EVENT SIMULATION USING MACHINE LEARNING AT THE ATLAS DETECTOR

JERNEJ DEBEVC

Fakulteta za matematiko in fiziko
Univerza v Ljubljani

Most modern high energy physics experiments rely on a substantial amount of simulated collision events in an analysis to achieve acceptable statistical accuracy. Event generation using current Monte Carlo methods is highly CPU intensive and time consuming. A faster alternative presently being studied is utilizing machine learning algorithms, which proved to be orders of magnitude faster than traditional methods. In this article, I present the process of generating simulated events, describe generative models and demonstrate how they could be used to accelerate event production.

SIMULACIJA DOGODKOV Z UPORABO STROJNEGA UČENJA NA DETEKTORJU ATLAS

Večina modernih visokoenergijskih fizikalnih eksperimentov pri analizi potrebuje veliko količino simuliranih dogodkov, da dosežejo sprejemljivo statistično natančnost. Generacija dogodkov z Monte Carlo metodami, ki je trenutno v uporabi, je računsko zelo zahteven in časovno zamuden postopek. Možna alternativa, ki se trenutno proučuje, je uporaba algoritmov strojnega učenja, za katere se izkaže, da so rede velikosti hitrejši od tradicionalnih metod. Članek predstavlja postopek generiranja simuliranih dogodkov, osnove generativnih modelov ter možen način uporabe le-teh za pospešitev generacije dogodkov.

1. Introduction

One of the key steps in the analysis in high energy particle physics experiments is the calculation and generation of numerous simulated interactions and detector responses of events being studied at an experiment. Such generated events are crucial as they provide a means of comparing theoretical predictions of interaction processes with actual measured data from the experiment.

The technique used to produce simulated events, which has seen extensive use in the field over the past several decades, is Monte Carlo (MC) simulation. The main challenge of Monte Carlo simulation is the fact that it is computationally expensive, requiring hundreds of thousands of CPU cores to run continuously to satisfy the requirements of the LHC experiments. Therefore, with the ever increasing need for larger sets of simulated events, generating enough in a sufficiently short time period is becoming an overwhelming task which calls for new approaches that would solve the challenge.

Firstly, the article presents the simulation procedure on the example of the ATLAS experiment and how the simulation may be accelerated using machine learning techniques. Next, one of these techniques is described in detail and an example is given showing how it could be used.

2. Simulation of events at the ATLAS experiment

2.1 Current simulation production chain

The ATLAS experiment [1] (cf. Fig. 1) is a general purpose particle physics detector situated at the Large Hadron Collider (LHC) in CERN, Geneva. It has the task of measuring highly energetic proton-proton or heavy ion collisions produced by the LHC with the current maximum center of mass energy of 13 TeV. It consists of multiple subsystems, each measuring different quantities and particles. Closest to the interaction point are the pixel detectors, which measure the trajectory of charged particles. It is followed by a series of electromagnetic and hadronic calorimeters, measuring particle energies. The outermost part of the detector is the muon spectrometer, measuring the momentum of muons. Superconducting magnets in the detector produce strong magnetic fields, which are used to bend charged particles for momentum measurement.

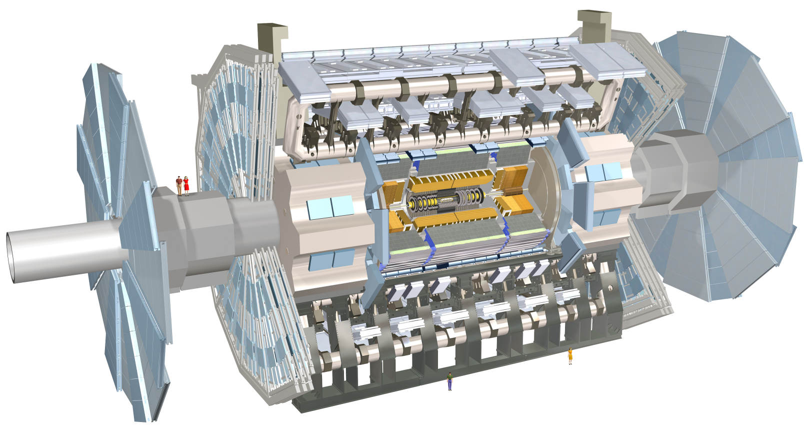


Figure 1. The ATLAS detector. A part of the detector is missing in order for the inner structure of the detector to be visible. (Source: ATLAS Experiment © 2008 CERN)

