Electromagnetic shower generation with Graph Neural Networks

V. Belavin[1], **A. Ustyuzhanin**[1,2]

[1]National Research University Higher School of Economics, Myasnitskaya 20, 101000 Moscow, Russia

[2]Yandex School of Data Analysis, Timura Frunze 11 bld 2, 119034 Moscow, Russia

E-mail: vbelavin@hse.ru

Abstract.

In this work, we propose an approach for electromagnetic shower generation on a track level. Currently, Monte Carlo simulation occupies 50-70% of total computing resources that are used by physicists experiments worldwide. Thus, speedup of the simulation step allows to reduce simulation cost and accelerate synthetic experiments. In this paper, we suggest dividing the problem of shower generation into two separate issues: graph generation and tracks features generation. Both these problems can be efficiently solved with a cascade of deep autoregressive generative network and graph convolution network. The novelty of the proposed approach lies in the application of graph neural networks to the generation of a complex recursive physical process.

1. Introduction

Electromagnetic showers analysis is an essential part of almost all physical experiments because there are a lot of processes where electrons and gamma rays are produced. These particles produce electromagnetic cascades (showers) while entering the calorimeter. This process accounts for a significant part of the energy release. At this moment the most convenient approach for shower generation is Monte-Carlo Radiation Transport simulation. All interactions with matter are simulated in consequent order which leads to massive calculations that are taking up to 70% of all CPU time spent on simulation [1]. To address the performance issue there are several propositions made so far.

First of all, fast parameterization is proposed in [2]. Authors suggest describing shower with three density functions describing longitudinal, transversal and azimuthal energy distribution respectively. This approach heavily relies on the assumption that these distributions are sufficient to describe the spatial energy distribution. The quality of simulation depends on the quality of parametrization. For longitudinal development, there exists a well-known parametrization, but parameterization for the transverse profile is not determined with the same precision [2].

Secondly, ATLAS collaboration has proposed frozen showers approach in 2008 [3]. The key idea is to use the library with pre-simulated showers, i.e. prior to production usage. Large-sized library of showers has to be preserved on a disk for different initial directions, energies and positions. During the simulation, when a particle enters a detector, simulator checks conditions for the usage of the library of particles. If conditions are met shower from the library is sampled.

This method is best suited for medium-energy particles and suffers from expressiveness due to the size limitation of the preserved library.

Thirdly, "Killing" method is used in a case of low energy particles. It prevents further Monte-Carlo simulation, because it have negligible effect on physical analysis. This method is only applied for low-energy particles.

These approaches can be combined [3], i.e. high-energy(> 10 GeV) particles are simulated with fast parameterization, medium-energy(< 1 GeV) - with shower library approach and low-energy particles are killed. In the field of machine learning, methodology of 3D-calorimeter response simulation via generative adversarial networks is proposed [1].

We should note that all approaches that we have discussed are trying to simulate calorimeter response to the showers that are passing through it. And none is aimed to simulate tracks, that are generated by particles passing through the calorimeter. In other words, there are no works so far that are trying to generate EM-cascades on the track level. Or to be more explicit, simulation of electromagnetic showers in emulsion cloud chambers (ECC), which is a special type of sampling electromagnetic calorimeters.

Our proposed methodology, firstly, tries to extend deep learning techniques to the field of shower generation and, secondly, addresses an issue of a computationally inefficient generation with Monte-Carlo methods.

The code is available in https://gitlab.com/SchattenGenie/shower_generation

2. Data sample

Training data for our experiments have been generated with open source ship software FairShip [4]. FairShip is used in SHiP experiment [5] for simulation purposes.

ECC Brick consists of lead sheets and emulsion films layers which are glued to a plastic base. The emulsion works like a camera, taking a photo of particles passing through it. Each brick consists of 57 emulsion films. Each of the films plays the role of the calorimeter and is used for track detection from the electromagnetic shower. Each track is described by 5 variables: its location in detector(three cartesian coordinates, i.e. x, y, z) and its direction($\theta_x,\ \theta_y$). θ_x is a slope of the track in X projection. All showers are produced by particles moving in a positive direction of z-axis perpendicular to XY plane.

