EPJ Web of Conferences **251**, 03055 (2021) https://doi.org/10.1051/epjconf/202125103055
CHEP 2021

Conditional Wasserstein Generative Adversarial Networks for Fast Detector Simulation

John Blue[1,*], *Braden* Kronheim[1], *Michelle* Kuchera[1], and *Raghuram* Ramanujan[1]

[1]Davidson College, Davidson, North Carolina 28035, United States

Abstract. Detector simulation in high energy physics experiments is a key yet computationally expensive step in the event simulation process. There has been much recent interest in using deep generative models as a faster alternative to the full Monte Carlo simulation process in situations in which the utmost accuracy is not necessary. In this work we investigate the use of conditional Wasserstein Generative Adversarial Networks to simulate both hadronization and the detector response to jets. Our model takes the 4-momenta of jets formed from partons post-showering and pre-hadronization as inputs and predicts the 4-momenta of the corresponding reconstructed jet. Our model is trained on fully simulated $t\bar{t}$ events using the publicly available GEANT-based simulation of the CMS Collaboration. We demonstrate that the model produces accurate conditional reconstructed jet transverse momentum (p_T) distributions over a wide range of p_T for the input parton jet. Our model takes only a fraction of the time necessary for conventional detector simulation methods, running on a CPU in less than a millisecond per event.

1 Introduction

Comparing theoretical predictions with experimental results in high energy particle collisions such as those produced at the LHC is a challenging problem. In addition to the complicated calculations that go into predicting the final states of the collisions, it is necessary to either simulate the response of the detector to the final state particles and apply event reconstruction algorithms to the simulated data, or go in the inverse direction and unfold the detector effects from experimental results. When the first approach is taken and high accuracy is needed, studies use Monte Carlo (MC)-based detector simulators such as `GEANT4` [1]. Unfortunately, this accuracy comes at a significant computational cost, and processing a single LHC event can take on the order of minutes. Several publicly available fast simulators such as `Delphes` [2] have been introduced as a quicker, lower fidelity, alternatives to the full MC simulations. Such fast simulators parameterize the detector response function R_D, and then randomly sample from R_D for each particle in the event.

In this work, we take advantage of recent advances in deep generative modeling to learn R_D using conditional Wasserstein Generative Adversarial Networks (cWGAN). We focus on the simulation of jets, collimated clusters of particles arising from the hadronization of quarks and gluons. There now exists a significant body of work on many aspects of jet simulation. In

*e-mail: joblue@davidson.edu

particular, many studies have simulated the calorimeter response using Generative Adversarial Networks (GANs) [3–5], auto-regressive models [6], autoencoders [7], and graph neural networks [8]. Other works have used GANs to produce jet images [9] and predict kinematic observables of dijet events [10]. Our work differs from previous studies in that we condition our model on jets formed at the parton level, post-parton showering and pre-hadronization, and the model directly predicts the 4-momenta of reconstructed jets. This choice of input to the model seeks to increase the speed at which the full event can be simulated as the chosen event generator would stop before hadronization. We refer to this approach and project as `Falcon`.

The paper is organized as follows. We first describe details of the simulated data we have used and follow with an overview of cWGANs. Details of our model are presented next, followed by our results and conclusions.

2 Simulated Data

In order to train our model, we generated roughly $168,000$ $t\bar{t}$ events at 13 TeV without pileup using a standard CMS Open Data [11, 12] workflow in the publicly available software library CMSSW [13] of the CMS Collaboration. Event generation was performed with `Pythia 8` [14] and the simulation of the CMS detector was done with `GEANT4`. Events were reconstructed using the Particle Flow (PF) reconstruction algorithm [15] and reconstructed jets were clustered from PF candidates using the anti-k_t algorithm [16] with distance parameter $R = 0.4$.

