University of Cambridge
Department of Physics
Part III Project Report

Fast Simulation of Particle Physics Detectors using Machine Learning Tools

Candidate Number: 8259X
Supervisor: Dr. Christopher Lester

May 14, 2018

Abstract

Simulations of particle physics detectors are important as they provide a theoretical reference for experimental measurements, however current simulators are either too slow (GEANT4) or too imprecise (Delphes) to satisfy demand. Generative Adversarial Networks (GANs) are a relatively new idea in generative modeling but have nonetheless exploded from the field of Machine Learning through to the Natural Sciences. This report aims to assess the possibility of using GANs in a conditional setting to provide fast simulations of a detector's response. Two toy datasets, MNIST and CIFAR10, are first used to train on as a proof of concept for the GAN implementation proposed and realistic images of both datasets are shown to be produced by the generator. Simulation of a Delphes response to an incoming electron with properties p_T, η and ϕ is then performed to redress the particle physics balance, where it is shown that the GAN can produce simulations at a rate faster than Delphes and with a good degree of accuracy. This report represents a small advance in the direction of ultimately creating a fast generative model with the accuracy and precision of a full detector simulation and in a conditional setting.

Contents

1 Introduction

1.1 Motivation

Particle detector simulations have become a mainstay of High Energy Physics (HEP) experiments since their introduction in the late 1970s [1]. Before GEANT was released in 1978 [2], simulations of detectors were done in a rather ad-hoc way involving simple back of the envelope calculations. Since their inception, detector simulations have allowed HEP experiments to deliver physics results of outstanding accuracy and can take almost all of the credit.

If we turn to the current state of affairs, however, we find that HEP needs ever-increasing computing power as outdated legacy code can no longer exploit the full performance offered by modern day CPUs [3]. Experiments, as a result, have to make decisions on what to simulate since Full Simulations (FS) offered by GEANT4 are much too slow to satisfy the increasing demands that exist.

Delphes is an alternative to a FS that offers a parametric Fast Simulation (FASTSIM) of the detector response [4], however by its very nature is too imprecise for our needs as experiments probe higher and higher energies. It is this problem that this report looks to address by utilising the burgeoning field of Machine Learning (ML) as a tool to provide these fast simulations, ultimately with the accuracy and precision of a full detector simulation.

1.2 Literature Review

In every respect the field of HEP is playing catch-up with the ever expanding field of ML. There exists a plethora of innovation directed primarily at the capabilities of neural networks and their ability to learn complex non-linear mappings; their applications to particle physics are without end.

In 2014, Ian Goodfellow made a breakthrough in the field of generative modeling with his paper entitled "Generative Adversarial Networks" (GANs) [5] and this was widely regarded as the most significant discovery of the last 10 years within the ML community. Since their introduction, previously exciting ideas in ML, such as Deep Boltzmann Machines [6], have fallen by the way-side and one can find a new paper detailing another iteration of GANs almost every week. The MIT Technology Review have recently put GANs on a list of the "2018 Top 10 Breakthrough Technologies" [7], confirming their increasing relevance in generative modeling.

In this spirit, there have been a number of recent studies utilising GAN implementations in the field of particle physics. Luke de Oliveria et al. in 2017 used a Locally Aware GAN to generate 2D representations of energy depositions from particles interacting with a calorimeter [8]. More recently, they conducted a similar study which looked at producing these images with conditionality on the inputs [9]. Conditionality is particularly important because it means one is not constrained to generate a certain type of event, therefore allowing useful physics applications.

There is a notable gap in the literature for conditional generative modeling, however, especially in a particle physics context. This project will therefore endeavour to investigate the possibility of using conditional GANs in this setting, thereby filling those gaps.

1.3 Report Outline

This report takes a somewhat chronological ordering of ideas that made up the research.

Section 2 includes a theoretical background of conditional GANs (cGANs) along with an introduction to Delphes and the way in which it parameterises its inputs. Section 3 includes an overview of the implementations taken, such as building and training the GANs and applying them in physics. In section 4 the results will be presented and discussed, including limitations to the approach. Section 5 will finish this report with final conclusions.

2 Theoretical Background

2.1 Conditional Generative Adversarial Networks

2.1.1 Vanilla (c)GANs

Vanilla GANs refer to the early generation of GANs that were *first* proposed by Ian Goodfellow and then extended by other authors in the immediate fallout of his paper; one such variant is the conditional GAN [10], pictured below.

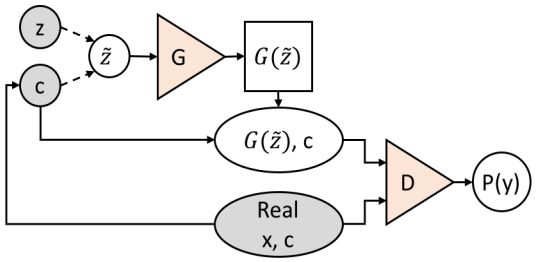


Figure 1: *A general vanilla cGAN architecture. This figure was taken from [11].*

The general cGAN framework is designed to learn a mapping from a simple latent distribution ($p_{\tilde{z}}$) to a given data distribution (p_{data}), conditioned on some class label. Conditionality is achieved by concatenating a class label (c) to the prior latent noise (z)[1], to form a joint hidden representation $\tilde{z} = [z\ c]$. The generator (G) provides a mapping $G(\tilde{z};\ \theta_G)$, which takes $p_{\tilde{z}}$ to the generator's distribution p_G, with the goal of bringing p_G close to p_{data}. The discriminator (D) in turn provides the mapping $D(x, c; \theta_D)$, outputting a scalar value $y \in [0,1]$. This value of y indicates the discriminator's prediction of whether the sample x it was fed originated from p_G or from p_{data}. The generator and discriminator are pitted against each other, playing a minimax game, defined by:

$$\underset{G}{min}\underset{D}{max}V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}\left[\log D(x,c)\right] + \mathbb{E}_{z\sim p_{\tilde{z}}(\tilde{z})}\left[\log(1 - D(G(\tilde{z}),c))\right] \quad (1)$$

The minimax game may be summarised by the flow diagram in Figure 2[2]. The targets for D and G are non-cooperative; this creates error gradients that are back-propagated through both the generator and discriminator for learning.