3. Proposed methodology

To solve the problem of shower generation in ECC Bricks we suggest to reformulate it as a combination of **graph generation** with subsequent **tracks features generation**. This allows us to divide the problem into two simpler ones. In the graph, each node is associated with one track in the shower.

We suggest this approach because the electromagnetic shower is a structured object that could be naturally represented with graph, in other words, this parameterization naturally arises from the underlying physics of the process.

3.1. Data prepossessing

Firstly, we need to transform shower from $(x, y, z, \theta_x, \theta_y))_{i=0}^{N}$-representation to directed graph(Fig. 1, 2). Firstly, we create an empty graph with a number of nodes equal to a number of tracks in the shower and some random one-to-one mapping. Then we choose the graph node, corresponding to the very first shower track, and connect it with the graph node, corresponding to the subsequent track, via edge.

By "subsequent tracks " we mean pairs of tracks that can be classified in one of two groups:

- two tracks that are produced by one particle and subsequent in z-dimension;

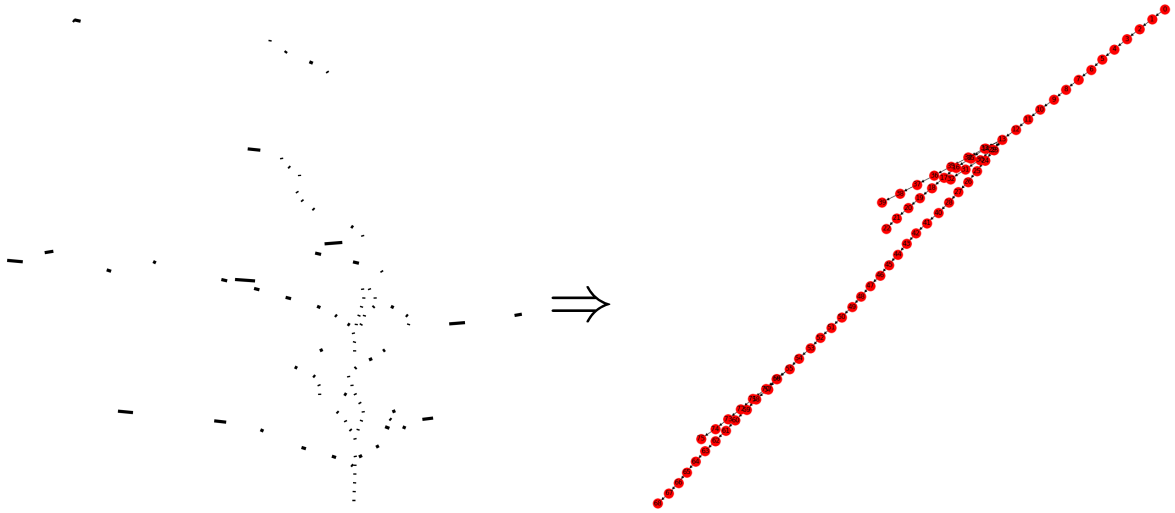

Figure 1. Example of one EM-shower in ECC Brick

Figure 2. Graph representation of the shower

- two tracks that are produced by two particles in a case when one particle is a child of another particle, one track is the first track of one particle and another track is the last track of another particle.

By connecting "subsequent" tracks we get a special type of directed graph: directed rooted tree or arborescence.

Secondly, to apply geometric deep learning we have to construct an adjacency matrix. To do so we need to define an order of a graph, i.e. propose some order of nodes. To reduce computational complexity we want non-zero elements of adjacency matrix to be closer to the diagonal. To achieve this we apply the Breadth-First traversal algorithm to the graph and sort rows and columns of adjacency matrix according to the traversal order of nodes given by this algorithm.

Also, we apply data augmentation to prevent overfitting: dropping random tracks and rotating shower around the z-axis.