Since our goal is to map directly from the parton jets to reconstructed jets, an algorithm is needed to identify the appropriate set of partons prior to hadronization. These partons were identified as follows. We start with an empty set of partons. Then, for each stable generated particle, we recursively traverse through each mother particle, checking if the particle encountered was a parton. Each unique parton was added to the set. The set of partons so obtained were then clustered into jets using the anti-k_t algorithm implemented in the `FastJet` software package [17] again with distance parameter $R = 0.4$. In each event, a parton jet and a reconstructed jet were matched if $\Delta R = \sqrt{\Delta\eta^2 + \Delta\phi^2} < 0.35$, where $\Delta\eta$ is the difference in pseudo-rapidity between the two jets and $\Delta\phi$ is the difference in azimuthal angle. Only parton jets with $p_T > 20\,\text{GeV}$ were considered for matching. This procedure resulted in $928,991$ parton jet–reconstructed jet pairs that were used to train the model.

3 Conditional Wasserstein Generative Adversarial Networks

The goal of our machine learning model is to generate samples $x \sim p(x|y)$, where x is the 4-momentum of a reconstructed jet given the 4-momentum of the associated parton jet y from the event generator. We use a cWGAN model with two extensions to the original Generative Adversarial Network (GAN) architecture proposed in [18]. A standard GAN consists of two neural networks: a generator G samples from a noise distribution $p_z(z)$ and produces an output x according to the generated distribution $p_g(x)$, while a discriminator D takes in samples from the generated distribution as well as samples from the true distribution $p_r(x)$ and determines from which distribution a given sample originated. During training D “learns" to differentiate between the generated distributions, while G learns to “fool" D by producing samples that are as realistic as possible. Given a perfect discriminator, in [18] it is shown that training a GAN is equivalent to minimizing the Jensen-Shannon divergence between p_g and p_r [18].

The first extension, the Wasserstein GAN introduced in [19], instead minimizes the Wasserstein distance between p_g and p_r, given by

$$W(p_r, p_g) = \max_{w \in \mathcal{W}} \mathbb{E}_{x \sim p_r(x)}[D_w(x)] - \mathbb{E}_{z \sim p_z(z)}[D_w(G_\theta(z))], \tag{1}$$

where the generator is parameterized by θ, the discriminator is parameterized by w, and $\mathcal{W}$ is the set of parameters such that D is a $K-$Lipschitz function for some K. In practice, a Wasserstein GAN is trained by alternating between several iterations of gradient *ascent* on D using the following loss function

$$L(\theta, w) = \frac{1}{N}\sum_{i=1}^{N} D(x_i) - \frac{1}{N}\sum_{i=1}^{N} D(G(z_i)),$$

in order to estimate the Wasserstein loss, followed by a step of gradient descent using G. In order to enforce the Lipschitz constraint, we used an additional "gradient penalty" term in the loss function as described in [20]. The discriminator in a Wasserstein GAN is termed the "critic" to emphasize that it no longer serves as a classifier.

The second extension is that of a conditional GAN [21]. A conditional GAN seeks to learn a conditional distribution $p_r(x|y)$, where y is some additional information such as a class label, or in our case, the 4-momentum of a parton jet. To do so, the generator accepts a noise vector z and the additional information y and produces a sample x. When trained with the Wasserstein distance, the critic takes (x, y) pairs either from the generator or from the true distribution. This results in the loss function

$$L(\theta, w) = \frac{1}{N}\left[\sum_{i}^{N} D(x_i|y_i) - \sum_{i}^{N} D(G(z_i, y_i)) + \lambda \sum_{i=1}^{N} (\|\nabla_{(\hat{x},\hat{y})} D(\hat{x}_i|\hat{y}_i)\| - 1)^2\right], \tag{2}$$

where the last term is the gradient penalty. To calculate the gradient penalty, the gradient of the critic is calculated with respect to inputs $\hat{x}$ and $\hat{y}$, which are convex combinations of data points from the true and generated distributions. The coefficient λ controls how strongly the gradient penalty is enforced.