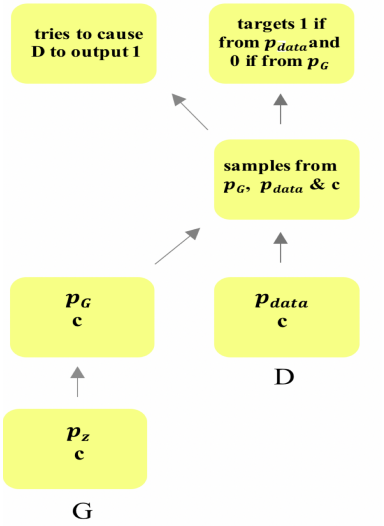


Figure 2: *A flow diagram of the non-cooperative objectives of the discriminator and generator of a vanilla cGAN during training.*

[1] Usually a Gaussian with μ=0, σ=1 and dimensions=100.

[2] Practically speaking, p_{data} = training image, c = training label and p_G = "fake" training image.

It is the hope of this adversarial training that the generator learns to fool the discriminator such that it is maximally confused and outputs 0.5 each time; this corresponds to the point when the generator has *approximated* the underlying distribution of p_{data} such that it produces realistic samples.

2.1.2 Wasserstein (c)GANs

The main problem with vanilla GANs presented above, however, is that there exists no useful way to ensure their convergence, or measure it. This becomes especially important when one attempts to train the generator to learn particularly complex and varied distributions. In a particle physics context this may be the detailed modeling of particle showers, the slowest step in a GEANT4 simulation.

The convergence in the minimax objective defined by equation 1 can be interpreted as minimising the Jensen Shannon (JS) divergence. However, in some limiting cases the JS divergence does not supply useful gradients for back-propagation through the generator [12], as demonstrated by Figure 3. The Wasserstein GAN (WGAN) implementation instead minimises the Earth Mover (EM) distance to reach equilibrium, avoiding the problem of vanishing gradients. The EM distance represents the minimum cost of transporting (probability) mass to transform the distribution p_G into the distribution p_{data}, and crucially is everywhere *continuous* and *differentiable*.

The objective function for the discriminator may now be written down as [13]:

$$W^{(D)}(D,G) = \mathbb{E}_{x\sim p_{data}}\left[D(x,c)\right] - \mathbb{E}_{z\sim p_{\tilde{z}}(\tilde{z})}\left[D(G(\tilde{z}),c)\right] \quad (2)$$

For the generator, the cost function is now simply:

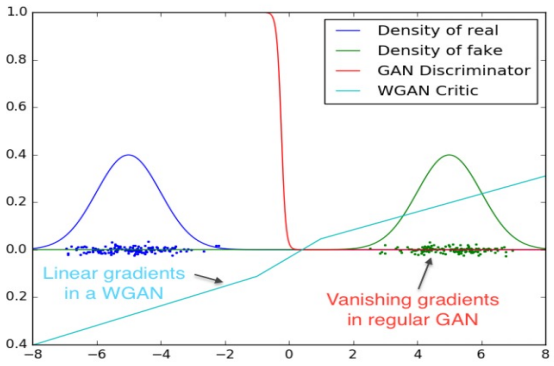


Figure 3: *The gradients produced for back-propagation by a vanilla GAN discriminator (red) and a WGAN "critic" (cyan) for non overlapping Gaussian distributions, p_{data} (blue) and p_G (green). The WGAN provides well behaved gradients for the entire range of values. This figure was taken from [12].*

$$W^{(G)} = -W^{(D)}(G,D) \quad (3)$$

The difference between equations 1 and 2 arises from the fact that the discriminator now emits an unbounded real number rather than a probability. In this sense the discriminator is no longer a discriminator, but a "critic". The flow diagram of Figure 2 may be modified to reflect this change as shown in Figure 4.

The original WGAN paper discusses the role of the critic as learning a K-Lipschitz continuous function to compute the EM distance. To ensure "Lipschitz smoothness" the paper suggests clipping the weights of the critic to lie within a range (-d,d), where d is some hyper-parameter. However, Improved WGANs were later developed to deal with the problems of using this weight clipping [14]. The paper suggests to instead use a gradient penalty to constrain the norm of the critic's output with respect to its input, which also acts to ensure "Lipschitz smoothness" [15]. This penalty is

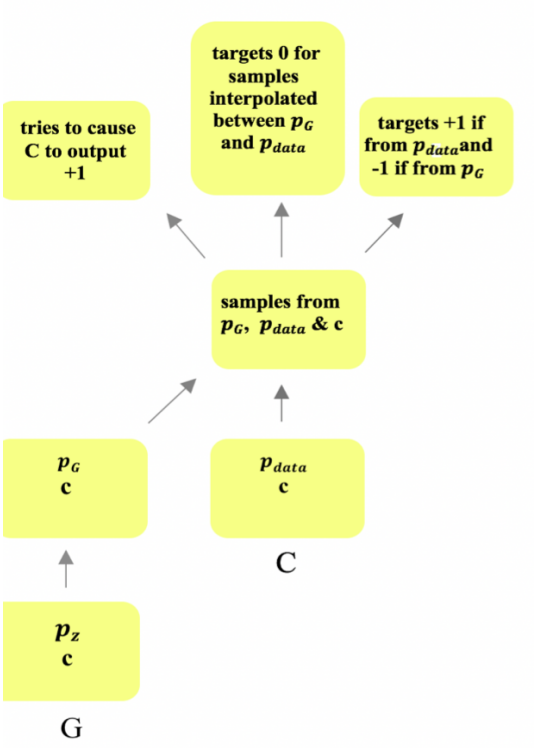


Figure 4: *A flow diagram of the non-cooperative objectives of the critic and generator of a cWGAN-GP during training. The target 0 corresponds to the critic's desired decision on an interpolated "image" in the GP case, whereas the new targets of +1 and -1 reflect the discriminator's new role as a critic.*

applied on a linear interpolation between data points and samples and acts therefore to smooth out the manifold between p_{data} and p_G; this is of paramount importance for complex distributions.