During a typical analysis, simulated events have to be generated for comparison with the measured events (example on Fig. 2) in order to identify already known processes (called background) as well as potential new physics processes (called signal). For this reason the simulated events should be produced in the same format as the true detector response in order to utilize the same reconstruction algorithms on both kinds of data, thereby eliminating potential biases which would arise from using different software tools and procedures [2].

The entire simulation infrastructure for event production consists of various components (stages) [2]. The simulation starts with event generation. In this phase a single interaction of protons or heavy ions at a time is simulated using theoretical predictions with the end result being particles entering the detector volume, each having its own four-momentum. Some particles which are created in the interaction process are very unstable and decay rapidly. Particles with a proper lifetime $c\tau > 10\,\mathrm{mm}$ are considered stable. Their decay is left to the next phase of the simulation: simulation of the detector response. Particles with a proper lifetime $c\tau < 10\,\mathrm{mm}$ are considered not to interact with the detector and are decayed by the event generator [2]. Examples of event generators being used at ATLAS are PYTHIA [4], HERWIG [5] and Sherpa [6].

As mentioned above, the next phase is the detector simulation, the most computationally expensive part of the process. During the detector simulation, each particle from the event generator is propagated through the entire detector by a dedicated software framework named GEANT4 [7]. It simulates interactions of the particle with the material of the detector in great detail, recording interactions, particle creation and annihilation, energy and charge deposits in active detector volumes etc. In order for this to function properly, GEANT4 has to be aware of the entire detector geometry. Therefore, an exact 3D model of the detector, which includes every component, has to be passed to the simulation software.

Complicated geometry within the detector has a big impact on computation time. The major contributors in this regard are the electromagnetic calorimeters. Due to their complex accordion shape, shown in Figure 3, around 80% of the full detector simulation time is devoted to calorimeter simulation alone [2].

The next phase of the simulation is digitization, where the detector response is converted to voltage and current pulses that would be measured and recorded, if the simulated event occured

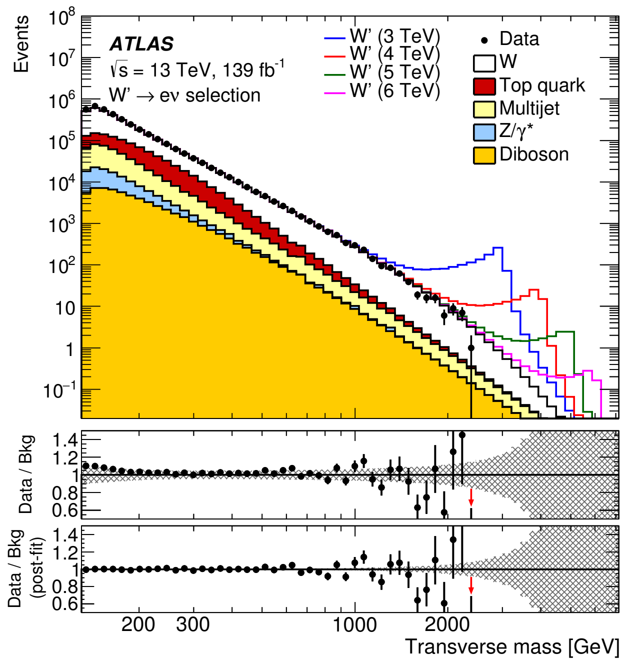


Figure 2. An example of comparison between the measured events (labeled Data) and simulated events at the analysis of ATLAS data. (Source: [3])