4 Implementation Details

The cWGAN was implemented using the TensorFlow package [22] and the Keras interface [23]. All of the code used for training the model is available in a public GitHub repository [24]. Before the data were provided to the model, the features p_T and energy E were scaled by taking the base-10 logarithm and then all features were normalized to have a mean of zero and unit variance. Both the generator and critic are fully connected neural networks. A diagram showing the layers, nodes, and activation functions of each network is shown in figure 1.

During training, batches are formed by randomly sampling 64 elements from the dataset of matched parton jet and reconstructed jet 4-momenta, as well as sampling 64 "noise vectors" of length 10 from a uniform distribution over the interval $[0, 1]$. In a generator update step, the parton jet 4-momenta and noise vectors are passed into the generator, which then produces a batch of generated reconstructed jet 4-momenta. These generated reconstructed jet 4-momenta and their associated parton jet 4-momenta are passed into the critic, and the output of the critic is used to calculate the second term of equation 2, as it is the only term that depends on the generator. The gradient update is then performed using the RMSProp algorithm [25].

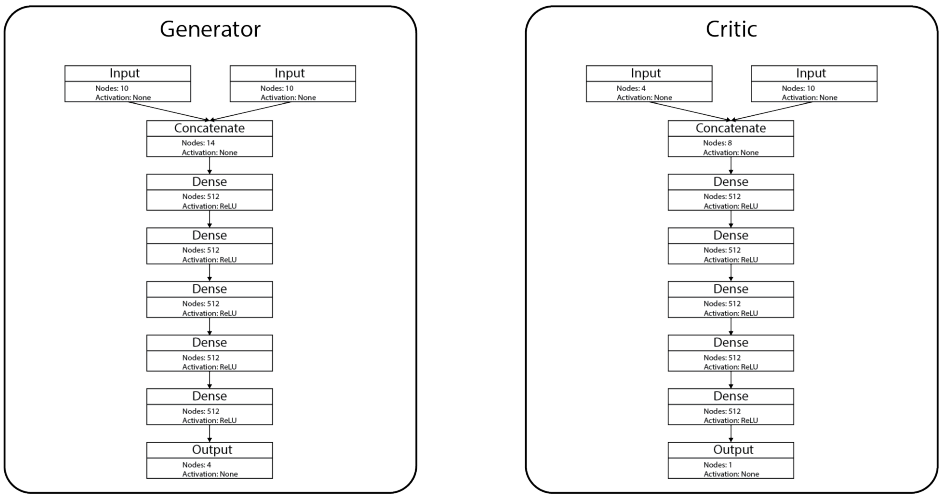

Figure 1. The generator and critic. Both networks have two inputs which are concatenated, followed by five dense layers with 512 nodes each, followed by an output layer.

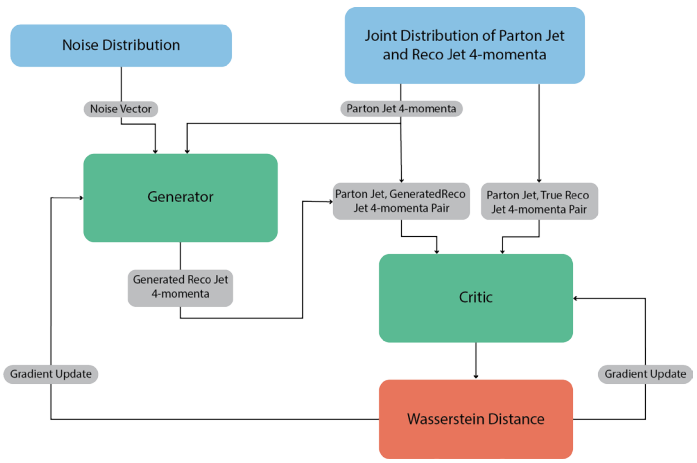

Figure 2. The cWGAN. The generator takes in parton jet 4-momenta and noise vectors, and outputs generated reco jet 4-momenta. The Critic takes in reco jet, parton jet 4-momenta pairs, and the output of the critic is used to estimate the Wasserstein distance between the real and generated distributions. The Wasserstein distance is then used for gradient updates for both networks.

