EPJ Web of Conferences **251**, 03051 (2021) https://doi.org/10.1051/epjconf/202125103051
CHEP 2021

Graph Variational Autoencoder for Detector Reconstruction and Fast Simulation in High-Energy Physics

Ali Hariri[1,*], *Darya* Dyachkova[2], and *Sergei* Gleyzer[3]

[1]American University of Beirut
[2]Minerva Schools at KGI
[3]University of Alabama

Abstract. Accurate and fast simulation of particle physics processes is crucial for the high-energy physics community. Simulating particle interactions with the detector is both time consuming and computationally expensive. With its proton-proton collision energy of 13 TeV, the Large Hadron Collider is uniquely positioned to detect and measure the rare phenomena that can shape our knowledge of new interactions. The High-Luminosity Large Hadron Collider (HL-LHC) upgrade will put a significant strain on the computing infrastructure and budget due to increased event rate and levels of pile-up. Simulation of high-energy physics collisions needs to be significantly faster without sacrificing the physics accuracy. Machine learning approaches can offer faster solutions, while maintaining a high level of fidelity. We introduce a graph generative model that provides effective reconstruction of LHC events on the level of calorimeter deposits and tracks, paving the way for full detector level fast simulation.

1 Introduction

Simulations of particle interactions are crucial for particle physics research. The Large Hadron Collider is delivering the highest energy proton-proton collisions ever recorded in the laboratory, permitting a detailed exploration of elementary particle physics. It is uniquely positioned to detect and measure the rare phenomena that can shape our knowledge of new interactions and possibly resolve the present tensions of the Standard Model. LHC experiments have already observed the long-sought after Higgs boson [1, 2], and it is hoped that they will further reveal evidence for new physics beyond the Standard Model.

After the next LHC run and the ensuing shutdown, the LHC will begin its 'High-Luminosity' (HL-LHC) operations around 2027 [3], targeting a five-fold increase in yearly data collection, with an additional ten years of operation. Advanced computational paradigms, including advanced machine learning techniques and dedicated hardware that accelerates their application for detector simulation and event reconstruction, will be necessary to attain the main physics goals of the HL-LHC [4] due to significant challenges posed by increased levels of pile-up and the rarity of sought-after signals.

*e-mail: aah71@mail.aub.edu

Probabilistically, the reconstruction of collision events is modeled using the equation:

$$p(\text{r-particles}|\theta) = \int R(\text{r-particles}|\text{particles})H(\text{particles}|\text{partons}) \times P(\text{partons}|\theta)\, d\text{particles}\, d\text{partons} \quad (1)$$

where P is the probability density of observing a set of reconstructed objects given a point in the parameter space, H refers to the hadronization process where mapping from the parton to the particle level occurs and R(particles) is the detector response [5]. The latter is estimated using Monte Carlo-based full simulation or parametric methods. This poses a computational bottleneck for current simulation frameworks that model collision events using Monte Carlo methods. The current forward simulation models [6] are difficult to parallelize and do not scale well with the rising computational complexity. Parametrized models such as [7] are faster but suffer from a loss in fidelity compared to full simulation which limits their utility.

1.1 Machine Learning for Fast Simulation

Many useful applications for particle physics that emerged from the recent advances in deep learning [4]. One of the actively researched topics is the potential of generative models for simulating collision events. In [8], a combination of a Variational Autoencoder (VAE) and a Generative Adversarial Network (GAN) is used to simulate electromagnetic showers in calorimeters. Other studies focus on GANs for QCD Dijet events [9] and hadronic jets [10].

In this study, we make use of VAEs [11] and geometric deep learning [12] to learn a compressed representation of the data to be used for reconstruction of high-energy physics events. More specifically, we develop a Graph Variational Autoencoder to learn the representations of single jet events data in a high dimensional space and then reconstructing them in an end-to-end fashion through an encoder and a decoder, setting the ground for future work to embed such generative models into a full simulator. In contrast to existing methods, *graph representation learning* can handle irregular grids with non-Euclidean geometry [13], encode physics knowledge via graph construction [14] and introduce *relational inductive bias* into data-driven learning systems [15]. In our study we use spatial graph convolutional layers [16] to learn the properties of the graph-like structures and spectral clustering layers to compress these graphs into smaller, more representative nodes. Graph neural networks have been recently applied to other applications in high-energy physics [17].