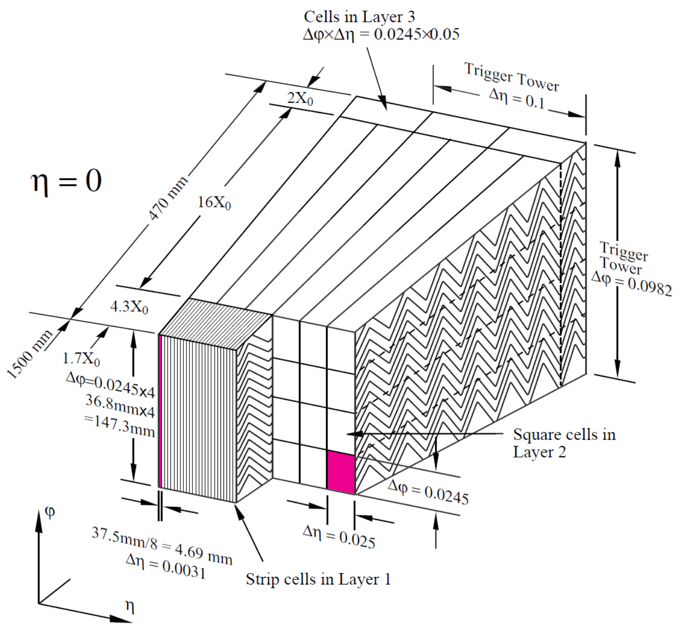


Figure 3. A section of the ATLAS calorimetry, showing the complex accordion structures, the most computationally expensive component in the detector simulation. (Source: [1])

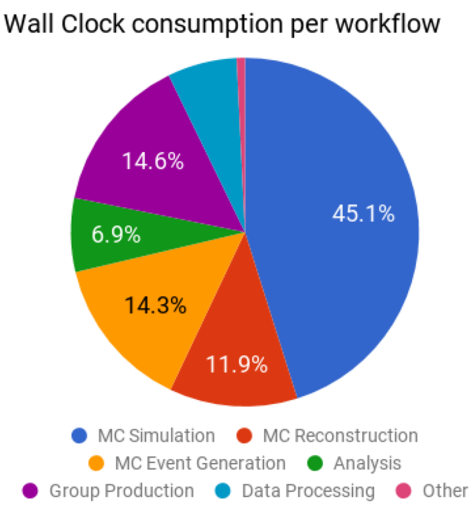


Figure 4. Computing activities are shown with their respective contributions to the CPU consumption. A large fraction of this time is devoted to MC simulation. Values are shown for the year 2017. (Source: [9])

in the detector. At the end of the digitization phase, the simulated events are in the same format as the measured events and can be fed into reconstruction and later analysis algorithms for further processing.

2.2 Generation speed considerations

As seen in the previous section, the simulation production chain is quite long, with each step being relatively CPU intensive. A single event simulation therefore takes on the order of minutes to complete. This fact becomes especially significant if we consider that a typical analysis requires $\mathcal{O}(10^9)$ simulated events in order to achieve sufficient statistical accuracy [2]. Producing an adequately sized volume of simulated events is thus computationally highly demanding. In order to have sufficient computing power to produce simulated events on acceptable time scales, a massive world-wide supercomputer architecture called the "World-wide LHC Computing Grid" [8] is being utilized for computation. It is a vast collection of computing infrastructure spread across about 170 sites in 40 countries, cumulatively contributing almost one million CPU cores and 500 PB of data storage. Figure 4 shows the fraction of time spent on every subtask. As mentioned, the largest portion of the computing time is spent on detector simulation, contributing to almost half the computing time. Event generation and reconstruction have a smaller but still significant impact.

Considering the high time complexity of the entire simulation production chain, optimization is a priority in simulation software development. As the LHC is being constantly upgraded to achieve higher energies and collision frequencies, the amount of data taken from collisions by the ATLAS detector has an upwards trend as well. Such a trend must be followed by the amount of simulated events, again to achieve sufficient statistics in order to optimize the measurement uncertainties. The increase in demand of more simulated data cannot be expected to be matched by the increase of computing power, as such upgrades require large financial investments. Consequently, new alternative solutions and methods have to be found to improve the rate of event production.