This Improved WGAN, in a conditional setting, shall be termed "cWGAN-GP" in this report, where GP stands for gradient penalty. This adaptation to a very recent iteration of GANs was successful on both the toy datasets as a proof of concept, but also for the particle physics applications, the motivation of this project.

For a full explanation, which expands in more detail all of the theory mentioned here, please see Appendix A.

2.2 Delphes

Delphes offers a FASTSIM approximation to a FS and is well used in HEP experiments around the world [16]. Delphes takes ∼0.1-1s to simulate an event compared with ∼10-1000s for a FS and thus provides a speed gain spanning many orders of magnitude. However, in practice the motivation of Delphes is mainly only for phenomenological studies, such as looking for the observability of given signals [17], since the stochastic parameterisations are unsatisfactory for really instructive particle physics applications. Despite this, for a primary investigation into conditional generative modeling of a particle detector's response, simulating Delphes is an apt and tangible place to begin.

The profile layout of the idealised detector simulated by Delphes is shown in Figure 5. The general work-flow of the modules that Delphes applies to an incident electron is shown in Figure 6, which is a simplified version of the one found in [18].

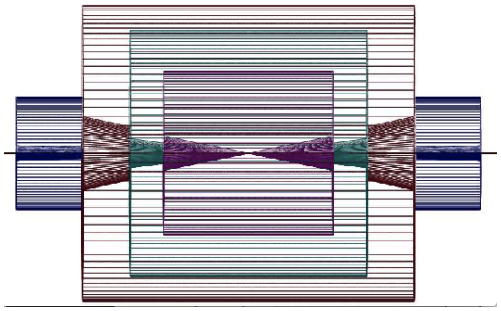


Figure 5: ***Pink**: Central tracking system. **Green**: Electromagnetic and Hadronic sections. **Red**: Muon system. **Blue**: End-cap calorimeters to extend the pseudorapidity. This figure was taken from [17].*

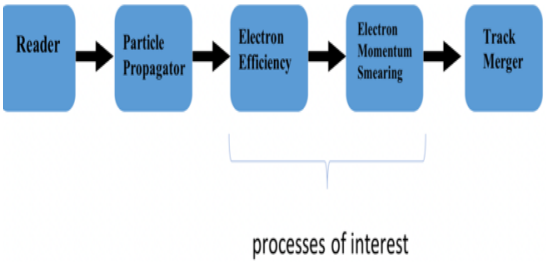


Figure 6: *A very general flow diagram of the processes Delphes applies to an incident electron. This project looks specifically at simulating Delphes' electron efficiency and electron momentum smearing modules.*

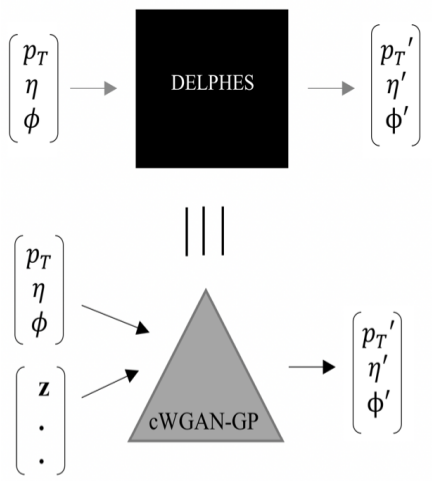


Figure 7: *This figure perfectly captures the problem at hand; the cWGAN-GP must learn the parameterisations of Delphes in order to supply a useful simulation.*

As already discussed, neural networks are by their very nature completely non-linear; in this way they are essentially a "black box", where one can have no understanding of the operations that happen within. Delphes' response, highlighted in Figure 6, involves applying an efficiency formula with an inherently probabilistic nature followed by a completely stochastic momentum smearing process to all detected electrons. These processes take an incident electron with properties (p_T, η, ϕ) to a response signaling an electron with properties ($p_T{}'$, η', ϕ'), where p_T is the transverse momentum of the electron, η is the pseudorapidity describing the angle of the electron relative to the beam axis and ϕ is the azimuthal angle. To provide useful simulations therefore, the cWGAN-GP must learn this mapping through exposure to the training dataset, with details of how this dataset was collected to follow in section 3.2.

With reference to Figure 7, the reason that we need prior random noise doesn't seem so obvious in this setting. However, a generator net G, applied to a single sample (p_T^1, η^1, ϕ^1) without noise, will learn only to generate one sample ($p_T^1{}'$, $\eta^{1\prime}$, $\phi^{1\prime}$) rather than an infinite number of different samples. This one-to-one correspondence would otherwise completely undermine the stochasticity of Delphes that we are trying to emulate. The problems caused by this requirement of noise will be discussed in section 4.

3 Methods

3.1 Implementation

All code was written in Python v3.5 using a Spyder notebook. Keras v2.0.4 [19] was used with a Tensorflow v1.0.6 [20] backend to write all of the ML programs due to the ease of the new functional API. All code was run on a Macbook Pro with specifications: 3.1GHz dual-core Intel Core i5, Turbo Boost up to 3.5GHz. A shortcoming of this project was that a more powerful system, such as the one recently installed in the HEP department, was not used. The results, although positive, could have been improved simply by running training for

longer which becomes more feasible with more power.

3.2 Datasets

Two open-source datasets, MNIST and CIFAR10, were used to train on for the preliminary stages of the investigation. Training data created by utilising the Delphes source code was then used for producing a generator that could emulate the detector response.

