \sum_{i\in\theta_0}\log(f'(x_i,\theta')) + \sum_{i\in\theta}\log(1-f'(x_i,\theta')). \tag{B2}$$

This works well when the reweighting and fitting happen on the same 'level'. However, if the reweighting happens at truth level (before detector simulation) while the fit happens in data (after the effects of the detector), this procedure will not work. It works only if the reweighting and fitting both happen at detector-level or both happen at truth-level. The following is an alternative method:

$$\theta^* = \underset{\theta'}{\operatorname{argmax}} \min_g \sum_{i\in\theta_{\mathbf{0}}} \log(g(x_i)) + \sum_{i\in\theta} w(x_i,\theta)\log(1-g(x_i)), \tag{B3}$$

where $w(x_i,\theta) = f(x_i,\theta)/(1-f(x_i,\theta))$ is a trained DCTR using binary cross entropy as in the main body. The intuition of the above equation is that the classifier g is trying to distinguish the two samples and we try to find a θ that makes g's task maximally hard. If g cannot tell apart the two samples, then the reweighting has worked. This is similar to the minimax graining of a GAN, only now the analog of the generator network is the reweighting network which is fixed and thus the only trainable parameters are the θ'. The advantage of this second approach is that it readily generalizes to the case where the reweighting happens on a different level:

$$\theta^* = \underset{\theta'}{\operatorname{argmax}} \min_g \sum_{i\in\theta_{\mathbf{0}}} \log g(x_{D,i}) + \sum_{i\in\theta} w(x_{T,i},\theta)\log(1-g(x_{D,i})), \tag{B4}$$

where x_T is the truth value and x_D is the detector-level value. In simulation (the second sum), these come in pairs and so one can apply the reweighting on one level and the classification on the other.

Asymptotically, both this method and the one in the body of the DCTR paper learn the same result: $\theta^* = \theta_0$. To see this for the second method, consider the same logic as in Appendix A. Conditioning on x and θ, the optimal g is given by

$$g = \frac{\mathbb{E}[Y|X=x]}{(1-\mathbb{E}[Y|X=x])w(x,\theta)+\mathbb{E}[Y|X=x]}, \tag{B5}$$

which reduces to the result of the previous appendix when $w=0$. For fixed g, the loss is maximized when g is independent of x, which happens if $(1-\mathbb{E}[Y|X=x])w(x,\theta) \propto \mathbb{E}[Y|X=x])$, which means that $w(x,\theta)$ is proportional to the likelihood ratio between the two samples. An example implementation of this method in Keras can be found at Ref. [72].