2 Data and Pre-Processing

For the study, we use a representative sample of top quark pair events generated with Pythia 6 and available on the CERN Open Data Portal [18]. The input data is processed by following the recipe in [19]. The input data contains a total of 30000 samples, each containing 3 channels: Tracks, Electromagnetic Calorimeter (ECAL) and Hadronic Calorimeter (HCAL), respectively, all of which are target channels for the purpose of this work. For HCAL and ECAL information is consumed at the level of reconstructed hits. For Tracks the information is contained at the level of the projected track to ECAL surface. To visualize the events in different sub-detectors, we display them on a mesh with 85 x 85 segmentation (Fig. 1), shown at ECAL resolutyion.

We pre-process the data by selecting the non-zero hit locations within the array, providing their respective x and y locations as per the calorimeter segmentation. Afterwards, we concatenate the x,y locations with their corresponding reconstructed hit energy at that location. At this stage, each sample has the shape Nx3 where N is the number of non-zero hits within

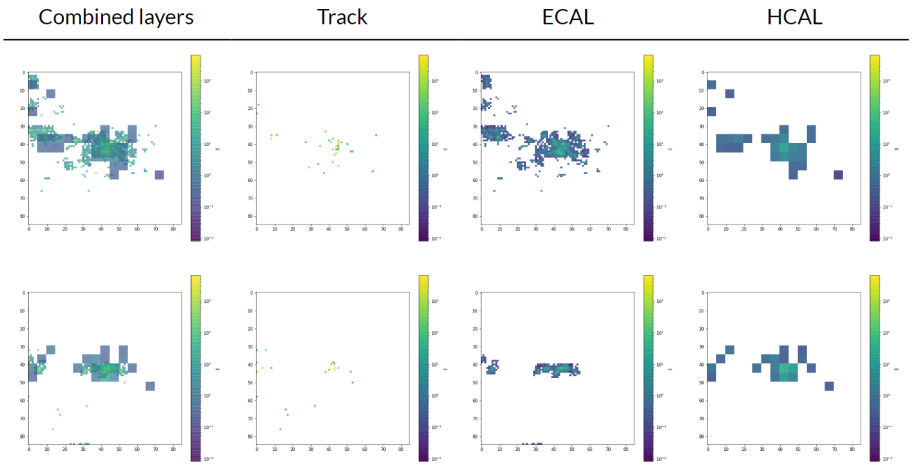

Figure 1. The visualization of the energy deposits in ECAL and HCAL calorimeter layers for two distinct *ttbar* events (top and bottom). The tracks layer shows the tracks projected to the ECAL surface. The leftmost image is the combined one from all the three layers.

the detector. In the next section we show that N is also the number of nodes within one graph representing a jet.

3 Graph Representation of Events

A graph is denoted by $G = (V, E, A)$ where V is the set of vertices composing the graph, E the set of edges connecting these vertices, and A being an N x N adjacency matrix describing the relationship between the nodes, where N is the total number of nodes in the graph. Associated with each vertex V is a set of features describing it. These are given in a feature vector $X \in R^{N \times D}$ with D being the number of features per node. After performing the convolution operation on neighbor node attributes, this signal needs to be aggregated and propagated k times in order to update the features of the node under study for the next iteration. A hidden Graph Convolutional Network (GCN) is given by $H^i = f(H^{i-1}, A)$ where $H^i = N \times X^i$ represents the node features at iteration i and f is a propagation function that defines the output based on the input. For the propagation function, rectified linear unit (ReLU) is the one most commonly used, defined as $f = max(0, x)$, x being the input [20]. The simplest form of the propagation rule accounts for the node features by summing them prior to aggregation. This method known as the sum rule is given by: $h_{vi} = A_i X = \sum_{j=1}^{N} A_{ij} X_j$ where h_{vi} represents features of node v at the i^{th} iteration.