MNIST

MNIST is a labeled toy dataset with 60,000 training images and 10,000 test images (taken from p_{data}) of numbers ranging from 0 to 9 with dimensions of 28×28×1 [21]. The training labels are the numbers 0 to 9 attributed to a training image of that number.

CIFAR10

CIFAR10 is a labeled dataset of 50,000 training images and 10,000 test images (taken from p_{data}) with dimensions 32×32×3 (the 3 refers to the Red, Green and Blue (RGB) colour channels) [22]. The dataset is split in to the following 10 classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. These correspond to our training labels, c. The dataset is therefore incredibly varied and complex and so acted as an important, almost overreaching target. This, however, directed research into more stable and powerful GANs, which indeed is a requirement for our ultimate aim of modeling the most complex simulations of GEANT4.

Delphes

In order to bypass having to run entire simulations, snippets of the source code that apply the electron efficiency and electron momentum smearing formulae were used, taken directly from [23], an excellent repository of all Delphes' modules. The details of the electron momentum and electron momentum smearing processes are included for reference in Appendix B. Two datasets were used to investigate the success of cWGAN-GP at simulating Delphes, referred to as Dataset A and Dataset B.

For **Dataset A** a single condition was inserted into Delphes 1,000,000 times with arbitrary p_T, η and ϕ values of 4.25 GeV, 2.5 radians and 1.5 radians respectively. Three values of $p_T{}'$, η' and ϕ' were returned, thus creating the training data. This dataset was used to ensure that cWGAN-GP could accurately simulate Delphes for just one condition.

For **Dataset B** 1,000,000 real-valued inputs of p, η and ϕ were generated randomly, but uniformly, between values of 0-20 GeV, -2.5-2.5 radians and 0-2π radians respectively. Firstly, p_T was calculated using $p_T = \frac{p}{\cosh(\eta)}$. Once again, passing the vector (p_T, η, ϕ) through the source code that applies the parameterisations, one gets a vector containing three numbers back: $p_T{}'$, η' and ϕ', paired to the original three numbers. In the same way as *Dataset A*, these pairs of three numbers act as our training "images" ($'$) and training labels ().

Dataset B, the one used extensively in this report, includes only incident electrons with momentum of at most a magnitude of 20 GeV; this decision was made simply because if this value had been made bigger the training dataset would have been too sparse for a given size. Despite this, all pa-

rameterisations applied by Delphes, which depend upon the values of p_T and η, are covered within this range.

3.3 Architectures

MNIST and CIFAR10

The architectures for the MNIST and CIFAR10 datasets use convolutional layers since for natural images there is a direct correlation (in pixel value) between neighbouring pixels but not with pixels that are far away [24]; this is of course what gives natural images structure. Convolutional layers are only locally connected to achieve this goal. The full architectures are not included since they are already well understood and can be found in a lot of the literature, such as [25][3]. The MNIST and CIFAR10 networks had 7,497,858 and 18,337,348 parameters respectively, with this difference being due to the relative complexity of the datasets.

Delphes

For the Delphes dataset there are no correlations between the inputted training images and as such fully connected layers were used for learning. Two architecture depths were investigated for the Delphes datasets, referred to as Architecture 1 and Architecture 2.

Architecture 1 is shown in Figure 8. It has 5,747,716 parameters and as such may be defined as a medium sized network.

Architecture 2 has one extra dense layer of dimension 4096 compared to *Architecture 1* in both the generator and critic networks [4]. It has 22,768,644 parameters and therefore may be defined as a deep network.

One thing to note about both architectures is that the conditional label is added at every layer of the generator and critic. The most common implementations of cGANs in general involve simply feeding the label to the first layer. In this study, each conditional label contained only three values and so it proved more instructive to input the label more than once, thus minimising the possibility of the GAN "missing" the condition. This decision was inspired, in part, by [26].

3.4 Training

One of the drawbacks with the (c)WGAN implementation is that the training takes a long time to converge. This speed becomes especially slow with the gradient penalty approach implemented using (c)WGAN-GP because the model must interpolate between p_G and p_{data} and perform gradient descent on these "dummy" values as well, as highlighted in Figure 4.

The training used the Adam optimiser with a learning rate of 0.001 for both the generator and critic, with "beta1" and "beta2" values of 0.5 and 0.9 respectively. The Adam optimiser calculates an exponential moving average of the gradient and the squared gradient, where "beta1" and "beta2" control the decay rates of these moving averages [27]. Training times, although not hugely important if a good generator can be created eventually, are detailed in the sections below.

[3] All architectures, including these two, can be found on the memory stick handed in with this report.

[4] Datasets A and B and Architectures 1 and 2 will be referred to in the results and discussion section.