3.2. Graph generation

Graph generation is performed with GraphRNN[6]. It is an autoregressive deep model where nodes are generated in sequential fashion one by one, followed by sequential generation of edges. Thus two recurrent neural networks are needed in this process: graph-level RNN that maintains the state of the graph and generates new nodes and edge-level RNN that generates edges in the

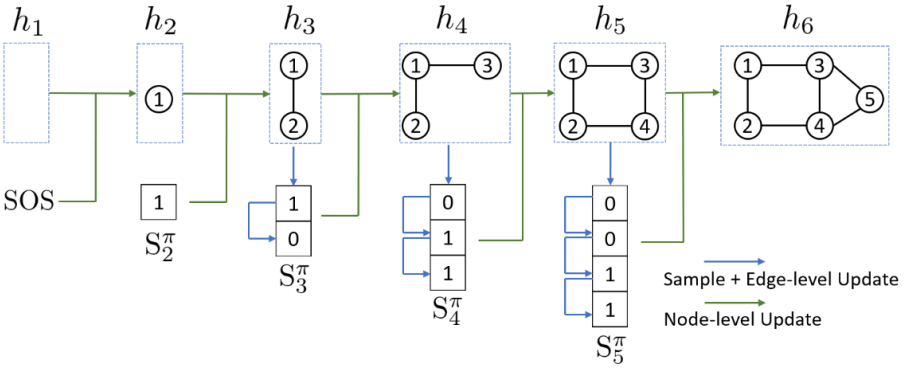

Figure 3. GraphRNN algorithm: node-level and edge-level updates are done with reccurent neural networks

graph(Fig. 3).

Algorithm 1: GraphRNN algorithm

Input: graph-level RNN $f(\cdot,\cdot)$, edge-level RNN $g(\cdot,\cdot)$, h – initial hidden state for graph-level RNN, ;
S_0 – an array of edges for node 0;
while S_i *is not EOS* **do**
  $i = i + 1$;
  $h_i = f(h_{i-1}, S_{i-1})$;
  S_i = zero array with length M + 1;
  for j *in* $1 \dots M$ **do**
    $S_i[j] = g(h_i, S_i[j-1])$
Output: S;

In this algorithm, S_i is an array of edge connections of node i with M previous nodes indicators. In other words, we are only reconstructing a diagonal band of the adjacency matrix of width M. It allows us to reduce the complexity of graph generation from N^2, i.e. full adjacency matrix generation, to NM, i.e. only prediction of diagonal band.

3.3. Signal generation

To make our network translation-invariant we predict changes of features for "subsequent" tracks instead of straightforward prediction of raw track features. However, due to the stochastic nature of shower development, the problem cannot be solved with naive regression, thus in this paper, we model distributions of features changes.

We are using graph generated from the previous step and embeddings of nodes provided by graph-level RNN. These embeddings are fed into graph convolution network[7]. This is a generalization of convolution networks that can be applied not only for regular N-dimensional grids but for arbitrarily structured graphs.

To be more precise in this work we are using Edge Convolutions[8] that have been used for point cloud segmentation problems, i.e. works well with geometric data.

To predict distributions of tracks features differences Mixture Density Network[9] is applied. The main idea behind MDN is to predict three groups of variables with a dense neural network: means of multivariate normal distributions – $\boldsymbol{\mu}_i$, diagonal covariance matrix – $\boldsymbol{\Sigma}_i$ and mixing coefficients – α_i. With this parameterisation we can model any arbitrary distributions as a mixture of normal distributions:

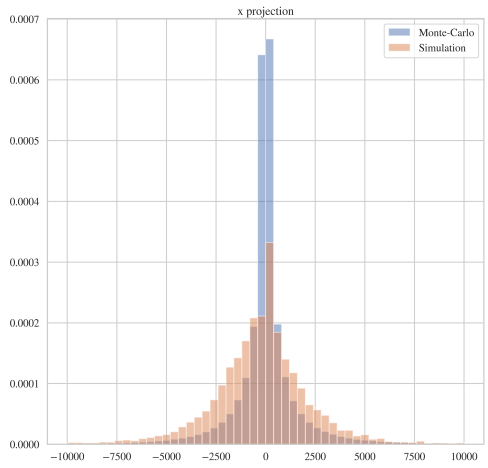

Figure 4. Distributions of x for MC and generated samples

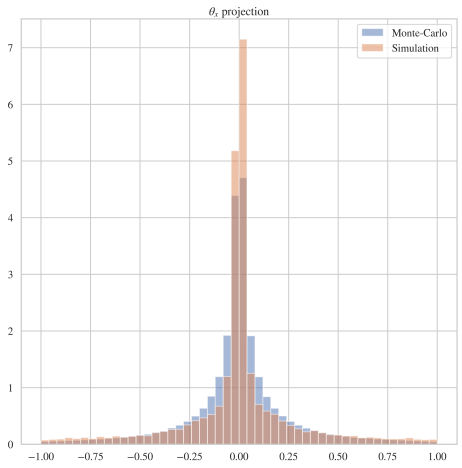

Figure 5. Distributions of θ_x for MC and generated samples

$$p(\boldsymbol{x}) = \sum_{k=1}^{K} \frac{\alpha_k}{(2\pi)^{1/n}|\boldsymbol{\Sigma}_k|} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_k)\right) \tag{1}$$

Network training was performed with the maximum log-likelihood approach.

3.4. Architecture and training

GraphRNN contains two blocks of RNNs for graph level generation and edge level generation. Graph-level RNN uses 4 layers of GRU cells with 196-dimensional hidden size each. The edge-level RNN uses 4 layers of GPU cells with 32-dimensional hidden state each.

The graph-level RNN hidden state is initialized with a small neural network that takes as input one feature – energy. The edge-level RNN hidden state is initialized with hidden state of graph-level RNN.

For a signal generation, we use teacher forcing[10] that forces the true graph to be an input for our neural network. GCN consists of 3 EdgeConv layers, each EdgeConv layer has a dense layer with 16 neurons and tanh that performs a learnable transformation of nodes embeddings. Mixture Density Network predicts parameters for 4 normal distributions and mixing coefficients.

We use two losses. One is a logistic loss for edge predictions in a graph:

$$\mathcal{L}_{edge} = \sum_{i,j} y_{i,j} \log p_{i,j} + (1 - y_{i,j}) \log(1 - p_{i,j}) \tag{2}$$

Another one is a negative log-likelihood for track features prediction:

$$\mathcal{L}_{nll} = -\sum_{i,j} \log p_{ij}(\Delta x, \Delta y, \Delta z, \Delta\theta_x, \Delta\theta_y). \tag{3}$$

Training is done with the following settings:

- 10000 showers in the data set with energies samples from uniform distribution: $E \sim \mathcal{U}[1\text{GeV}, 5\text{GeV}]$;
- M is equal to 12;
- 300 showers in one batch;

- dropout rate is 10%;
- GraphRNN trained for 200 epoch with Adam optimizer with lr $= 10^{-3}$;
- GCN and MDN are trained for 200 additional epochs with Adam optimizer with lr $= 10^{-3}$.

For showers with energies in the range $E \in \mathcal{U}[1\text{GeV}, 5\text{GeV}]$ number of nodes varies from ~ 40 to ~ 250. M was chosen so that the number of showers that could not be recovered is negligible. For $M = 12$ only 2.5% of showers from 10000 could not be recovered.

In Algorithm 2 shower generation procedure with trained GraphRNN, GCN and MDN models is presented.

Algorithm 2: Shower generation algorithm

Generate graph G with GraphRNN;
Predict $p_{ij}(\Delta x, \Delta y, \Delta z, \Delta\theta_x, \Delta\theta_y)$ with GCN and MDN ;
for $\forall$ *edges* $(i, j) \in G$ **do**
  sample $\Delta x, \Delta y, \Delta z, \Delta\theta_x, \Delta\theta_y \sim p_{ij}(\Delta x, \Delta y, \Delta z, \Delta\theta_x, \Delta\theta_y)$;
  $\text{track}_j = \text{track}_i + [\Delta x, \Delta y, \Delta z, \Delta\theta_x, \Delta\theta_y]$;
return tracks ;