One of the main alternative options being considered as a solution is using machine learning algorithms.

3. Generative modeling techniques

Machine learning (ML) is a field which has seen a significant increase in use over the past couple of years. It is the study of algorithms and models which can perform a specific desired task without being explicitly programmed to do so, i.e. they can learn only from the examples provided. The structures being used for this task are called neural networks that contain many parameters, which can be adjusted individually. Training the neural network consists of feeding it lots of training data and adjusting these parameters to minimize a defined loss function $\mathcal{L}$ accordingly. The choice of the loss function is problem dependent and ultimately defines what the neural network learns.

Machine learning algorithms are mostly used for classification purposes, which requires the training data to be labeled. For this particular purpose, the ML algorithms are combined in an original way in order to, given a training set of data, generate new samples similar to those in the training set by learning its properties from examples. Networks that can perform this task are called generative models. Note that - unlike with the usual classification problems - the training data does not require to be labeled or classified. The model only requires to see enough samples to produce similar new ones.

The two variants of generative models that have proven to perform well are Generative Adversarial Networks (GANs) [10] and Variational Autoencoders (VAEs) [11]. We will briefly mention GANs and describe VAEs in more detail.

3.1 Generative adversarial networks

The main idea of GANs (schematically shown on Fig. 5a) is to simultaneously train two individual neural networks [12]. One of them is the generator, which, given random noise as input, produces new samples that should follow a specific distribution provided by the training data. The other is called the discriminator. It has the task of determining whether the sample it is given is real (i.e. from the training data) or fake (i.e. produced by the generator). Only the discriminator ever gets to see the training data. The generator trains only by learning to fool the discriminator. The generator is optimally trained when the discriminator cannot distinguish between real and fake samples.

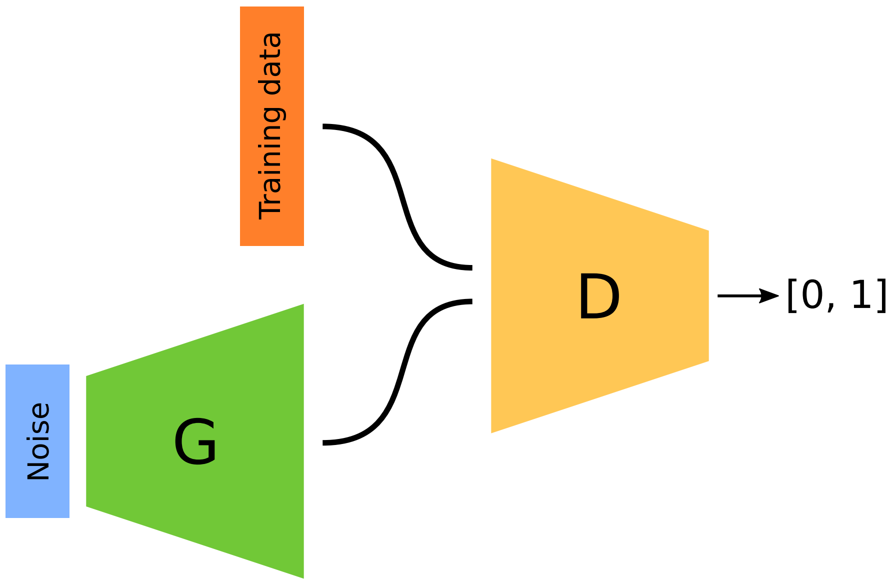
3.2 Variational autoencoders

To describe variational autoencoders in more detail we have to start by going back to the basics of what we are trying to achieve. Given a set of training data we want to extract some underlying defining features of the data and represent the samples using these features, which are usually called latent variables.