Jets in particle collisions are not characterized by such pre-defined topology. We map particle showers at the level of the CMS detector into graphs by considering a single particle hit to be a node within a larger graph whose interconnected components are the surrounding hits. In other words, we consider particle hits within a detector to be interconnected nodes in a graph. This mapping operation is done on the pre-processed image samples obtained from the CMS open data, serving as an input to our model.

Therefore, each event is characterized by a set of nodes whose features include x-y locations in 2D space in addition to their energies. We use the k-nearest neighbours algorithm in 2D space to connect the nodes, i.e each node is connected by an edge to k-nodes that are closest in terms of Euclidean distance given by $\sqrt{(x - x_i)^2 + (y - y_i)^2}$ with x_i and y_i referring

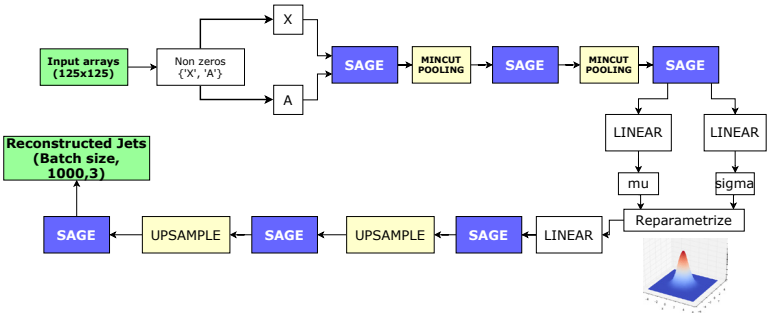

Figure 2. Model architecture of the Graph Variational Autoencoder showing GraphSAGE layers and pooling blocks

to this node's coordinates. To learn the graph properties of these showers we develop a Graph VAE architecture whose encoder embeds the node features into latent space dimensions.

After defining the graph topology and node features, we also need to address the graph embeddings. We use GraphSAGE [21], an inductive learning framework that assumes nodes within the same neighbourhood to have similar embeddings and proceed by aggregating information from higher-level neighbourhoods. Hence, we use dense GraphSAGE layers in a graph variational encoder-decoder model that learns to reconstruct graphs from a learned compressed representation in latent space. The latter is obtained at the level of the encoder by means of spectral clustering of the graph's nodes with similar latent features. More specifically, the graph is compressed using the MinCut pooling technique inspired by [22]. Finally, the decoder upsamples the feature and adjacency matrices as follows:

$$X^{rec} = S\,X^{Pooled}; A^{rec} = S\,A^{Pooled}S^{T} \tag{2}$$

where S is a learned cluster assignment matrix similar to the one defined in [22]. A pictorial representation of our model is given in Figure 2.

In a nutshell, our GVAE model takes as input a set of pre-processed images from the CMS Open Data release, maps them as interconnected graphs using the k-nearest neighbour approach and learns their latent space representation for reconstruction purposes.

4 Results

Our Graph Variational Autoencoder was trained on a Tesla T4 GPU on Google Colab [23] to reconstruct top quark-initiated jets. A clear visualization of our results is shown in Figure 3, where the fully-simulated events are compared to their corresponding GVAE reconstruction using the x-y coordinates of the hits on the one hand and the particle energies represented in the farmost color bar on the other hand. Our results indicate a high-level of agreement in 2D hit distributions and their respective energy scales. Similarly, the Earth Mover Distance metric [24] used to compute the cost of moving the predicted point clouds likewise indicates good agreement (Figure 4) while accounting for the energy and distance differences between real and reconstructed events. This metric represents the "work" that is needed to convert one particle event ε into another ε' by moving energy between particles i and j in the two events.

