Federal State Autonomous Educational Institution for
Higher Education National Research University
Higher School of Economics

Faculty of Computer Science
BSc Applied Mathematics and Information Science

BACHELOR'S THESIS

Research project

Creating of GAN for the Fast Generation of Calorimeters Responses in Elementary Particle Detectors in CERN

Submitted by Fakanov Pavel Alexandrovich
student of group 171, 4th year of study,

Approved by Supervisor:
Leading Research Scientist, Fedor Ratnikov

Moscow 2021

Contents

Annotation

Simulation of calorimeter responses plays a vital role in nuclear and particle physics. The existing approach to the generation of calorimeters responses is not applicable to the generation of all kinds due to its performance limitations. As part of this work, we proposed an approach to fast calorimeter response generation with time components in elementary particle detectors in CERN using generative adversarial networks. We applied Wasserstein Generative Adversarial Networks with gradient penalty and experimented with other architectures for signals generation. Our approach allows to achieve over 10,000 speed-up over the Geant4 simulation software.

Аннотация

Моделирование ответов калориметра играет жизненно важную роль в ядерной физике и физике элементарных частиц. Существующий подход к генерации откликов калориметров неприменим к генерации всех видов из-за его эксплуатационных ограничений. В рамках этой работы мы предложили подход к быстрой генерации отклика калориметра с временными компонентами в детекторах элементарных частиц в ЦЕРНЕ с использованием генеративных состязательных сетей. Мы применили генеративные состязательные сети Вассерштейна с градиентным штрафом и экспериментировали с другими архитектурами для генерации сигналов. Наш подход позволяет достичь ускорения в 10 000 раз по сравнению с используемым программным обеспечением для моделирования Geant4.

Keywords

High Energy Physics, Generative Adversarial Networks

1 Introduction

Simulation of calorimeter responses plays a vital role in nuclear and particle physics. As it is used for detectors design and comparisons between theoretical models and experimental data. Until the last few years, Monte Carlo-based methods were broadly used for particle simulation. One significant drawback of such an approach is the number of computational resources required for simulation. Due to a large amount of data during experimental runs, this approach no longer meets the performance requirements. A well-known approach to the particles interaction detailed simulation using Geant4 simulation software annually requires billions of CPU hours. This computational limitation has been successfully overcome for high granularity detectors using Generative Adversarial Networks [1]. The key limitation of the current approach [4], [5], [6] is the absence of a time component in the resulted signals, which is a crucial characteristic in nuclear and particle physics, allowing us to reconstruct events times. In this work, we attempt to apply Generative Adversarial Networks to calorimeter response generation with a time component in elementary particle detectors. Using this approach, we aim to achieve over 10,000 speed-up over the Geant4 toolkit approach. Code for our models is available on GitHub.[1]

2 Background

2.1 Generative Adversarial Network

Generative Adversarial Networks (GANs) have been widely used in different fields of deep learning for approximating very complex distributions. First introduced by I. Goodfellow in 2014 [1] the vanilla GANs are pairs of neural networks, a generator G, and a discriminator D, that are trained concurrently competing against each other. The generator network is learning to approximate data distribution given a random noise $z \in R^{|z|}$ that is sampled from a prior distribution $\mathbb{P}(z)$, which is commonly chosen to be a Gaussian distribution. The output of G - $\tilde{x}$ is expected to have similarity with

[1] https://github.com/whiteRa2bit/hep_generative_models

the real x that is drawn from the real data distribution $\mathbb{P}_r$. While the discriminator network takes either a real or generated sample and outputs single value indicating the probability of the input being a fake or real sample. The generated samples form a distribution $\mathbb{P}_g$. The training objectives of D and G can be expressed mathematically as:

$$\mathcal{L}_D^{GAN} = \max_D \mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_r}\left[\log D\left(\boldsymbol{x}\right)\right] + \mathbb{E}_{\tilde{\boldsymbol{x}}\sim\mathbb{P}_g}\left[\log\left(1 - D\left(\tilde{\boldsymbol{x}}\right)\right)\right] \tag{1}$$

$$\mathcal{L}_G^{GAN} = \min_G \mathbb{E}_{\tilde{\boldsymbol{x}}\sim\mathbb{P}_g}\left[\log\left(1 - D\left(\tilde{\boldsymbol{x}}\right)\right)\right] \tag{2}$$

From the discriminator D optimization objective it can be easily seen that it is simply a binary classifier with a maximum log likelihood objective. In [1] it was shown that in case the discriminator D is trained to optimality before the generator G next update then generator G objective is equivalent to minimizing the Jensen-Shannon divergence between $\mathbb{P}_r$ and $\mathbb{P}_g$.

2.2 Wasserstein GANs

Although GANs have shown great success in image generation, special treatements are required to overcome the issues while training the vanilla GAN, such as speed and stability of the training. Arjovsky *et al.* [2] argue that the Jensen-Shannon divergence approximation is a reason of stability and speed issues while training GANs. Instead, they propose to use Wasserstein-1 distance $W(q, p)$ also known as the Earth-Mover distance. Unlike the Jensen-Shannon divergence the Earth-Mover distance optimization is more likely to provide the gradients that are useful for updating the generator. The intuition behind the Wasserstein Distance consists in measuring the energy cost of moving and transforming a probability distribution over generated samples onto a target distribution over real samples. Formally, the optimization objective of WGAN as it was shown in [2] can be formulated as the minimax objective with the usage of Kantorovich-Rubinstein duality [16]:

$$\min_{G} \max_{D \in \mathcal{D}} \mathop{\mathbb{E}}_{\boldsymbol{x} \sim \mathbb{P}_r} [D(\boldsymbol{x})] - \mathop{\mathbb{E}}_{\tilde{\boldsymbol{x}} \sim \mathbb{P}_g} [D(\tilde{\boldsymbol{x}}))] \tag{3}$$

where $\mathcal{D}$ is the set of 1-Lipshitz functions. The idea here is that critic value approximates $K \cdot W(\mathbb{P}_r, \mathbb{P}_g)$, where K is a Lipshitz constant.

The WGAN value function results in a critic function whose gradient with respect to its input is better behaved than its GAN counterpart, making optimization of the generator easier. Additionally, WGAN has the desirable property that its value function correlates with sample quality, which is not the case for GANs.

To enforce the Lipschitz constraint on the critic, [2] propose to clip the weights of the critic to lie within a compact space $[-c, c]$. The set of functions satisfying this constraint is a subset of the k-Lipschitz functions for some k which depends on c and the critic architecture.

In 2017 significant improvements for the discussed approach were proposed in [3] by replacing weight clipping with gradient penalty. In particular, WGAN-GP penalizes the model if the gradient norm moves away from its target norm value instead of applying clipping to generator weights.

$$\lambda \mathop{\mathbb{E}}_{\tilde{x} \sim \mathbb{P}_{\tilde{x}}} \left[\left(\|\nabla_{\tilde{x}} D(\tilde{x})\|_2 - 1 \right)^2 \right] \tag{4}$$

That approach also allows to enforce the Lipschitz constraint. The proposed solution requires almost no hyperparameter tunning and is robust to the choice of generator architecture. See Figure 1 for further algorithm details.