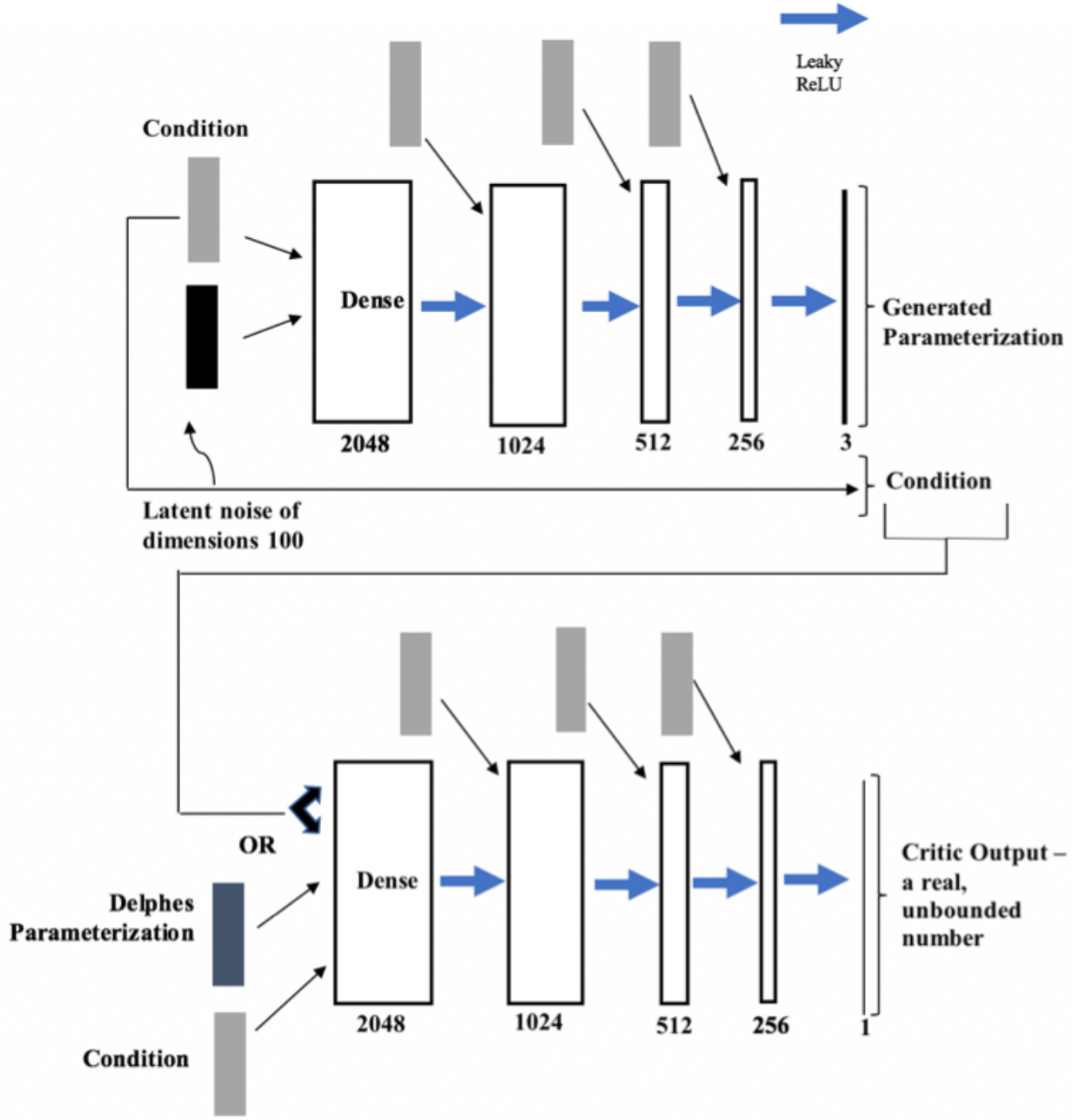


Figure 8: *Architecture 1 for the generator and critic networks of the implemented cWGAN-GP on the Delphes datasets. This architecture was used for the brute force approach relating to the loss graph in Figure 11.*

MNIST

MNIST took ∼2.5 hours to train using 10,000 training images, where training was stopped after just 25 epochs due to convergence and the quality of the outputs. This dataset actually only required a vanilla cGAN due to its simplicity.

CIFAR10

CIFAR10 took ∼58 hours to train using 17,500 training images, where training was stopped after 250 epochs. Generating new images from the trained model, however, was quick; 50 images from each class were generated afterwards in ∼20s.

Delphes

Delphes was trained using various combinations of the architectures and datasets mentioned previously. Firstly $\frac{1}{20}$ of *Dataset A* was used to train with *Architecture A* over 200 epochs and took ∼2 hours. Next, a brute force approach using *Architecture 1* and *Dataset B* took ∼93 hours to train over 200 epochs. The training was finally implemented using $\frac{1}{10}$ of *Dataset B* with the deeper network of *Architecture 2* and then this same fraction using *Architecture 1* but for 500 epochs. Both of these training times were on the order of $\mathcal{O}(10)$ hours.

4 Results and Discussion

MNIST

MNIST is an easy dataset to train on, however this exercise provided an important starting point for this investigation. The training converged incredibly quickly (∼4-5 epochs) and satisfactory images were being generated after just a couple of epochs, as shown in Figure 9.

Figure 9: *Generated MNIST images after 25 epochs of training.*

CIFAR10

CIFAR10 is one of the hardest and most varied datasets to train on and thus provided a significant challenge. The generator's task in any GAN implementation is to map an essentially infinite set of prior noise samples to approximate images of the training set. To "help the generator", [28] suggests feeding the same *set* of noise samples to the generator at each epoch, which is already an improvement on infinity. To extend this idea, [29] proposes various methods of making the prior noise more reflective of the training data. One of these methods involves compressing the training images to 10x10x3 and converting them from three colour channels to one; the full scheme for clarification is included in Appendix C. The "noise" fed into the generator for training after this implementation would therefore be static but also more reminiscent of the training data. These ideas were utilised for CIFAR10 and the generated images became considerably more realistic as shown in Figure 10, confirming this theoretical proposal. A vanilla cGAN turned out not to be satisfactory for this dataset unlike for the MNIST case; it was indeed this avenue that led to the ultimate implementation of cWGAN-GP for simulating Delphes.

CIFAR10, quite significantly, overreaches the current goal of using GANs in a conditional setting towards fast particle detector simulations. However, this avenue was investigated not only because of interest, but also with the long term view

Figure 10: *Generated CIFAR10 images after 250 epochs of training. I shall not restate the classes here as I hope the reader may be able to classify the images for themselves.*

of modeling full detector simulations with this technology. This success therefore represents a powerful proof of concept; that GANs can handle very complex data distributions and in a conditional setting, where this conditionality on CIFAR10 can not be found in the literature.