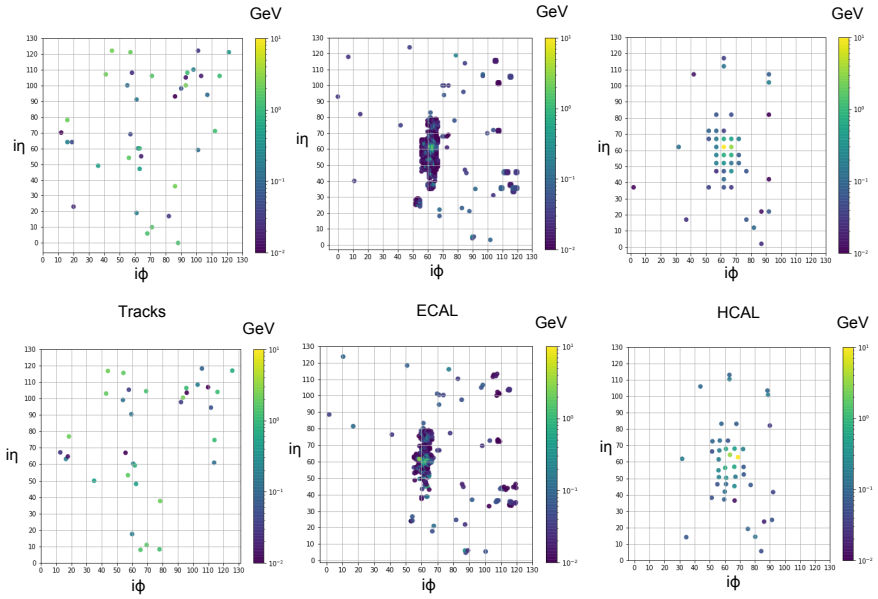

Figure 3. Original simulated top quark initiated jet (top) compared to the GVAE-reconstructed jet (bottom) in each of the three channels. The energy range is log-scaled for better visualization.

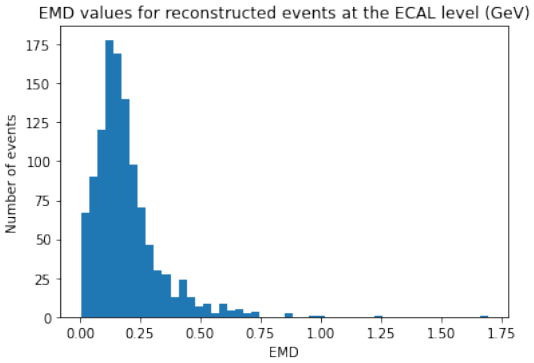

Figure 4. Earth Mover Distance distribution as described in Ref. [24]. Low values of EMD indicate good agreement.

It is described by the following equation:

$$EMD(\varepsilon, \varepsilon') = \min_{\{f_{i,j}\}} \sum_{ij} f_{ij} \frac{\theta_{ij}}{R} + \left| \sum_i E_i - \sum_j E'_j \right|$$

$$f_{ij} \geq 0, \qquad \sum_j f_{ij} \leq E_i, \qquad \sum_i f_{ij} \leq E'_j, \qquad \sum_{ij} f_{ij} = E_{min}$$

Finally, several trial runs showed an average inference time of 0.13088 seconds for GVAE reconstruction on a batch size of 64, orders of magnitude below typical full reconstruction times for simulation of top events, while maintaining a very high level of fidelity.

4.1 Multi-GPU Scaling

We next investigate how the GVAE model scales on multiple GPU devices. We first profile the code to monitor the process load taking place in CPUs and GPUs. To avoid a potential data-loading bottleneck, we opted to generate batches of graphs on the spot in the CPU prior to sending them to the training model on the GPU. This change has reduced the training time by 50%. Having trained on a single Tesla V100 GPU as a baseline, we proceeded by scaling the model using the Horovod library [25], a deep learning library for distributed training of deep learning models supporting many frameworks such as PyTorch, TensorFlow and MXNet. This framework facilitates distributed training across multiple GPUs using the Message Passing Interface (MPI) and takes as input the *size* parameter referring to the number of processes P in training, followed by the *Rank* parameter unique to process and numerated from 0 to P-1. Finally, *LocalRank* indicates the unique process ID within each device (0 to N_{GPUS}).