2.3 High Energy Physics

The physics programs of all experiments based at the LHC rely heavily on detailed simulation for all aspects of event reconstruction and data analysis. Simulated particle collisions, decays, and material interactions are used to interpret the results of ongoing experiments and estimate the performance of new ones, including detector upgrades. The use of machine learning methodologies in High Energy Physics is not a new approach and a lot of work has been done in that regard. The highly stochastic and

Require: The gradient penalty coefficient λ, the number of critic iterations per generator iteration n_{critic}, the batch size m, Adam hyperparameters α, β_1, β_2.
Require: initial critic parameters w_0, initial generator parameters θ_0.
while θ has not converged **do**
  for $t = 1, ..., n_{\text{critic}}$ **do**
    for $i = 1, ..., m$ **do**
      Sample real data $\boldsymbol{x} \sim \mathbb{P}_r$, latent variable $\boldsymbol{z} \sim p(\boldsymbol{z})$, a random number $\epsilon \sim U[0, 1]$.
      $\tilde{\boldsymbol{x}} \leftarrow G_\theta(\boldsymbol{z})$
      $\hat{\boldsymbol{x}} \leftarrow \epsilon \boldsymbol{x} + (1 - \epsilon)\tilde{\boldsymbol{x}}$
      $L^{(i)} \leftarrow D_w(\tilde{\boldsymbol{x}}) - D_w(\boldsymbol{x}) + \lambda(\|\nabla_{\hat{\boldsymbol{x}}} D_w(\hat{\boldsymbol{x}})\|_2 - 1)^2$
    end for
    $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m} \sum_{i=1}^{m} L^{(i)}, w, \alpha, \beta_1, \beta_2)$
  end for
  Sample a batch of latent variables $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m} \sim p(\boldsymbol{z})$.
  $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} -D_w(G_\theta(\boldsymbol{z})), \theta, \alpha, \beta_1, \beta_2)$
end while

Figure 1: WGAN with gradient penalty algorithm

non deterministic nature of High Energy Physics combined with complex and intricate nature of interactions makes the traditional approaches very time consuming. One of the pioneering work in the quest for the elusive Higgs boson through machine learning was undertaken at Fermilab. Further work proved that deep learning replaced the tedious feature engineering procedures and provided similar performance using low level features as well as discovering novel high level features providing higher classification accuracy. Image recognition networks employing convolutional and fully connected layers to classify between jets from single hadronic particles and overlapping jets from pairs of collimated hadronic particles resulted in modest performance improvement.

Moreover, the work of machine learning for HEP is not just limited to discrimination but also regression and triggering. A recent work even discussed analogies between quantum wave function and deep convolutional arithmetic circuits (convolutional layers with linear activation and pooling layers). A new frontier is that of physics simulation: only recently machine learning has been used to replace simulation, but the interest in these techniques is rapidly spreading through the experimental community.

2.4 GANs in high energy physics

Many applications of GANs to machine learning problems have been developed since they first were proposed. These include image processing [7], natural language generation [8], time series analysis [9]. There also have been proposed various approaches for applying GANs to high energy physics [4], [5], [6]. For example, for the detector simulation and event generation, that includes calorimeter responses generation.

A detailed study on the possible applications of deep learning techniques for the generation of calorimeter responses was done in a work [6], that provided promising results in the simulation of particle showers using Generative Adversarial Networks. Further improvements were presented in [4], where a model based on Wasserstein Generative Adversarial Network with gradient penalty was used for the simulation of high granularity detectors. In both works, authors managed to achieve 100,000× speed-up over Geant4 simulation software.

3 Dataset

3.1 Data Collection

Considering datasets available for benchmarking our models, we need to note here that there is no publicly available dataset. Therefore, we generated 7000 events using Spacal Simultaion code for LHCb. An event after the preprocessing is performed presents 9 signals for corresponding detectors. Before that it consists of a list of particles with their coordinates, energies and other additional information. The code consists in a complete re-implementation of geometry and readout, to allow modular scale up of dimensions. It also implements a parametrization of optical transport to speed up simulation time. Energy in the training data was set to 1GeV.

3.2 Data preprocessing

Before using the data we additionally have to preprocess it, so we can use it for further model training. Data preprocessing is performed in three steps. The first step in the preprocessing is parsing a ROOT tree, that consists of a list of independent columns. In our case we are interested in particle 3D coordinates (x, y, z), particle total energy, detector numeric value, timestamp and event number, so we can group particles of the same event. In Figure 2 we can see the density of particles distribution along different detectors.

The second step in our preprocessing pipeline is extracting all unique events and detectors and saving their corresponding data as separate files for an easier exploration and further training usage, as the total data size exceeds 50 GB that makes it not straightforward to work with.

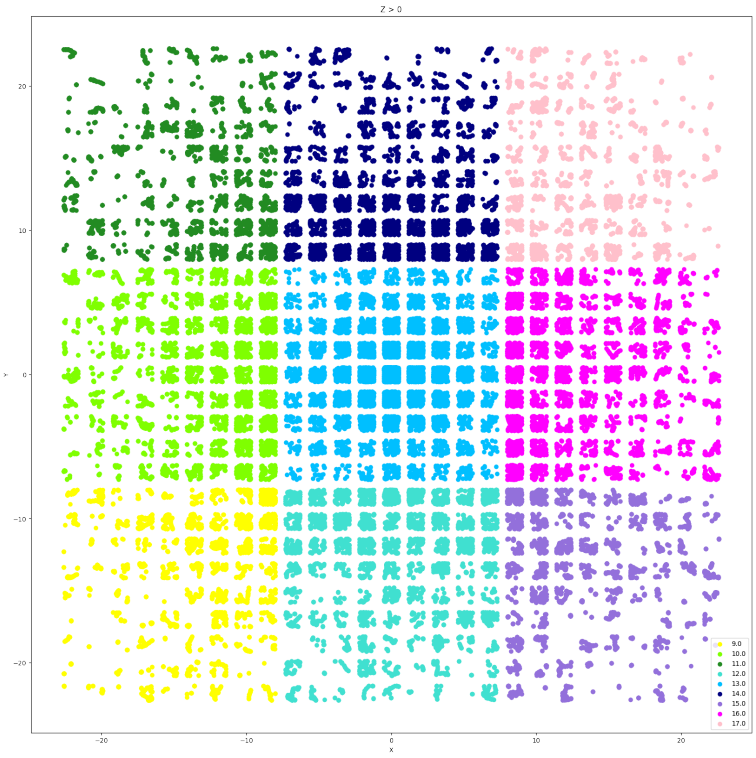


Figure 2: Density of particles distribution along detectors. Different detectors presented in different colors.

The third step of our preprocessing pipeline is building signals from our raw data, the signals that will be further used for training our models. From raw particles we build the final signal the following way:

1 We consider the particles for a given event and a detector

2 We set $n_{dim} = 1024$, then we split the particles to n_{dim} groups based on their timestamp, so that corresponding time intervals have the same length.

3 After that for each interval we calculate the total energy, let $energy_i$ be the total energy of group, then we set the i-th signal component value to $energy_i$

4 Additionally we perform another signal pre-processing step for calculating reference time. Reference time can be further used for calculating time of signals interactions that is one of the key characteristics. A baseline algorithm for its calculation takes the energy index that corresponds to a half of the amplitude level of a post-processed signal. For processing a raw signal we apply a kernel:

$$\frac{(x-y)^2}{e^{x-y}} \tag{5}$$

In Figure 3 you can see the final signal before and after the kernel was applied. In Figure 4 you can see the final data sample that we will try to generate with our models.