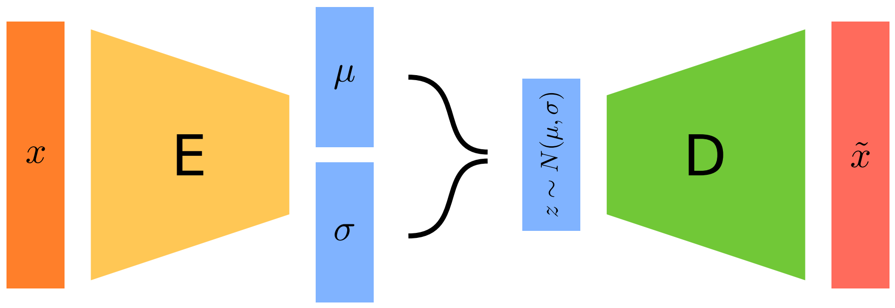
We can extract these features by passing the data set through a neural network shown on Figure 5b. Firstly, we take a sample x from the training data and feed it through the first neural network, called the encoder. Since we want to have the latent variables normally distributed and uncorrelated, the encoder outputs means μ and standard deviations σ of a Gaussian distribution. Usually, we want the dimensionality of the latent space smaller than that of the original data (e.g. we want to describe a handwritten digit in only a couple of parameters). This of course implies a loss of information contained within the training data.

Secondly, an input sample z for the second neural network, the decoder, is generated from the distribution given by the encoder. The decoder tries to produce the most accurate copy $\tilde{x}$ of the input x, i.e. given low-dimensional parameter values in the latent space, retrieve the most amount of information about the initial input x.

Let us consider the structure of the loss function $\mathcal{L}$ to achieve the desired behaviour. As mentioned above, the desired output of the decoder should be as close to the initial input as possible.



(a) Schematic representation of a GAN. Random noise is the input to the generator (G). Both the generator's output and the training data are fed into the discriminator (D), which gives a prediction whether the input was real or fake.



(b) Schematic representation of a VAE. The encoder (E) takes a training sample x and outputs a mean μ and standard deviation σ of a Gaussian distribution. A random z is generated which follows this distribution and is the input to the decoder (D), which produces a sample $\tilde{x}$ that is as close to x as possible (ideally $\tilde{x} = x$).

Figure 5. Schematical figures of a GAN and VAE, two popular generative models. Each trapezoid represents a neural network. The height of the shapes represents their size.

This is easily described in the loss function as

$$\mathcal{L}_{\mathrm{MSE}} = |x - \tilde{x}|^2 \,. \tag{1}$$

Although the right side of the equation is not exactly the mean squared error (MSE), but rather off by a constant (the dimension of x), this is not important in this case, as we are only searching for the minimum of $\mathcal{L}$.

The second term in the loss function comes from constraints in the latent space. We assume the latent variables are normally distributed and prefer them to have the mean at 0 and standard deviation 1. We can describe this by comparing the distributions using Kullback-Leibler (KL) divergence. The KL divergence between a probability distribution p and a reference probability distribution q is defined as

$$D_{\mathrm{KL}} = \int p(x) \ln \frac{p(x)}{q(x)} \, \mathrm{d}x \,. \tag{2}$$

Evaluating the KL divergence for two Gaussian distributions, one having mean 0 and standard deviation 1, gives

$$D_{\mathrm{KL}} = -\frac{1}{2} \left(1 + 2 \ln \sigma - \mu^2 - \sigma^2\right) \,. \tag{3}$$

Assuming we are considering uncorrelated variables, we can add such a term for each latent variable directly into the loss function as

$$\mathcal{L}_{\mathrm{DKL}} = -\sum_i \frac{1}{2} \left(1 + 2 \ln \sigma - \mu^2 - \sigma^2\right) , \tag{4}$$

where i goes over all latent variables, to have the latent space distributions close to the standard Gaussian. The loss function therefore has the form

$$\mathcal{L} = (1 - \beta)\, \mathcal{L}_{\mathrm{MSE}} + \beta\, \mathcal{L}_{\mathrm{DKL}} \,. \tag{5}$$

We add the parameter β as a weight to both terms [13].