For the discriminator update step, two batches of size 64 are sampled from the jet dataset, as well as a batch of noise vectors. The noise vectors, and one of the batches of parton jet 4-momenta are passed into the generator to create a batch of data from the generated distribution. Convex combinations are formed from the generated batch and the real batch, and then the real batch, generated batch, and combination batch are passed into the critic in order to calculate the loss as given in equation 2. The gradient update is performed using the RMSProp algorithm as well. A diagram of the flow of data inside of the cWGAN is shown in figure 2. We used a ratio of five critic updates for every generator update, and used a learning

rate of $5e-5$ for the critic and $1e-5$ for the generator. We set the gradient penalty coefficient λ to 10.

5 Results

In this section we refer to the reconstructed jets obtained from the GEANT4 simulation as the "true reco" jets, and the jets from the cWGAN as the "predicted reco" jets.

5.1 Timing

In this section we present results on the time taken to simulate events using our machine learning approach, and compare them with the time taken using `Delphes` as an example of a fast detector simulator. All times were generated using a Linux CentOS 8 machine with thirty-two Intel Xeon E5-2620 v4 CPUs running at 2.1 GHz. We first investigate the amount of time saved by skipping the hadronization step. Table 1 shows the amount of time taken to generate events in `Pythia 8` with and without the hadronization step, at a center of mass energy of 13 TeV with the hard subprocess of $gg \to t\bar{t}$. As seen in the table, once `Pythia 8` is being used to generate thousands of events, it appears that hadronization takes around 5-15% of the total event generation time.

Number of Events	Time with Hadronization (s)	Time without Hadronization (s)	Percent Change
5	0.65	0.69	+7%
50	1.09	1.19	+9%
500	8.04	5.95	-25%
5000	62.12	59.14	-5%
10000	129.85	110.70	-15%
50000	630.61	550.07	-13%%

Table 1. Time to generate events in `Pythia 8` with and without Hadronization.

We next compare the speed of our machine learning model and `Delphes`. To get times for the cWGAN, we generated events in `Pythia 8` without hadronization, clustered the partons in the event into jets using the anti-k_t algorithm as implemented in `FastJet` with distance parameter $R = 0.4$, and then passed the jets into the cWGAN. We called our TensorFlow model directly in our C++ code using the CppFlow library [26]. To get the times for `Delphes`, we ran `Pythia 8` inside `Delphes` using the "DelphesPythia8" executable offered as part of the `Delphes` software.

As a first comparison, Table 2 shows the amount of time taken to do the entire event simulation process using either the cWGAN or `Delphes` as the detector simulator. We observed that using the cWGAN reduced the entire event simulation time by over half.

We next compared the amount of time taken by both the cWGAN and `Delphes` in addition to the time taken by the event generator. To do this, we took the times listed in Table 2 and subtracted the amount of time taken to generate the events through hadronization with `Pythia 8`, cluster the stable particles into jets using the anti-k_t algorithm as implemented in `FastJet` with distance parameter $R = 0.4$, and write the jet four-momenta to disk. This comparison is shown in Table 3. When compared in this fashion, we observed that the cWGAN was over an order of magnitude faster than `Delphes`.

Number of Events	CPU time using cWGAN approach (s)	CPU time using Delphes (s)	Percent Change
5	1.90	1.61	-15%
50	2.35	2.86	+22%
500	7.53	13.72	+82%
5000	58.02	122.70	+115%
50000	578.98	1221.70	+110%

Table 2. Comparison of the CPU time used to simulate events using a cWGAN as the detector simulator, and using `Delphes` as the detector simulator.

Number of Events	Additional CPU time to simulate events using cWGAN (s)	Additional CPU time to simulate events using Delphes (s)	Percent Change
5	1.16	0.87	-25%
50	1.11	1.62	+46%
500	1.20	7.39	+516%
5000	2.35	67.03	+2752%
50000	34.24	676.96	+1877%

Table 3. Comparison of the CPU time used after using `Pythia8` to simulate events using a cWGAN and `Delphes`.