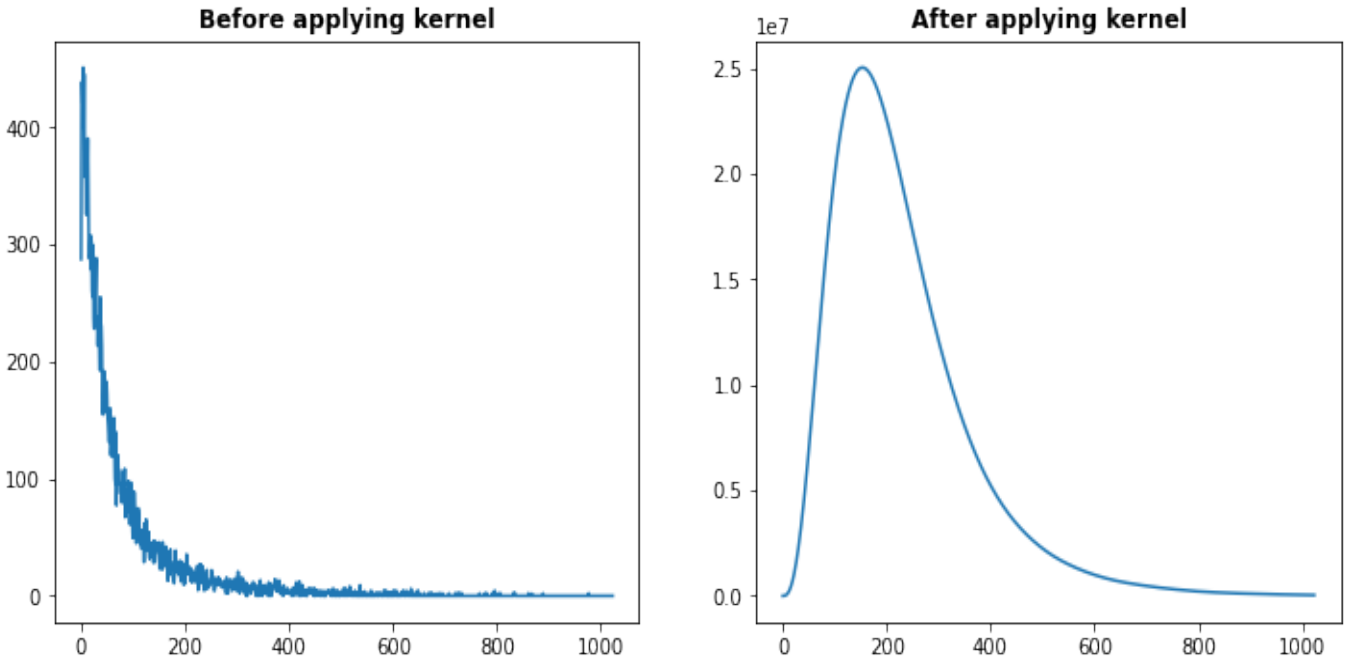


Figure 3: Kernel

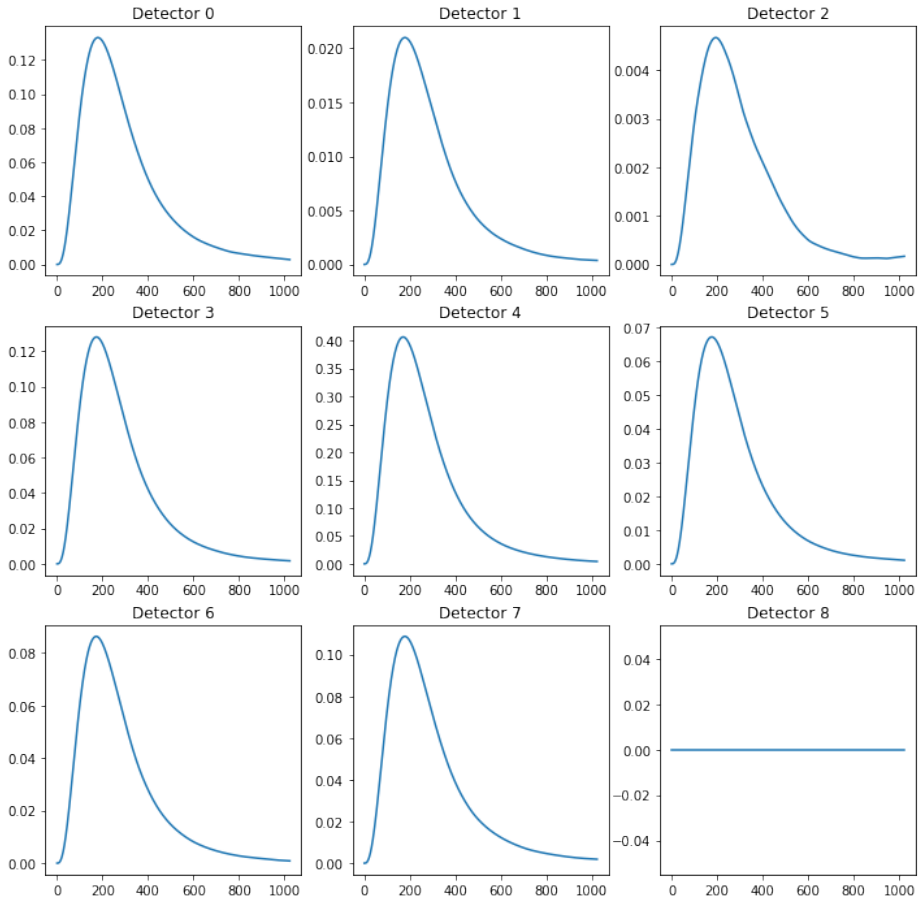


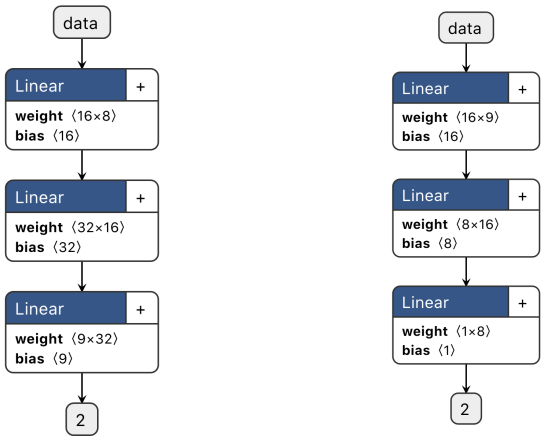
Figure 4: A data sample after the pre-processing.

4 Amplitudes Generative Model

As mentioned above, the problem of the fast generation of calorimeter responses has been considered in many works [4], [5], [6]. The key novelty of this work is that it considers the time component of the signals in comparison with previous works, where only high granularity detectors were considered.

Instead of solving an original task of generating a combination of nine signals, as it may be considered a difficult one we decided to start with a baseline approach where we predicted amplitudes of our signals and their shapes separately. When we managed to succeed with that approach we moved towards the implementation of a generic generative model that generates all signals at once.

The amplitude of a signal can be defined as the maximum value of the signal. Due to the high variation of mean amplitudes between detectors, all amplitudes were normalized using min-max normalization using the corresponding detector statistics. Below we will describe the used architectures for both generator and discriminator and



(a) Generator (b) Discriminator

Figure 5: Amplitudes Generative Model. Generator and Discriminator architectures

then dive deep into the traning details and quality assessment techniques.

4.1 Generator Architecture

As the task is relatively simple and we have to generate only nine numbers we came up with a simple architecture for the generator, consisting of three fully connected layers with ReLU [13] activation. The generator receives a noise 8-dimensional vector and produces nine detector amplitudes. A detailed architecture can be found in Figure 5.

4.2 Discriminator Architecture

Similarly to the generator, we also used an architecture containing only fully connected layers with ReLU [13] activations. It takes a 9-dimensional vector as an input and produces the probability whether the input comes from real data or was generated by a generator. A detailed architecture can be found in Figure 5.

4.3 Training

We implemented all of our models using PyTorch [11] deep learning framework. The training was performed on a single Tesla K80.

As this task consists of generating 9 numbers from random noise we used vanilla Generative Adversarial Network without additional modifications. The model was

trained with batch size 128 for 2000 epochs. For optimization we follow the approach of [2] and perform 5 gradient descent steps on discriminator , then one step on generator , using Adam [17] as a solver. The model was trained with learning rate 0.0001.