With this we can now train the VAE on our training data set. The question that now remains is how to produce new samples with the trained network. We have seen that we can represent the main features of the data in a lower-dimensional latent space, with the latent variables following an uncorrelated standard normal distribution. Therefore, we can only take the decoder of the VAE, generate random numbers following independent Gaussian distributions in the latent space, which is possible due to the latent space variables being uncorrelated, and give them as input to the trained decoder. The decoder outputs new samples, which are similar to those in the training set.

4. Event generation using generative models

We can now use this architecture to generate new simulated events describing particle collisions in high energy physics experiments. We simply take a set of simulated events as the training data and train the generative model to produce similar events. Note that this implies that simulated events still have to be produced the classical way as described in Sec. 2. The advantage comes from the smaller amount of events that have to be produced in this way, as the majority is produced by the generative model.

The speed increase of generating events with neural networks is tremendous, usually being $\mathcal{O}\big(10^8\big)$ times faster than the classical method. There is a drawback to using machine learning however. As mentioned in Sec. 3, the output sample from a generative model is never exactly the same as the input, therefore, an inaccuracy is introduced. In terms of simulated events, this means that an event can become unphysical. Consequently, a compromise has to be made between speed and accuracy. Considering that generated events have to be very close to real simulated events, this problem has to be studied closely when integrating generative modeling into the simulation production chain for use in a real analysis.

There are also different options when integrating generative modeling into the production chain. One can choose to skip the entire MC production chain and immediately produce final events used in the analysis. The other option is to replace ML algorithms only for the most time consuming part of the production chain. For example, we mentioned in Sec. 2 that the calorimeter simulation is very expensive and could therefore be simulated using generative models. All other parts of the simulation would still be done using classical MC methods.

5. An example of the concept

To show how we can use generative modeling for event production, we can present a simple example. For this we take $2 \cdot 10^5$ events from the Higgs data set [14] available at the UCI Machine Learning Repository [15]. The Higgs data set has 28 features which are produced during a scattering interaction. For the simplicity of this example, only 6 of those are used. The distributions of these features are shown in Fig. 6a in blue. Our goal now is to produce new events which follow the same distributions as the given data.

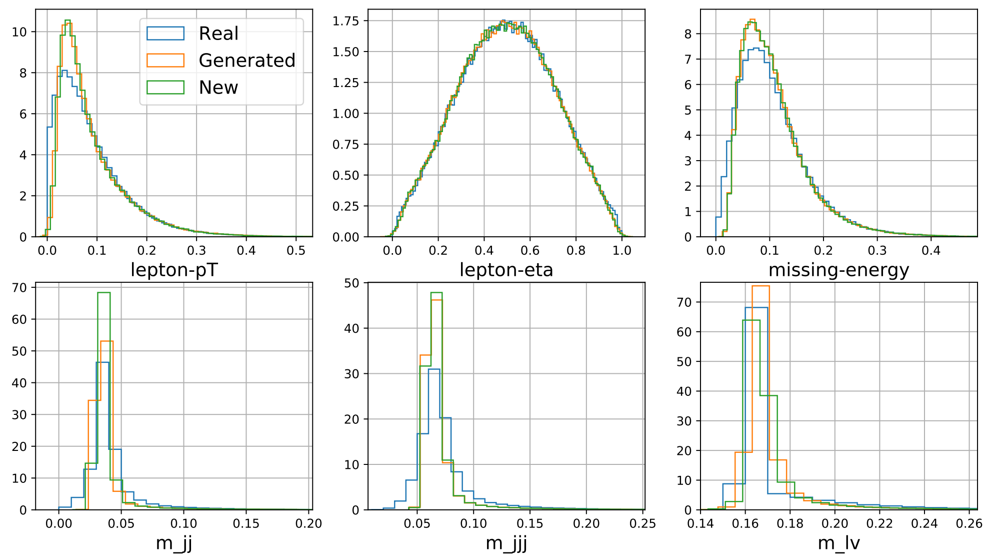
Using `Tensorflow` [16], a popular Python library for machine learning, we construct a small VAE, the encoder and decoder both consisting of two hidden layers with 50 neurons each and a latent space of the same dimension as the input. We take (5) as our loss function with $\beta = 0.001$ and train the network on the Higgs data set samples.