5.2 Momenta Predictions

For parton jet 4-momenta y and reco jet 4-momenta x, the matching procedure creates a joint distribution of 4-momenta $p(x, y)$. Figure 3 shows the marginal distribution $p_r(x) = \int p_r(x, y)dy$ from the training dataset and the marginal distribution predicted by the model $p_g(x) = \int p_g(x, y)dy$. As seen in figure 3, there are certain characteristics of the distributions the model does not capture, such as the dip at $\eta \approx \pm 3$ (a result of the CMS detector geometry). However, the model does learn the overall shapes of the marginal distributions.

Figure 4 shows the full joint distribution $p(x, y)$ of p_T for matched parton and reconstructed jets, where the counts of each histogram bin are log-scaled. It is apparent that the model is able to learn many important features of the joint distribution, including non-Gaussian effects for low p_T jets.

The true goal of the model is to sample from the conditional distribution $x \sim p_g(x|y)$. To evaluate the model's ability to produce accurate conditional distributions, we set p_T bins around a central value a, and then took every reco jet that matched a parton jet with $p_T \in [a - \delta, a + \delta]$ for a given bin width δ. As we make δ smaller, we approach the conditional distribution for $y = a$. Since we are approximating the density by drawing samples from the joint distribution, we are limited in our ability to make δ small by the size of our dataset. Several examples of such "conditional" distributions are shown in figure 5. The model shows the ability to predict accurate distributions conditioned on a range of parton jet p_T values. We note that even in the high parton jet p_T regime in which there is little training data, the model is still able to produce reasonable conditional distributions as seen in the bottom right of figure 5.

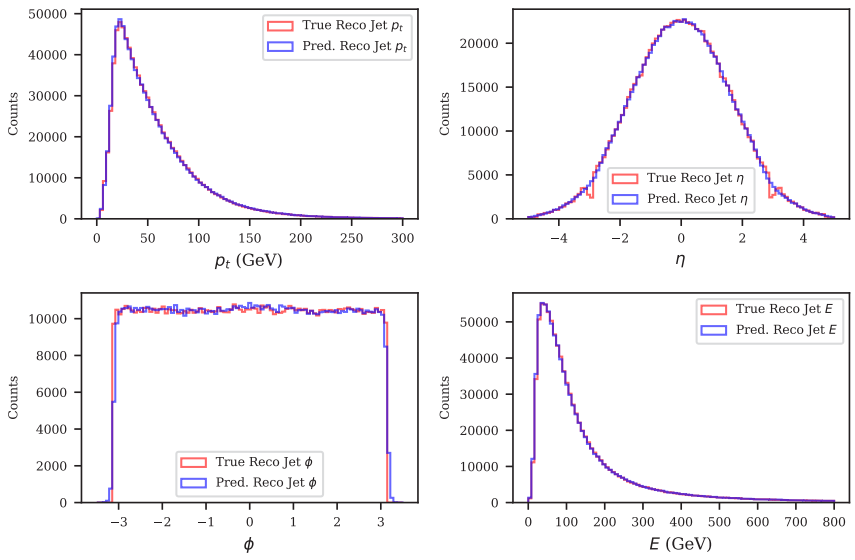

Figure 3. Marginal distributions of the four components of the four-momenta, p_T, η, ϕ, and E.