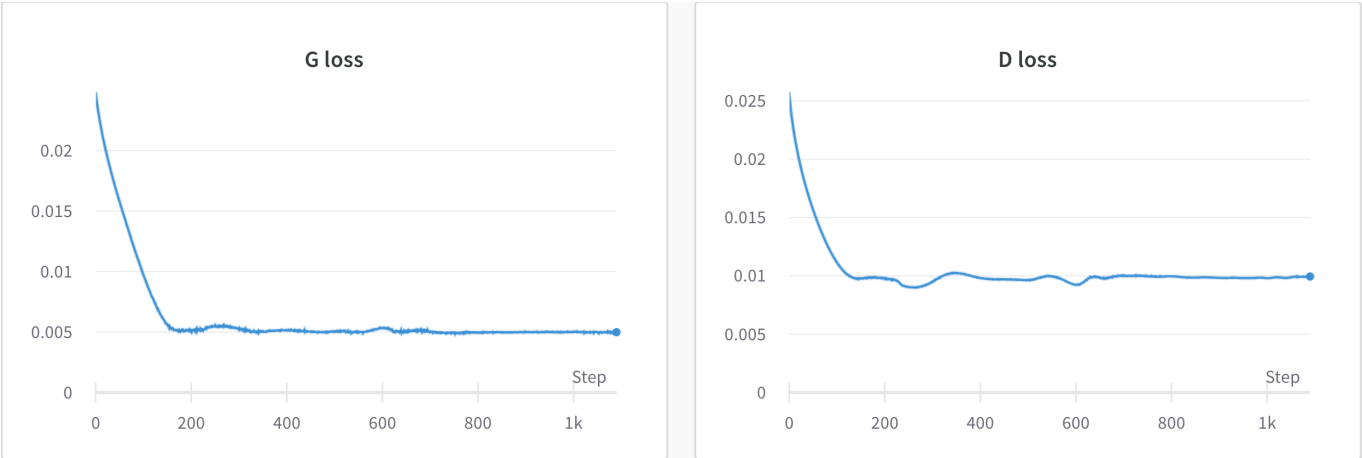


Figure 6: Amplitudes Generative Model. Losses

4.4 Quality Assessment

We compare the trained models based on Wasserstein distance between generated and real distributions for each detector. See Figure 8 for illustrations. Additionally, during training we track amplitude correlations distance, that is a Frobenius norm between of correlations matrixes difference normalized by the number of matrix elements. Formally, we define $\mathbb{M}_g \in \mathbb{R}^{9*9}$ for generated data and $\mathbb{M}_r \in \mathbb{R}^{9*9}$ for real data. We define $\mathbb{M}[i][j]$ as a Pearson correlation between amplitudes for detector i and amplitudes for detector j. Then amplitude correlations distance can be defined as $\frac{||\mathbb{M}_g - \mathbb{M}_r||_F}{81}$. The lower the value the better our model can consider the interaction between detectors.

5 Shapes Generative Model

The goal of Shapes Generative model is to generate signals without taking their amplitudes into account. To achieve this, we train a separate model for each detector $G_{detector}$ and $D_{detector}$. That is, we normalize all the signals before feeding them to $D_{detector}$ to make sure that all of them have values range in $[0, 1]$. The generator model $G_{detector}$ accepts gaussian noise as an input and generates a signal $x \in \mathbb{R}^{1024}$. The

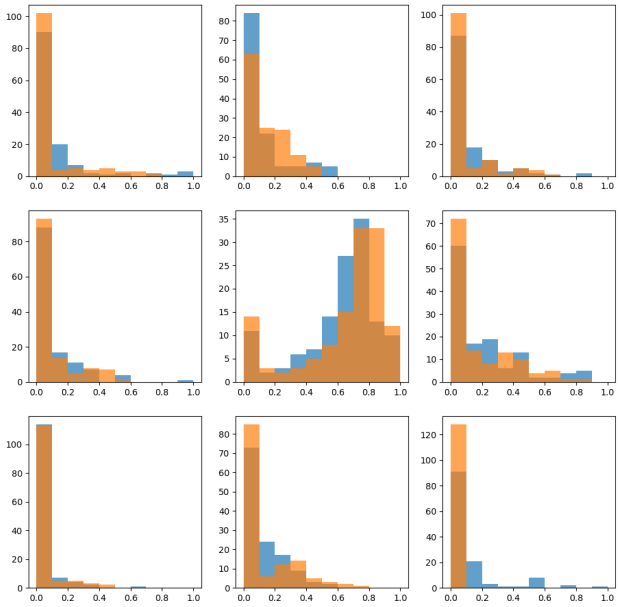


Figure 7: Amplitudes Generative Model. Real data distributions in blue, generated distribution in orange

obvious disadvantage of the proposed approach, is an independence between different detectors, signals shapes and amplitude values. That problem will be addressed by the Signals Generative model that we will discuss in the section Signals Generative Model.

Below we discuss the used architectures for $G_{detector}$ and $D_{detector}$ models and describe training details.

5.1 Generator Architecture

Generator architecture is shown in Figure 9. It contains one fully-connected layer to transform input noise to a high-dimensional vector, that is followed by 5 convolution blocks. Each convolution block consists of a convolution layer, batch normalization layer [12] and Leaky Rectified Linear Unit [14] as an activation function, with the exception of the output layer of the generator, in which we don't use batch normalization and prefer Hyperbolic Tangent Function as an activation.