With the network trained we can now visualize the latent space. Fig. 6b shows the latent space distributions of the data in blue. All are very close to standard normal distributions which follows from the fact that we requested this in the $\mathcal{L}_{\mathrm{DKL}}$ term of the loss function. The minor deviations result from the loss function having a second term, requiring the output samples to be good approximations of the input data as well. By feeding the latent samples to the decoder we can produce distributions in the feature space shown in Fig. 6a in orange. The distributions are similar to the input ones, but still have slight differences. A good choice of all training parameters and a larger neural network would achieve better accuracy.

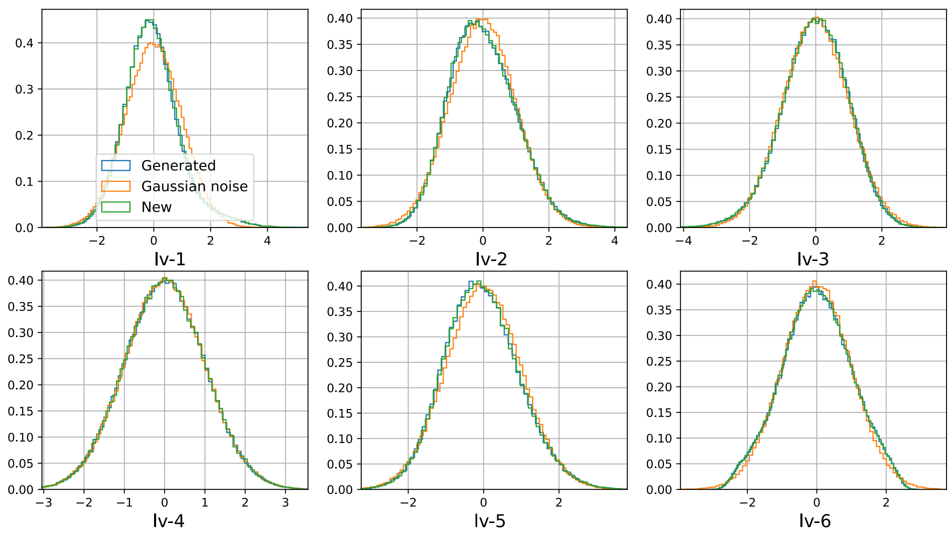
To generate new events we produce new samples in the latent space and feed them through the decoder. Both distributions are shown in Fig. 6. Firstly notice that the distributions in the latent space are more similar to the generated ones and do not follow the standard normal distributions perfectly. The reason for this is that a density information buffer [17] was used when generating new latent space samples. This method takes into account small but important deviations of the latent distributions from the standard normal distributions and therefore produces more accurate results. When we take a look at the feature distributions, the new events have a similar distribution as the generated events and are therefore slightly different from our initial distribution. However, considering that these distributions were produced by a very small and simple neural network, these results can be considered promising. Also, during training of the network many internal parameters can be fine tuned to produce better results. There is also an option of providing more training data and enlarging the neural network to get a more accurately trained model.

6. Conclusion

Machine learning is a promising faster alternative for production of simulated collision events for high energy particle physics experiments. As shown in the example, a simple neural network is capable of learning the distributions of the features to a good degree of accuracy. For a real experiment,



(a) Comparison of distributions of the chosen features in the Higgs data set. Distributions are plotted for the values in the data set (real), the values we obtain after training as decoder output (generated), and for the newly produced events (new).