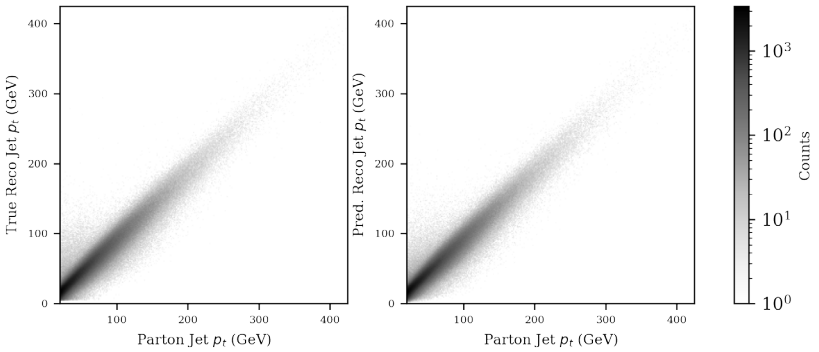

Figure 4. Joint p_T distribution of parton jet p_T and true reco jet p_T (left), and parton jet p_T and predicted reco jet p_T (right). The counts in each histogram bin have been log-scaled to emphasize the tails of the distribution.

6 Conclusions

In this study, we make use of advances in deep generative models and demonstrate the utility of cWGANs as a fast detector simulator for jets. The cWGAN is capable of producing realistic conditional distributions of reconstructed jet p_T in time that is orders of magnitude less than the conventional detector simulation process. Our model also presents an advantage over existing fast detector simulators in that it takes the partons in an event pre-hadronization as an input. This saves time in the overall event simulation as event generators do not need to be run all the way through hadronization.

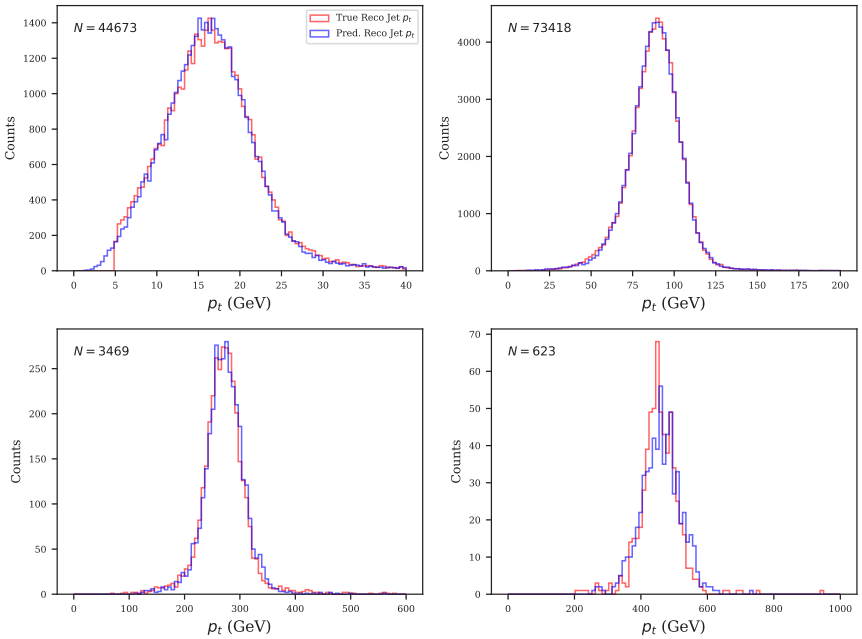

Figure 5. Conditional p_T distributions showing the distributions of true and predicted reco jet p_T for reco jets matching parton jets with p_T in the range: $[18, 22]$ GeV (top left), $[90, 110]$ GeV (top right), $[270, 330]$ GeV (bottom left), and $[450, 550]$ GeV (bottom right).

An important next step in the development of cWGANs for detector simulation is the addition of pileup interactions to the training data. We chose not to include pileup interactions in our simulated events in this preliminary study, but this is a potential avenue for further investigation.

7 Acknowledgments

We would first like to thank Dr. Harrison Prosper and Dr. Sergei Gleyzer for their guidance and advice over the course of this research, as well as Dr. Steve Mrenna and Dr. Nishita Dessai for their valuable suggestions on identifying the pre-hadronization parton-level event and Dr. Sezen Sekmen for numerous discussions about fast detector simulation. We also thank the CMS collaboration for making available CMSSW as well as both simulated and real data through the CMS Open Data portal.