Delphes

The training data for Delphes is different in one major respect to the MNIST and CIFAR10 datasets; the conditionality space is effectively continuous. In fact, no literature exists for which the conditionality imposed on one of these neural networks differs from simply a "one-hot vector" of N dimensions with class n being denoted by $(0\ 0\ 0\ \ldots 1\ \ldots 0)$, with the 1 in the n^{th} position. This infinite conditionality space causes significant problems during training; how after all can a GAN learn when it only has access to a fraction of the distribution? Further to this, a GAN may only learn given *enough* training data and as such will struggle to capture the tail of a distribution, where the frequency of those occurrences are far lower. Despite these limitations, the cWGAN-GP was trained to simulate Delphes, where the training strategy was varied to investigate how effective different methods of learning the latent manifold could be.

In order to test whether the GAN could transform an incident electron in a "Delphes-like-way" post training, it was decided to be most instructive if the "same electron" was fed into both detector simulations (Delphes and cWGAN-GP) a large number of times and their output distributions compared. In this way an average response to one particular electron can be ascertained, providing a way to quantify the success of cWGAN-GP. This decision came from an (erroneous) initial motivation of comparing the similarity of distributions created by many *different* electrons, such as a log distribution. The reason this method turned out not to be particularly informative was because there was no measure on what cWGAN-GP was doing to any one particular electron in an event of 1000.

The motivation of creating *Dataset A* was due to the initial concerns about the complexity of the latent manifold. cWGAN-GP was therefore first trained using this dataset, with the generated simulation after training shown in Figure 12 on the next page. One can see that the cWGAN-GP could mostly capture the distribution generated by Delphes, struggling only at the tail. This implementation however provided no variance on the conditionality and as such this model struggled generating realistic distributions with different conditional labels as input, which is ex-

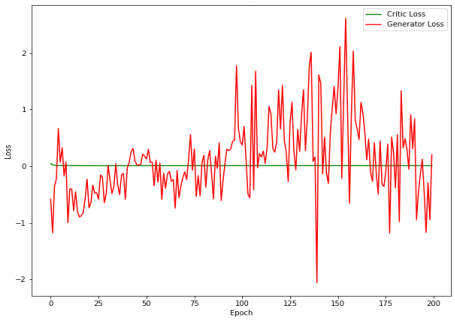
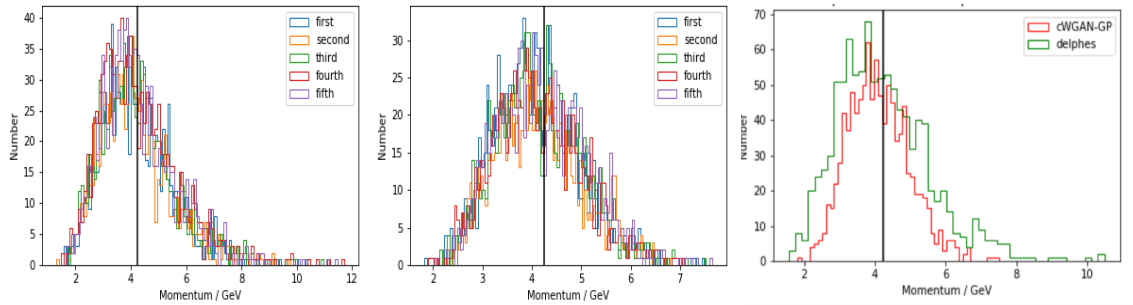


Figure 11: *Loss graph of the generator and critic over 200 epochs trained on Dataset B using Architecture 1.*

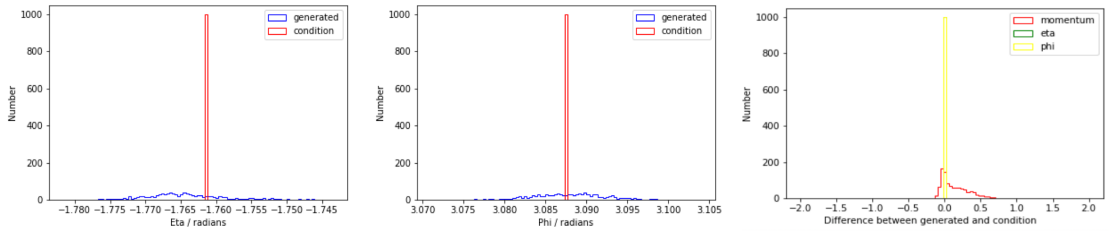


(a) *Momentum distribution produced by Delphes with p_T, η and ϕ values of 4.25 GeV, 2.5 radians and 1.5 radians respectively.*

(b) *Momentum distribution produced by cWGAN-GP with p_T, η and ϕ values of 4.25 GeV, 2.5 radians and 1.5 radians respectively.*

(c) *A comparison between the distributions in sub-figures (a) and (b); the GAN struggles at the tail where there is little to train on.*

Figure 12: *The simulated response of Delphes and cWGAN-GP to an identical incident electron.*



(a) *Produced η distribution after 200 epochs with a random condition of -1.762 radians.*

(b) *Produced ϕ distribution after 200 epochs with a random condition of 3.087 radians.*

(c) *A difference plot between a condition fed in to cWGAN-GP 1000 times and the produced output.*

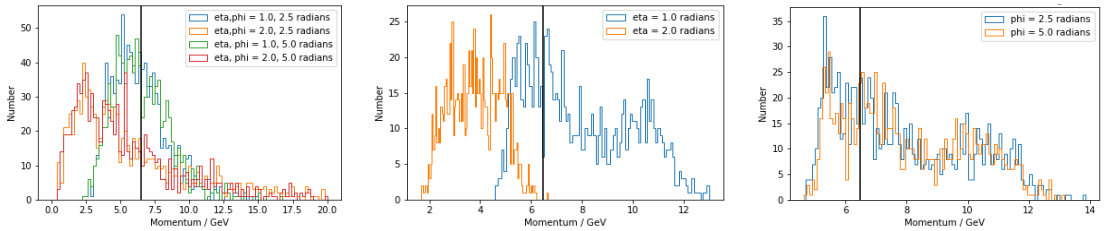
Figure 13: *A demonstration that the generated η and ϕ distributions lie very close to their conditional labels, as desired.*

pected.