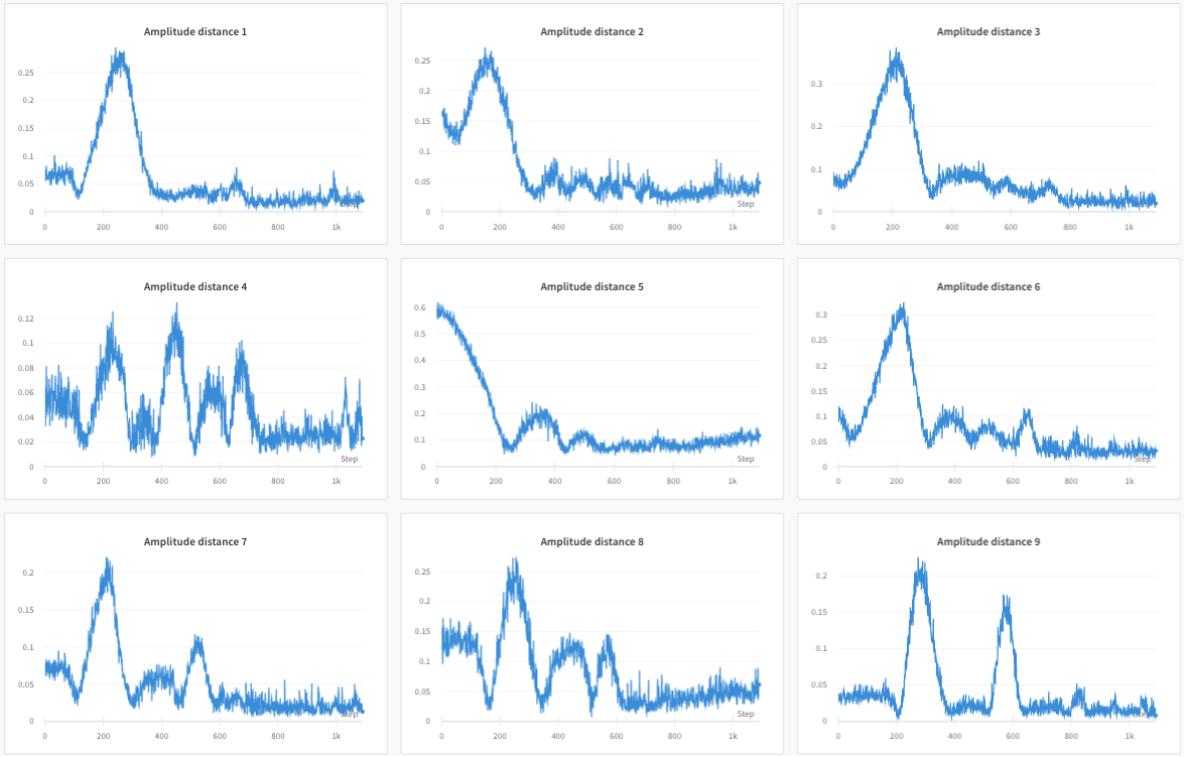


Figure 8: Amplitudes Generative Model. Amplitude distances

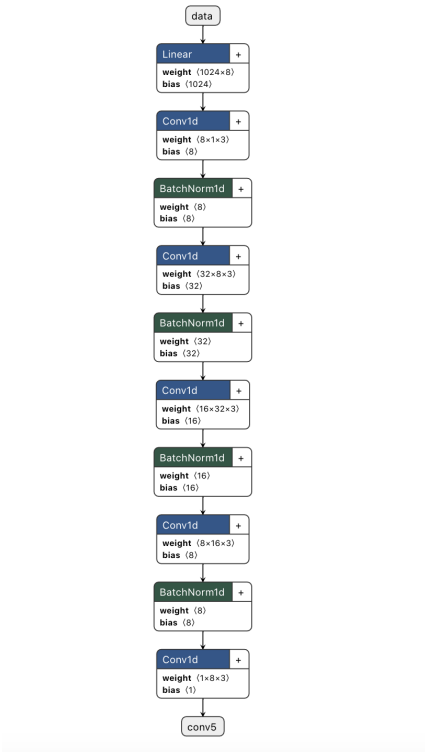
5.2 Discriminator Architecture

Discriminator architecture is shown in Figure 9. For a discriminator, we used 3 convolution blocks and a fully-connected layer to transform the final hidden-state to a single value. Instead of using batch normalization in each convolution block we used layer normalization [15] that has proven to perform better while training Wasserstein GAN [2] with gradient penalty [3]. That is, each convolution block consists of a convolution layer, layer normalization layer, average pooling and Leaky Rectified Linear Unit as an activation function.

5.3 Training

This task is considered to be a more complicated one due to the dimensionality of our training samples, as mentioned above, each signal contains 1024 energy values. As the initial approach with the vanilla Generative Adversarial model didn't succeed, we implemented the Wasserstein-GAN with gradient penalty.

The model was trained with batch size 32 for 4000 epochs. For optimization we follow the approach of [2] and perform 5 gradient descent steps on discriminator , then



(a) Generator (b) Discriminator

Figure 9: Shapes Generative Model. Generator and Discriminator architectures

one step on generator , using Adam [17] as a solver. The generator model was trained with learning rate 0.0003 and the discriminator model was trained with learning rate 0.0001. We set noise dimension to 64. For training WGAN-gp we set $\lambda = 10$ and use the parameters from [3] for Adam [17] optimizer.

5.4 Quality Assessment

During training we observe samples of real and generated data, see Figure 10 for an example. We also compare the trained models based on Wasserstein distance between reference times for generated and real data for each detector. See Figure 11 for illustrations. Additionally we track generic WGAN-GP metrics, such as generator loss, discriminator loss and gradient penalty. See Figure 12 for details. Similarly to Amplitudes Generative Model we monitor the correlations distance for reference times to evaluate how precisely the model takes the interaction between detectors into account.

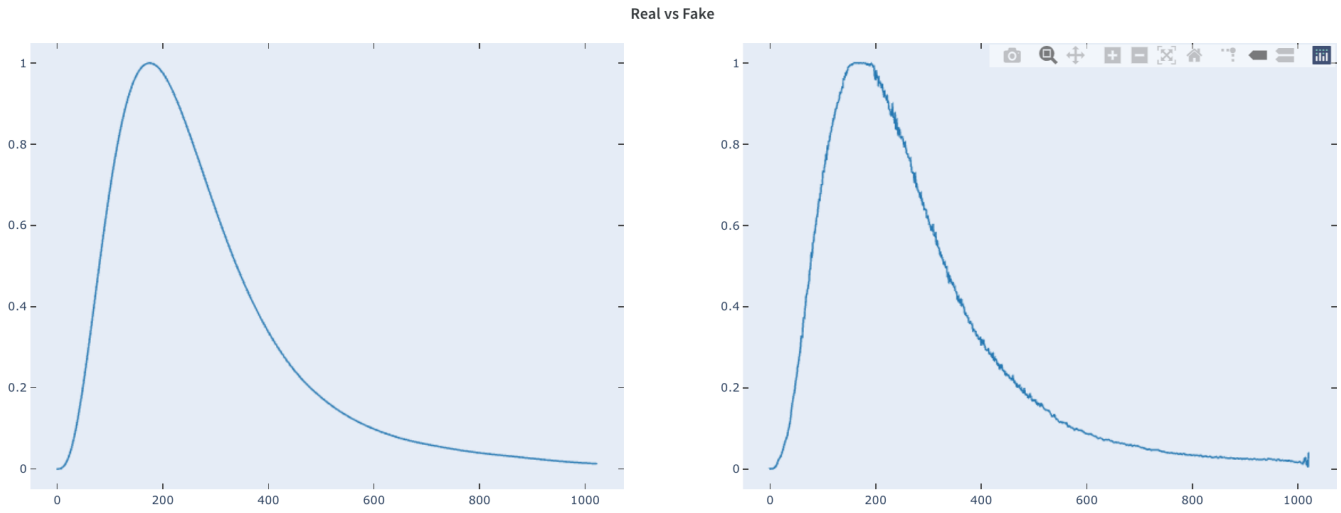
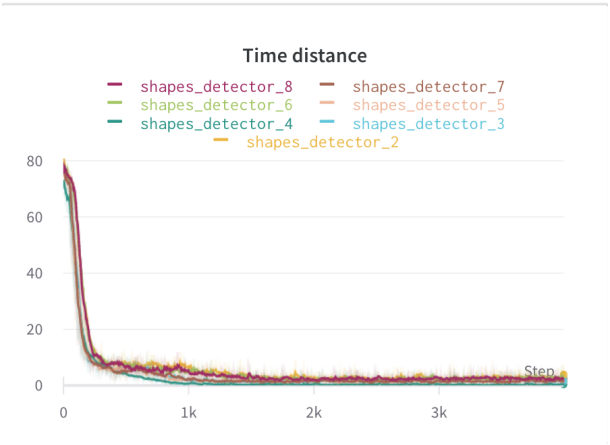
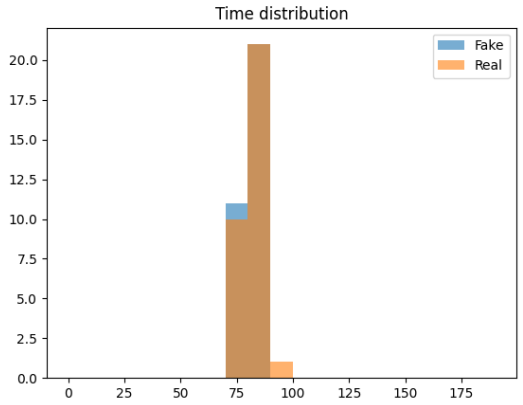


Figure 10: Shapes Generative Model. Real data on the left, generated data on the right



(a) Time distances



(b) Reference time distributions

Figure 11: Shapes Generative Model. Metrics

6 Signals Generative Model

In this section, we address the previously mentioned problem regarding the independence between different detectors, signals shapes and amplitude values. The proposed model generates detector signals $x \in \mathbb{R}^{9\times1024}$. Similarly to the shapes generative model we normalized our signals using a min-max approach across detectors in order to achieve numerical stability during training.