We train our GVAE model on a cluster using Volta V100 GPUs having 16 GB of RAM. Following profiling and code optimization for enhanced CPU performance, we scale the training on multiple GPUs using the Horovod library. To get a baseline performance, we compare the results to the training on a single GPU with a batch size of 32 for 100 iterations, i.e a total of 3200 graph samples (Figure 6).

Horovod Weak Scaling on p GPUs with p={1,2,3,4} for 100 iterations on NVIDIA DGX V100				
	GPU Processes execution time (in seconds) for 3200 samples			
	One	Two	Three	Four
Mean Execution Time (s)	69.34	85.48	94.96	101.54
Stddev Execution Time (s)	3.05	1.85	2.26	1.00
Speedup	1.00	1.62	2.19	2.73
Parallelization efficiency	1.00	0.81	0.71	0.68

Figure 5. Table showing scaling efficiency

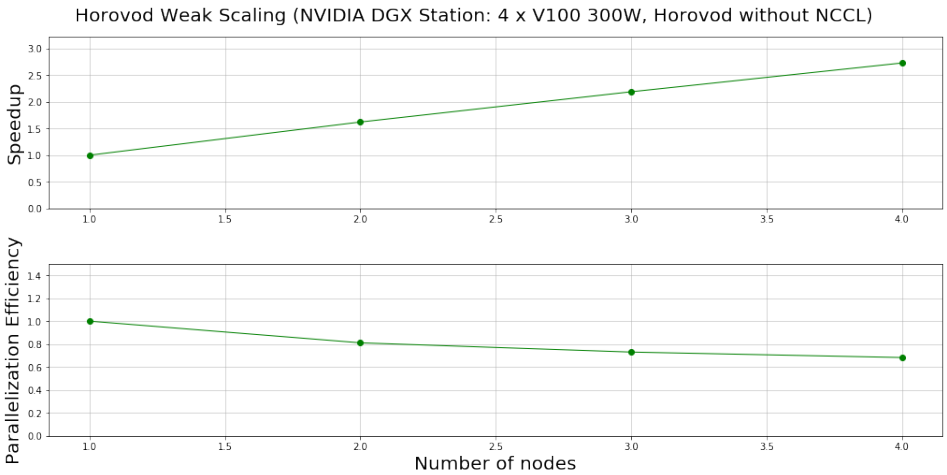

Figure 6. Comparison plot of how our model scales to multiple GPUs

Throughout scaling we notice an increase in the performance with an increase in the GPU devices used. To calculate the resulting speedup from scaling, we take as reference the Mean Execution Time (MET) resulting from training on one GPU. For $N_{GPUs} = 2$, the mean execution time is 85.48 seconds. Bearing in mind that we are training twice the amount of data overall, we get MET per GPU = 42.74 seconds. The resulting speedup for $N_{GPUs} = 2$ is given by

$$\frac{MET_{singleGPU}}{MET\,per GPU using N_{GPUs}} = \frac{69.34}{42.74} = 1.62 \tag{3}$$

Applying the same speedup calculation, going from 2 to 4 GPUs, we get speedups of 1.62, 2.19 and 2.73, respectively. Therefore, we conclude that our model scales well on multiple GPUs using Horovod, which is useful for future work and similar graph generative models. However, there is still some loss in scaling performance due to parallelization efficiency that can be explained by an inefficient communication between the GPUs once parallel batch training has been done. This issue can be overcome by programming the kernel directly. Further optimization could be deferred to future work.

Additionally, we used NVIDIA's Automatic Mixed Precision (AMP) library that enables mixed precision training of deep learning models with support to distributed parallel training. This approach has the potential to boost the speedup of model training by enabling the support of FP16 operations throughout the model's layers. This step offers several benefits such as the reduction in required memory for FP16 as compared to 32-bit floating points, faster data transfer and linear algebra operations with lower precision formats. Using multi-gpu scaling methods allows us to additionally speed-up the performance during training and inference, something that is very challenging to accomplish with present simulation methods.