(b) Comparison of distributions of the latent space variables (abbreviated lv). Distributions are plotted for the values obtained as encoder output (generated) and for the values used to produce new events (new), all of which are compared to the standard normal distribution (Gaussian noise).

Figure 6. Distributions of the six chosen features in the Higgs data set (a) and latent space variables (b).

these distributions have to match almost perfectly, which is harder to achieve. A few studies have been conducted for this purpose [17, 18] and show promising results. Further research is still needed however to improve the performance.

Generative models therefore seem an ever more suitable alternative to classical algorithms and might indeed be used in an actual analysis of the LHC data during the next couple of years.

REFERENCES

[1] ATLAS Collaboration, *The ATLAS Experiment at the CERN Large Hadron Collider*, Journal of Instrumentation **3** (2008), S08003.

[2] ATLAS Collaboration, *The ATLAS Simulation Infrastructure*, The European Physical Journal C **70** (2010), 823–874.

[3] ATLAS Collaboration, *Search for a heavy charged boson in events with a charged lepton and missing transverse momentum from pp collisions at* $\sqrt{s} = 13$ TeV *with the ATLAS detector*, Phys. Rev. D **100** (2019), 052013.

[4] T. Sjöstrand, S. Mrenna and P. Skands, *PYTHIA 6.4 physics and manual*, Journal of High Energy Physics **2006** (2006), 026.

[5] G. Marchesini, B. R. Webber, G. Abbiendi, I. G. Knowles, M. H. Seymour and L. Stanco, *HERWIG 5.1 - a Monte Carlo event generator for simulating hadron emission reactions with interfering gluons*, Computer Physics Communications **67** (1992), 465–508.

[6] T. Gleisberg, S. Hoeche, F. Krauss, A. Schaelicke, S. Schumann and J. Winter, *SHERPA 1. , a proof-of-concept version*, Journal of High Energy Physics **2004** (2004), 056.

[7] S. Agostinelli, J. Allison, K. Amako, J. Apostolakis et al., *Geant4 - a simulation toolkit*, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **506** (2003), 250–303.

[8] K. Bos, N. Brook, D. Duellmann, C. Eck, I. Fisk, D. Foster, B. Gibbard et al., *LHC computing Grid: Technical Design Report. Version 1.06 (20 Jun 2005)*, Technical Design Report LCG (CERN, Geneva, 2005).

[9] J. Elmsheuser, A. Di Girolamo, *Overview of the ATLAS distributed computing system*, EPJ Web Conf. **214** (2019), 03010.

[10] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio, *Generative Adversarial Networks*, arXiv:1406.2661 (2014).

[11] D. P. Kingma and M. Welling, *Auto-Encoding Variational Bayes*, arXiv:1312.6114 (2013).

[12] I. Goodfellow, *NIPS 2016 Tutorial: Generative Adversarial Networks*, arXiv:1701.00160 (2016).

[13] C. P. Burgess, I. Higgins, A. Pal, L. Matthey, N. Watters, G. Desjardins and A. Lerchner, *Understanding disentangling in* β*-VAE*, arXiv:1804.03599 (2018).

[14] P. Baldi, P. Sadowski and D. Whiteson, *Searching for exotic particles in high-energy physics with deep learning*, Nature Communications **5** (2014), 4308.

[15] D. Dua and C. Graff, *UCI Machine Learning Repository* (University of California, Irvine, School of Information and Computer Sciences, 2017), http://archive.ics.uci.edu/ml.

[16] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado et al., *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems* (2015), software available from tensorflow.org.

[17] S. Otten, S. Caron, W. de Swart, M. van Beekveld, L. Hendriks, C. van Leeuwen, D. Podareanu, R. R. de Austri and R. Verheyen, *Event Generation and Statistical Sampling for Physics with Deep Generative Models and a Density Information Buffer*, arXiv:1901.00875v2 (2019).

[18] B. Hashemi, N. Amin, K. Datta, D. Olivito and M. Pierini, *LHC analysis-specific datasets with Generative Adversarial Networks*, arXiv:1901.05282 (2019).