Below we discuss the used architectures for generator and discriminator models and describe training details.

Figure 12: Shapes Generative Model. Losses

6.1 Generator Architecture

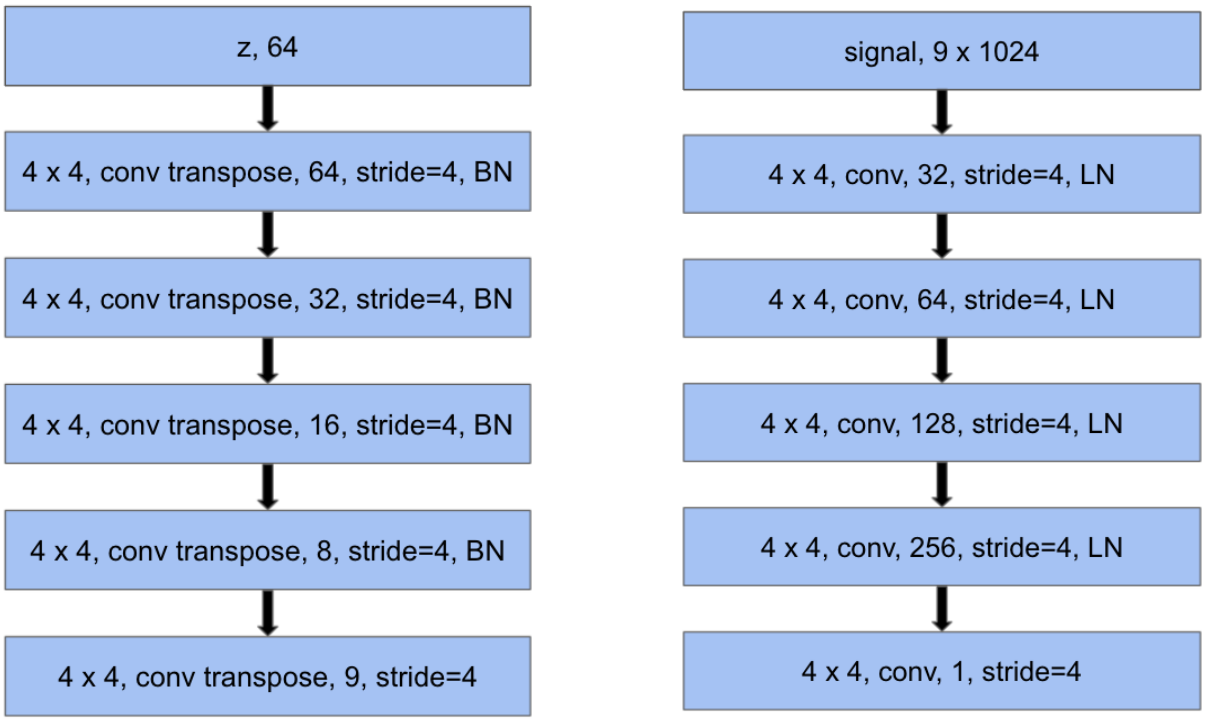
We adapt our generator architecture from the one in [10]. We use a combination of modules of the form convolution-BatchNorm-ReLU. As we are dealing with signals data we used 1D convolution layers instead of 2D convolution layers, that are usually applied to images. Notably, no fully connected or pooling layers are used. Details of the architecture are provided in Figure 13.

6.2 Discriminator Architecture

The architecture of the discriminator is identical to the one in [10]. Following DCGANs, ReLU activations and LeakyReLU activations are used for the generator and the discriminator, respectively. See Figure 13 for the details of the architecture.

6.3 Training

The model was trained with batch size 32 for 4000 epochs. For optimization we follow the approach of [2] and perform 5 gradient descent steps on discriminator , then one step on generator , using Adam [17] as a solver. The generator model was trained with learning rate 0.0003 and the discriminator model was trained with learning rate 0.0001. We set noise dimension to 64. For training WGAN-gp we set $\lambda = 20$ and use the parameters from [3] for Adam [17] optimizer.



(a) Generator (b) Discriminator

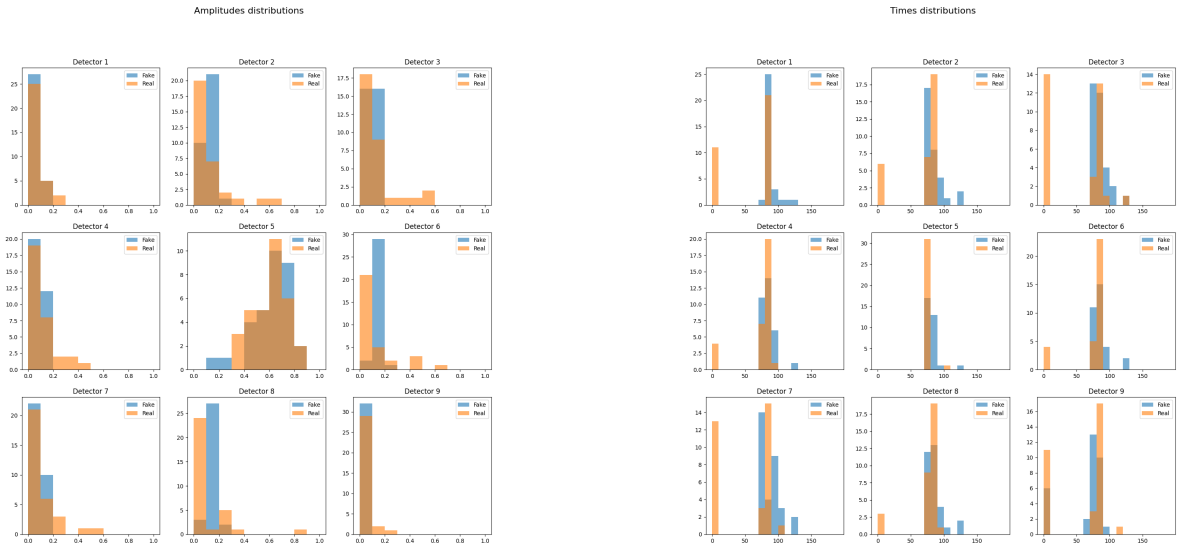
Figure 13: Signals Generative Model. Generator and Discriminator architectures

6.4 Quality Assessment

For the Signals Generative Model we combine the metrics from previous models. We track the distributions of real and amplitudes and times of signals, that can be found in Figure 14. To validate the generation results we also observe the generated signals. See Figure 15 for examples. The selection of the best model was performed based on Wasserstein distances between amplitudes and times of generated and real signals, that can be found in Figure 17. WGAN-GP training metrics can be found in Figure 16.

7 Simplified Generative Model

From the previous sections we can see that the GANs model are really sensitive to the dimensionality of input data. This motivates us to restrict our model only on generating the required physical characteristics. In this section we consider a Simplified Generative Model, that addresses the mentioned problem, instead of generating high-dimensional signals the proposed model produces only reference times and amplitudes.