References

[1] S. Agostinelli, J. Allison, K. Amako, J. Apostolakis, H. Araujo, P. Arce, M. Asai, D. Axen, S. Banerjee, G. Barrand et al., Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **506**, 250 (2003)

[2] J. De Favereau, C. Delaere, P. Demin, A. Giammanco, V. Lemaître, A. Mertens, M. Selvaggi, Journal of High Energy Physics **2014** (2014), 1307.6346

[3] M. Paganini, L. De Oliveira, B. Nachman, Physical Review Letters **120**, 1 (2018), 1705.02355
[4] M. Paganini, L.D. Oliveira, B. Nachman, Physical Review D **97**, 14021 (2018)
[5] M. Erdmann, J. Glombitza, T. Quast, Computing and Software for Big Science **3** (2019), 1807.01954
[6] Y. Lu, J. Collado, D. Whiteson, P. Baldi, Phys. Rev. D **103**, 036012 (2021)
[7] E. Buhmann, S. Diefenbacher, E. Eren, F. Gaede, G. Kasieczka, A. Korol, K. Krüger, arXiv preprint arXiv:2005.05334 (2020)
[8] A. Hariri, D. Dyachkova, S. Gleyzer, M. Awad, D. Morozova, *Graph Generative Models for Fast Detector Simulations in Particle Physics*, in *Machine Learning and the Physical Sciences Workshop at NeurIPS* (2020), 1, pp. 1–6, https://ml4physicalsciences.github.io/2020/files/NeurIPS_ML4PS_2020_138.pdf
[9] P. Musella, F. Pandolfi, Computing and Software for Big Science **2** (2018), 1805.00850
[10] R. Di Sipio, M.F. Giannelli, S.K. Haghighat, S. Palazzo, Journal of High Energy Physics, 2019(8), 110. (2019)
[11] CMS Collaboration, *CMS data preservation, re-use and open access policy* (2014), http://opendata.cern.ch/record/411
[12] CMS Collaboration, *Tracker-hit-enriched ttjets_hadronicmgdecays_8tev-madgraph* (2019), http://opendata.cern.ch/record/12200
[13] *Cms offline software* (2021), https://github.com/cms-sw/cmssw
[14] T. Sjöstrand, S. Ask, J.R. Christiansen, R. Corke, N. Desai, P. Ilten, S. Mrenna, S. Prestel, C.O. Rasmussen, P.Z. Skands, Computer Physics Communications **191**, 159 (2015)
[15] A.M. Sirunyan, A. Tumasyan, W. Adam, E. Asilar, T. Bergauer, J. Brandstetter, E. Brondolin, M. Dragicevic, J. Erö, M. Flechl et al., Journal of Instrumentation **12** (2017), 1706.04965
[16] M. Cacciari, G.P. Salam, G. Soyez, Journal of High Energy Physics **2008** (2008), 0802.1189
[17] M. Cacciari, G.P. Salam, G. Soyez, The European Physical Journal C **72** (2012)
[18] I.J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, *Generative adversarial networks* (2014), 1406.2661
[19] M. Arjovsky, S. Chintala, L. Bottou, *Wasserstein gan* (2017), 1701.07875
[20] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, A. Courville, *Improved training of wasserstein gans* (2017), 1704.00028
[21] M. Mirza, S. Osindero, *Conditional generative adversarial nets* (2014), 1411.1784
[22] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin et al., *TensorFlow: Large-scale machine learning on heterogeneous systems* (2015), software available from tensorflow.org, http://tensorflow.org/
[23] F. Chollet et al., *Keras*, https://github.com/fchollet/keras (2015)
[24] J. Blue, *alpha-davidson/falcon-cWGAN: Release for CHEP Submission* (2021), https://doi.org/10.5281/zenodo.4569082
[25] G. Hinton, N. Srivastava, K. Swersky, *Coursera neural networks for machine learning lecture 6*, http://www.cs.toronto.edu/~hinton/coursera/lecture6/lec6.pdf
[26] S. Izquierdo, *Cppflow*, https://github.com/serizba/cppflow