The cWGAN-GP was then trained on *Dataset B* using *Architecture A*. The random way in which this dataset was collected meant that there were likely to be no two conditions the same. One motivation for training in this way was to encourage the GAN to learn the distribution in a very non-superficial way. Figure 11 gives the evolution of the training loss of both the generator and the critic in this setting over 200 epochs. The critic's loss fell rapidly to 0, which means that the generator received useful feedback with *every* update (a motivation of cWGAN-GP). However, because the loss functions in the cWGAN-GP implementation relate to the quality of the outputs, we can see that because the generator was still oscillating about 0 it had not converged and thus could not generate wholly "satisfactory" images.

Of course the number of plots[5] that one could produce with different values of p_T, η and ϕ to test the trained model are infinite;

[5]N.B. For all plot labels "Momentum" refers to transverse momentum.



(a) *The "truth" distributions generated by Delphes, varying η and ϕ while keeping p_T constant. These distributions should be compared with the generated ones in sub-figures (b) and (c).*

(b) *Generated distributions where only the value of η has been changed. Despite the condition of η being unaffected, its value has an effect on the p_T distribution as captured by cWGAN-GP.*

(c) *Generated distributions where only the value of ϕ has been changed. Since ϕ is simply a spectator condition we hope that changing its value should have no effect, which is what we see.*

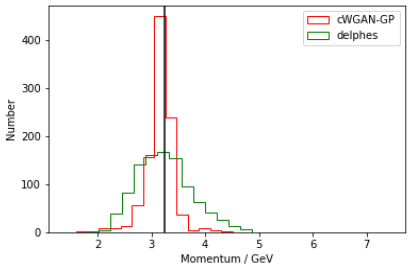
Figure 14: *The simulated response of Delphes and cWGAN-GP to an identical incident electron.*

the key is to find what the GAN performed well in amongst all of this. The parameterisations detailed in appendix B make no change to the values of η and ϕ. Figure 13 shows that indeed the generated η and ϕ values were within their conditional labels to a very good accuracy. This is especially comforting if we remember that the input prior noise is taken from a Gaussian distribution with standard deviation 1, which naturally causes a broadening of the outputs. Despite the fact that η and ϕ should not be affected by the parameterisations, in reality it is only ϕ which is a true spectator, since the magnitude of η affects both the electron efficiency and the momentum smearing formulae applied. The cWGAN-GP was therefore tested to see if it had captured this subtlety. Figure 14 shows the effect of varying η and ϕ with the same transverse momentum on the GAN distributions; that is from the cWGAN-GP's perspective, keeping the value of the first condition, p_T, constant, while varying the other two. One can see, with comparison to the given Delphes output in the same figure, that the cWGAN-GP has indeed learned that ϕ plays no part in the p_T distribution but that η does, and in a vaguely correct and satisfying way.

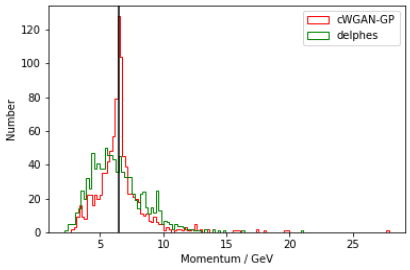
In truth, this brute force method of training the cWGAN-GP (training time ∼93 hours) could not provide an accurate simulation of Delphes, and is only a reasonable approximation within a certain region of the training dataset. Even though the GAN manages to capture the dependence of the p_T distribution on η and ϕ, it struggles to generate an accurate simulation across the high and low energy regimes.

In order to investigate ways to improve this training, the cWGAN-GP was also trained using a smaller subset of *Dataset B*, once with *Architecture 2* and once again with *Architecture 1*, but this time for more epochs[6]. In both of these extended implementations a Gaussian with mean 0 and standard deviation 2 was used as the prior noise, motivated by the need for a larger tail as we go to higher and higher energies. In both cases, the outputs were certainly improved, however the extra deviation on the noise acted to amplify the error

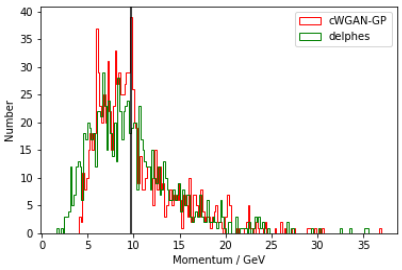
[6]See section 3.4: Delphes for details of this.



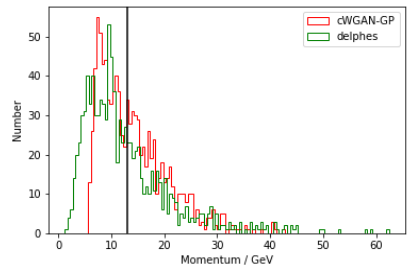
(a) $p_T = 3.24\ GeV$, $\eta = 1.0$ and $\phi = 2.5$



(b) $p_T = 6.48\ GeV$, $\eta = 1.0$ and $\phi = 2.5$

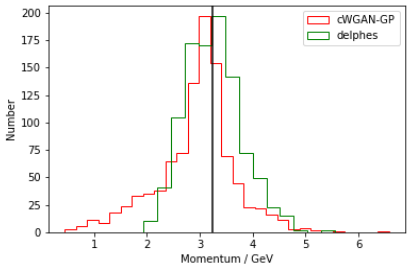


(c) $p_T = 9.72\ GeV$, $\eta = 1.0$ and $\phi = 2.5$

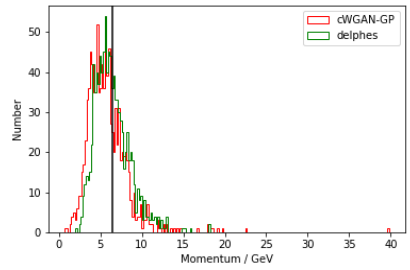


(d) $p_T = 12.96\ GeV$, $\eta = 1.0$ and $\phi = 2.5$

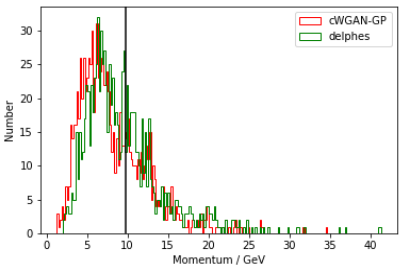
Figure 15: *Results generated for a larger network with 22,768,644 parameters trained for 100 epochs with 100,00 training images.*