(a) Amplitudes (b) Times

Figure 14: Signals Generative Model. Times and amplitudes distributions

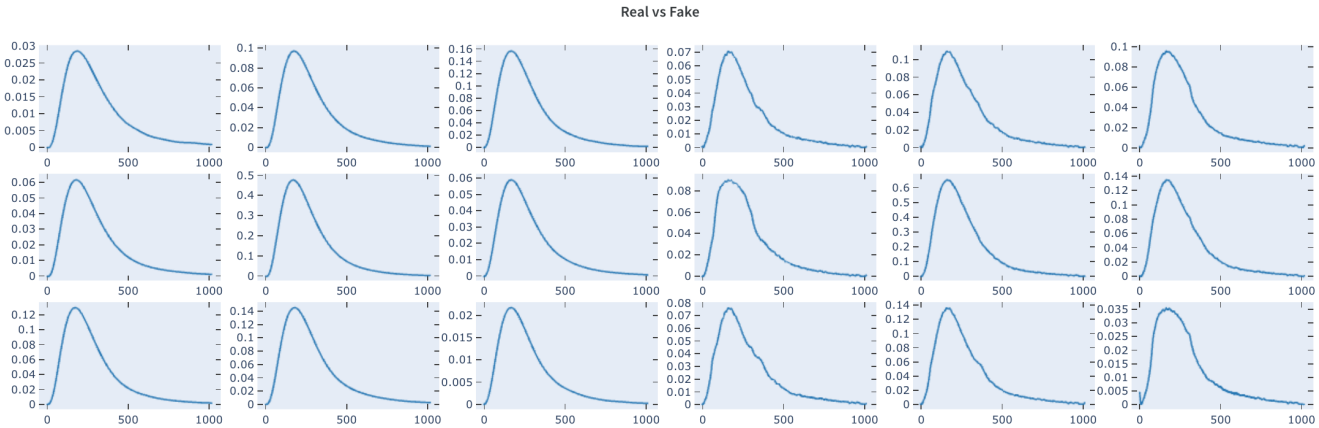


Figure 15: Signals Generative Model. Real data on the left, generated data on the right

It also solves the problem of the models we considered in the Amplitudes Generative Model and Shapes Generative Model sections.

7.1 Generator Architecture

For generator we used a similar architecture to our Amplitudes Generative Model. It also consists of fully-connected layers with ReLU as an activation function. The key difference here is that we would like amplitudes and reference times to be consistent, therefore, we used two additional learnable weight parameters, to make sure that amplitudes take zero values then and only then when reference times take zero values.

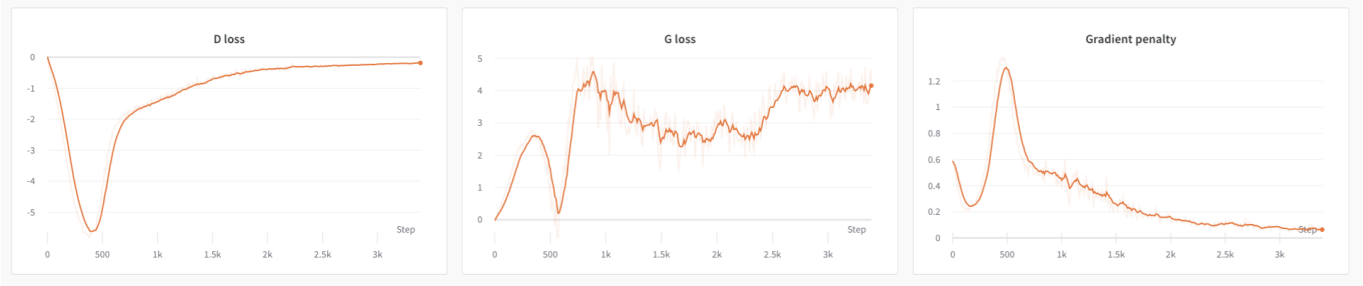


Figure 16: Signals Generative Model. Losses

(a) Amplitudes (b) Times

Figure 17: Signals Generative Model. Times and amplitudes distances

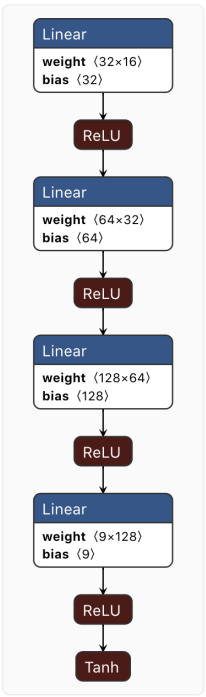
That is, the generator returns amplitudes generated values that are further transformed into time values via multiplication on two learnable weight parameters with Hyperbolic Tangent Function activation between. See Figure 18 for additional details.

7.2 Discriminator Architecture

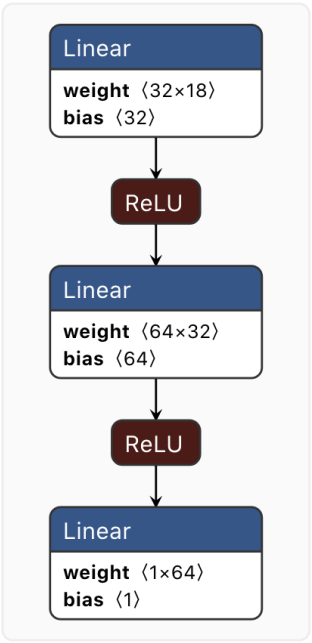
For discriminator we also used a similar architecture to our Amplitudes Generative Model. See Figure 18 for additional details.

7.3 Training

The model was trained with batch size 128 for 2000 epochs. For optimization we follow the approach of [2] and perform 5 gradient descent steps on discriminator , then one step on generator , using Adam [17] as a solver. The generator and discriminator models were trained with gradual warmup learning rate scheduling, initial learning rate was set to 0.001, for further details see . We set noise dimension to 16. For training WGAN-gp we set $\lambda = 10$ and use the params from [3] for Adam [17] optimizer.

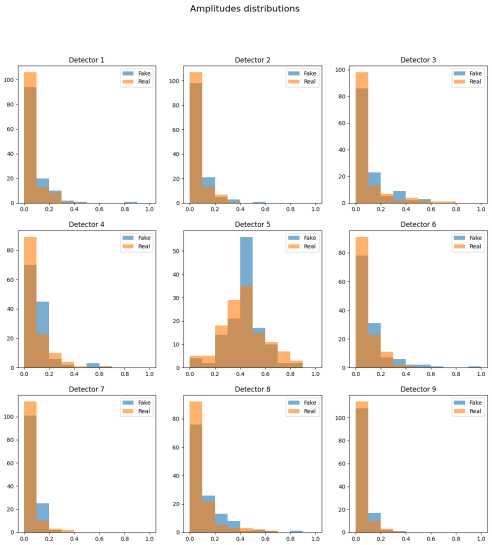


(a) Generator

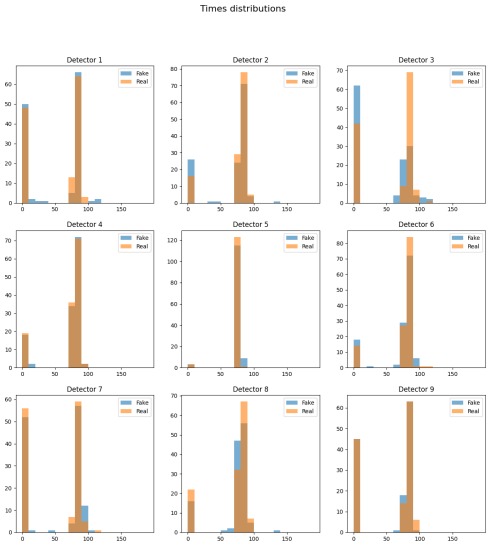


(b) Discriminator

Figure 18: Simplified Generative Model. Generator and Discriminator architectures



(a) Amplitudes

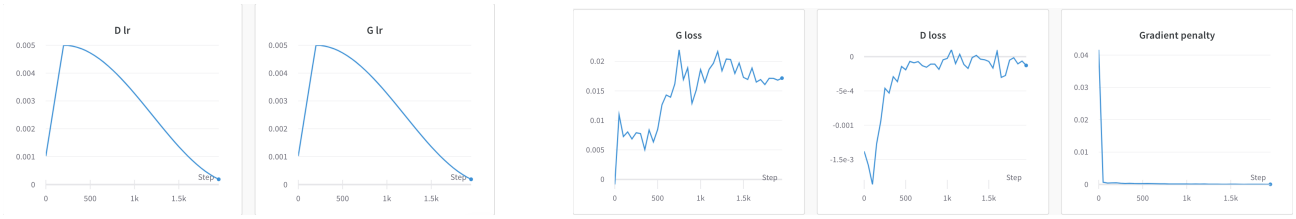


(b) Times

Figure 19: Simplified Generative Model. Times and amplitudes distributions

(a) Amplitudes (b) Times

Figure 20: Simplified Generative Model. Times and amplitudes distances



(a) Learning Rate Schedule (b) Losses

Figure 21: Simplified Generative Model. Learning rate schdule and losses.