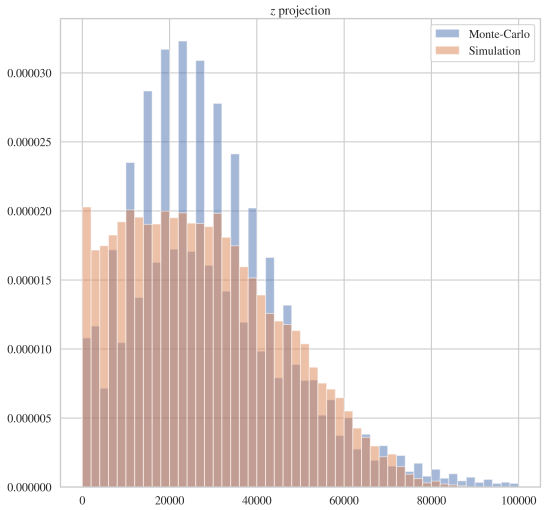

Figure 6. Distributions of z for MC and generated samples

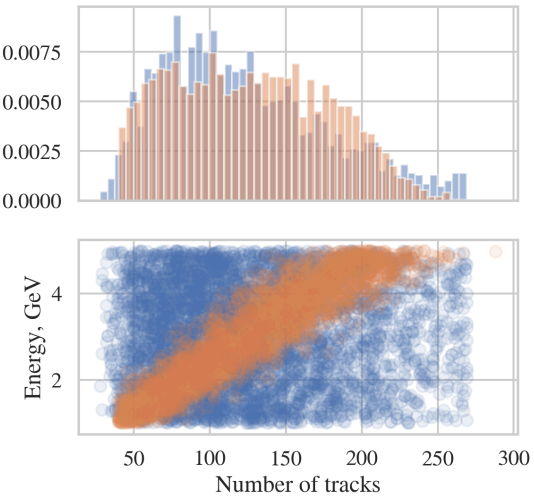

Figure 7. Marginal and conditional distributions of number of tracks in shower

4. Conclusion

Looking at distributions of x, θ_x, z(Fig. 4, 5, 6) one can compute Kolmogorov–Smirnov statistics to evaluate performance of our generative model. They are equal to 0.219, 0.177 and 0.154 respectively, which shows a noticeable discrepancy between generated samples and Monte-Carlo. Also, even though the neural network observes the energies of showers during training and should be a conditional generator, one can observe at Fig. 7 that network learns the correct marginal distribution of showers length but is not able to recover dependency of shower size as a function of energy. In further research, we are going to address this issue as well as ways to improve agreement between generated samples and Monte-Carlo samples.

Overall, in this work we propose a novel approach for shower generation on track level which has the following advantages:

- scales linearly with the number of tracks in a shower;
- allows exact log-likelihood estimation;
- can be interpreted easily – features distributions could be visualized and analysed analytically.

5. Acknowledgements

The research leading to these results has received funding from Russian Science Foundation under grant agreement 19-71-30020.

References

[1] Paganini M, Oliveira L and Nachman B 2018 CaloGAN: Simulating 3D high energy particle showers in multi-layer electromagnetic calorimeters with generative adversarial networks *Phys. Rev. D* **97** 2018

[2] G Grindhammer, S Peters 1993 The parameterized simulation of electromagnetic showers in homogeneous and sampling calorimeters *Int. Conf. on Monte Carlo Simulation in High Energy and Nuclear Physics*, (Tallahassee, Florida, USA)

[3] Barberio E at al 2009 Fast simulation of electromagnetic showers in the ATLAS calorimeter: Frozen showers *Journal of Physics: Conference Series* **160** 012082

[4] FairShip Project 2019 FairShip [software] Available from `https://github.com/ShipSoft/FairShip` [accessed 2019-05-24]

[5] Alekhin S, Altmannshofer W, Asaka T, Batell B, Bezrukov F, Bondarenko K and Curtin D 2016 A facility to search for hidden particles at the CERN SPS: the SHiP physics case *Reports on Progress in Physics* **79(12)** 124201

[6] You J, Ying R, Ren X, Hamilton W and Leskovec J 2018 GraphRNN: generating realistic graphs with deep auto-regressive models *International Conference on Machine Learning* pp. 5694-5703

[7] Kipf T N and Welling M 2017 Semi-Supervised classification with graph convolutional networks CoRR, abs/1609.02907

[8] Wang Y, Sun Y, Liu Z, Sarma S E, Bronstein M M and Solomon J M 2018 Dynamic Graph CNN for Learning on Point Clouds CoRR, abs/1801.07829

[9] Bishop C M 1994 Mixture density networks *Technical Report NCRG/4288* p. 7

[10] Williams R J, Zipser D 1989 A Learning Algorithm for Continually Running Fully Recurrent Neural Networks *Neural Computation* **1** p. 270-280