5 Conclusion

In this proof-of-concept work we shed light on the potential of graph-based architectures for learning the representation of high-energy collision events. Graph neural networks tackle the issue of data sparsity in particle detectors by allowing the model to directly learn from the hits deposited in the detector. Through spatial convolution our model was able to learn the interactions between hits in the detector and we sequentially compressed it by means of mincut pooling to preserve the most representative nodes in latent space. Finally, a trained decoder can upsample the compressed vectors to the original reconstruction, leading to a proof-of-concept simulator. We additionally benchmarked our model on single and multiple gpus, demonstrating low latency times and efficient multi-gpu scaling performance.

6 Acknowledgments

We would like to thank Michael Andrews, Bjorn Burkle, Christian Hundt and Giuseppe Fiameni for useful discussions. We thank them for their expertise and support. Ali Hariri was a participant in the Google Summer of Code 2020 Program. We additionally thank NVIDIA, Helmholtz Association and Mozhgan Kabiri Chimeh for support during a gpu hackathon that enabled us to study multi-gpu scaling.

References

[1] CMS Collaboration, Physics Letters B **716**, 30 (2012), 1207.7235
[2] ATLAS Collaboration, Physics Letters B **716**, 1 (2012), 1207.7214

[3] G. Apollinari, I. Béjar Alonso, O. Brüning, M. Lamont, L. Rossi, CERN-2015-005, FERMILAB-DESIGN-2015-02 (2015)
[4] K. Albertsson et al., arXiv e-prints (2018), 1807.02876
[5] S. Gleyzer, P. Orlando R. D., S. H.B., Sekmen, O. Zapata, arXiv preprint arXiv:1203.1488 (2016)
[6] R. Brun, R. Hagelberg, M. Hansroul, J. Lassalle, Report Number CERN-DD-78-2 (1978)
[7] J. de Favereau, C. Delaere, P. Demin, A. Giammanco, V. Lemaitre, A. Mertens, M. Selvaggi, Journal of High Energy Physics **2014(2), 57.** (2014)
[8] E. Buhmann, S. Diefenbacher, E. Eren, F. Gaede, G. Kasieczka, A. Korol, K. Krüger, arXiv preprint arXiv:2005.05334 (2020)
[9] R. Di Sipio, M.F. Giannelli, S.K. Haghighat, S. Palazzo, Journal of High Energy Physics, 2019(8), 110. (2019)
[10] P. Musella, F. Pandolfi, Comput Softw Big Sci 2: 8 (2018)
[11] D. Kingma, M. Welling, Foundations and Trends in Machine Learning **Vol. 12 (2019) No. 4**, pp 307 (2019)
[12] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, M. Sun, arXiv e-prints arXiv:1812.08434 (2018), 1812.08434
[13] J. Bruna, W. Zaremba, A. Szlam, Y. LeCun (2013), 1312.6203
[14] D. Zheng, V. Luo, J. Wu, J.B. Tenenbaum (2018), 1807.09244
[15] P.W. Battaglia, J.B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner (2018), 1806.01261
[16] T. Danel, P. Spurek, J. Tabor, M. Smieja, L. Struski, A. Slowik, L. Maziarka, arXiv e-prints, arXiv-1909 (2019)
[17] J. Shlomi, P. Battaglia, J.R. Vlimant, Machine Learning: Science and Technology **2(2, 021001)** (2020)
[18] *CERN Open Data Portal*, http://opendata.cern.ch
[19] M. Andrews, M. Paulini, S. Gleyzer, B. Poczos, Computing and Software for Big Science **4**, 1 (2020)
[20] I. Goodfellow, Y. Bengio, A. Courville, *Deep Learning* (MIT Press, 2016)
[21] W. Hamilton, R. Ying, J. Leskovec, Conference Notes from 31st Conference on Neural Information Processing Systems (2017)
[22] F. Bianchi, D. Grattarola, C. Alippi, Proceedings of Machine Learning Research (2010)
[23] E. Bisong, *Google Colaboratory* (2019), pp. 59–64, ISBN 978-1-4842-4469-2
[24] P. Komiske, E. Metodiev, J. Thaler, Physical review letters **123(4), 041801** (2019)
[25] A. Sergeev, M. Del Balso, arXiv preprint arXiv:1802.05799 (2018)