7.4 Quality Assessment

We track the distributions of real and amplitudes and times of signals, that can be found in Figure 19. The selection of the best model was performed based on Wasserstein distances between generated and real amplitudes and times, that can be found in Figure 20. Additionally, we log the correlations distances for times and amplitudes between detectors. See Figure 22. Finally, to validate the process of training and to detect unsuccessful models early we track generic WGAN-GP metrics, such as generator loss, discriminator loss and gradient penalty. See Figure 21 for details.

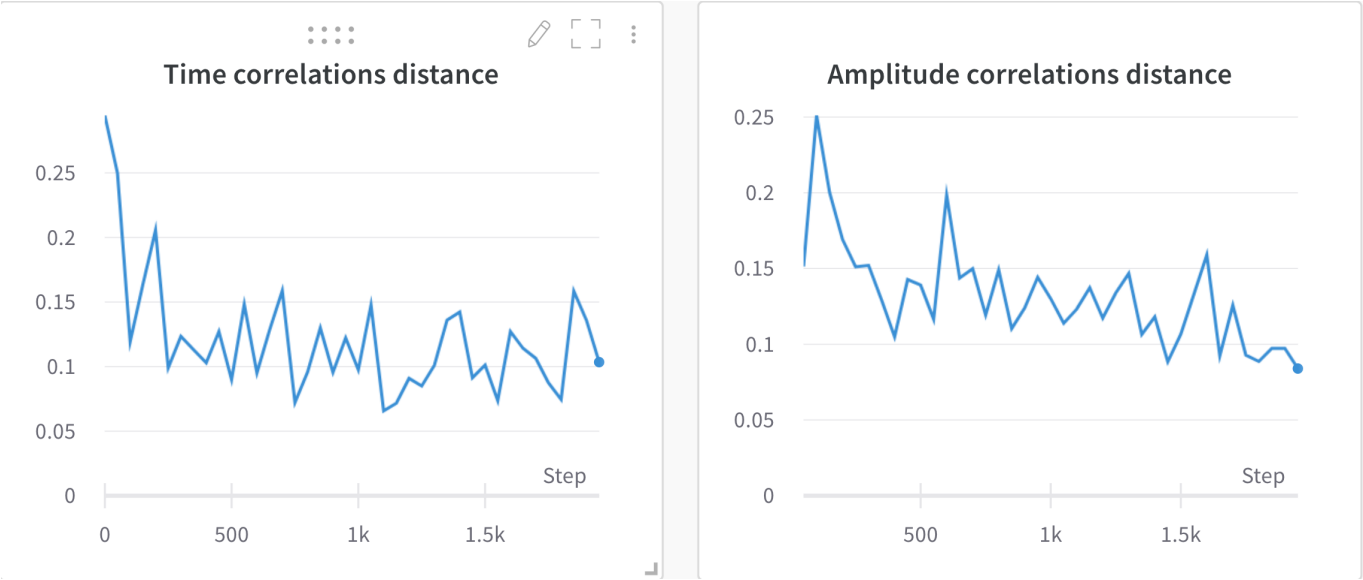


Figure 22: Simplified Generative Model. Correlations Distances

8 Results

In this section we present the results of the models comparison and discuss the benefits and drawbacks of each trained model.

We compare the trained models based on four metrics, that represent how accurately a model can recover an individual detector statistics and how well it can capture the interaction between detectors. For first, we average the amplitudes and times distances for all detectors correspondingly. These two metrics can be found as columns of Table 1 "Amplitudes dist." and "Times dist." correspondingly. For measuring how well the models can generate dependencies between detectors statistics we also consider "Times correlation dist." and "Amplitudes correlation dist." that were defined in previous sections. Overall, we can see that Simplified Generative Model outperforms two other models for all metrics except times distance, where the baseline model performs roughly twice better.

Table 1: Models Comparison

Model	Times dist.	Amplitudes dist.	Times correlation dist.	Amplitudes correlation dist.
Shapes + Amplitudes	**1.877**	0.038	0.851	0.924
Signals	19.205	0.064	0.481	0.746
Simplified	3.411	**0.015**	**0.104**	**0.084**

Despite the fact, that Signals Generative Model performs worse than the base-

line model based on individual detectors statistics it is also can be useful as it can generate realistic signals shapes, from which additional information can be extracted. Furthermore, it captures the dependencies between detectors statistics significantly better compared to the baseline model.

9 Conclusion

In this work, We examined methods of implementing generative models for physical signals with time components. Using our methods, we allow replacing the GEANT4 toolkit for calorimeters responses generation while achieving over 10,000x speed-up and still simulating realistic signals. We believe that the outcomes of this study will simplify the process of further data generation for research in high-energy physics. This paper has argued that the application of generative models for calorimeters responses generation has not been fully studied yet and there is a space for improvements to allow generating new data with time components 10,000 faster.

10 References

1 Goodfellow, Ian J., Pouget-Abadie, Jean, Mirza, Mehdi, Xu, Bing, WardeFarley, David, Ozair, Sherjil, Courville, Aaron C., and Bengio, Yoshua. Generative adversarial nets. NIPS, 2014.

2 Martin Arjovsky, Soumith Chintala, and Leon Bottou. Wasserstein generative adversarial networks. In ICML, pp. 214–223, 2017

3 Gulrajani, Ishaan, Ahmed, Faruk, Arjovsky, Martin, Dumoulin, Vincent, and Courville, Aaron C. Improved training of wasserstein gans. In Advances in Neural Information Processing Systems, pp. 5769–5779, 2017.

4 V. Chekalina, E. Orlova, F. Ratnikov, D. Ulyanov, A. Ustyuzhanin, and E. Zakharov, Generative models for fast calorimeter simulation: The LHCb case, EPJ Web Conf. 214, 02034 (2019).

5 Fedor Ratnikov. Generative Adversarial Networks for LHCb Fast Simulation. LHCb-TALK-2019-403, CHEP2019, Adelaide, November 2019. http://cds.cern.ch/rec

6 M. Paganini, L. de Oliveira, B. Nachman, CaloGAN : Simulating 3D high energy particle showers in multilayer electromagnetic calorimeters with generative adversarial networks, Phys. Rev. D97 (1) (2018) 014021.

7 Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. In ICLR, 2019.

8 Weili Nie, Nina Narodytska, and Ankit Patel. RelGAN: Relational generative adversarial networks for text generation. In International Conference on Learning Representations, 2019.

9 Dan Li, Dacheng Chen, Jonathan Goh, and See-kiong Ng. Anomaly detection with generative adversarial networks for multivariate time series.

10 A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434, 2015.

11 PyTorch. http://pytorch.org.

12 S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In ICML, 2015.

13 Nair, Vinod and Hinton, Geoffrey E. Rectified linear units improve restricted Boltzmann machines. In ICML, pp. 807–814, 2010.

14 A. L. Maas, A. Y. Hannun, and A. Y. Ng, in in ICML Workshop on Deep Learning for Audio, Speech and Language Processing (2013).

15 Jimmy Ba, Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer normalization. CoRR abs/1607.06450.

16 C. Villani. Optimal transport: old and new, volume 338. Springer Science Business Media, 2008.

17 D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. CoRR, abs/1412.6980, 2014.