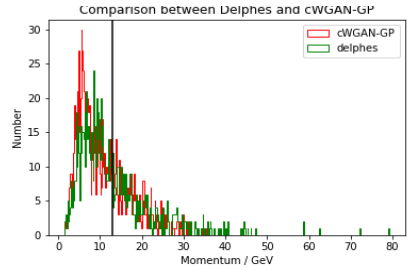
(a) $p_T = 3.24\ GeV$, $\eta = 1.0$ and $\phi = 2.5$



(b) $p_T = 6.48\ GeV$, $\eta = 1.0$ and $\phi = 2.5$



(c) $p_T = 9.72\ GeV$, $\eta = 1.0$ and $\phi = 2.5$



(d) $p_T = 12.96\ GeV$, $\eta = 1.0$ and $\phi = 2.5$

Figure 16: *Results generated for a smaller network with 5,747,716 parameters trained for 500 epochs on a dataset with 100,000 training images.*

on the angles η and ϕ as well as partially degrade the distribution at low p_T. Both of these networks took a much shorter time to train than in the brute force case because considerably less training data was used. The quality of the outputs for these regimes might point to an over-fitting problem in the primary case, where the GAN may have been trying to look for a one-to-one correspondence rather than learn a set of rules, which is the hope. The outputs of these two networks are shown in Figures 15 and 16 respectively.

The cWGAN-GP with *Architecture 2* (Figure 15) can be seen to struggle less with noise at lower momenta than the one with *Architecture 1* (Figure 16), a shallower network. This might correspond to the GAN compartmentalising its network for each regime such that no "leakage" from different training areas is observed. We can see that with the modified Gaussian noise, the detector simulation can reach the tail of the distribution more effectively than in Figure 14(b) (orange) from the earlier implementation. Indeed, the plots in Figure 16 show an excellent agreement to Delphes, except at lower energy levels. The answer, it seems, would be to combine a larger network with a longer training time such that different regimes can be modeled by the GAN, and accurately. One must keep in mind, however, that the larger the network the slower the ultimate generation step. Generation times for *Architecture 2* with $\sim 4\times$ as many parameters as *Architecture 1* was $\sim 2.5\times$ slower. Clearly when these generation times approach those of Delphes this method of simulating particle detectors becomes obsolete.

Lastly, to justify the use of the cWGAN-GP, a vanilla cGAN was also used to train on the Delphes dataset. The training was found to fall into local minima unless the dataset was prepared in exactly the way described in section 3.2; the cGAN could not handle the disjointedness of training on events $p_T = 0.0,\ 2.0,\ 5.0,\ 10.0,\ \ldots$ GeV. Even when training seemed not to fall in to these minima, the cGAN struggled considerably with accurate generation and could not extrapolate outside of regions of the dataset to any degree.

Of course, the main motivation of simulating these particle physics detectors using GANs is for a performance gain in terms of speed. Once trained, the primary cWGAN-GP was used to generate 1,000,000 events. This took approximately 1840 seconds, which translates to $\sim$0.002 s/event, a performance gain over Delphes of $\mathcal{O}(100)$, a satisfactory improvement. This result, coupled with the quality of the outputs, therefore justifies the approach of generating these simulations using ML techniques and proves there is an advantage of pursuing this avenue further, provided our networks don't get too big.

5 Conclusions

This project endeavored to explore the feasibility of applying ML techniques to produce fast simulations of a particle detector response. Due to their promise since their introduction in 2014, GANs were chosen as the tool to provide these simulations. The real emphasis of this project was to produce a generative model that could produce simulations in a conditional sense, thus allowing useful physics applications. Enforcing conditionality is difficult, however, as the extra information encoded in the inputs dramatically complicates the latent manifold. As such, conditionality was tested on with two toy datasets (MNIST and CI-

FAR10) as a proof of concept and satisfactory results were produced.

The particle physics was addressed by simulating the response of an idealised detector (as modeled by Delphes) to an incident electron. The training set was compiled by taking relevant parts of the Delphes source code that applies the electron efficiency and electron momentum smearing formulae. The chosen GAN (cWGAN-GP) was then trained in a number of different ways using various iterations of this dataset with the aim of mimicking Delphes and thus providing an informative simulation.

The results of this project prove that indeed we get a performance gain in terms of speed with these ML derived simulations. However, significant training time and a sufficiently large neural network is required to properly train on an effectively continuous (and therefore infinite) training set. The results produced in this report, which were indeed only limited by the available computation, suggest however that this *is* a method with promise. The implemented cWGAN-GP could be shown to handle conditionality and be stable to both mode collapse and spurious minima, unlike the vanilla cGAN. The produced distributions were satisfactory and the potential for improvement on this, I believe, simply requires "scaling-up" the implementations set out in this report.

Most conditional GAN implementations in the area of particle physics have looked only at auxiliary classification downstream of the generation process rather than conditionality on the inputs. This project is therefore novel, but future work is required to investigate the difficulties caused by a non-discrete conditionality space, especially when we ask these GANs to generate simulations with more complexity.

Acknowledgements

I would like to thank Dr. Christopher Lester who was always on hand whenever I sought his guidance. I would also like to thank Michael Albergo who proved a great help at the beginning of my research. Lastly, I'd like to thank Andrius Gabrauskas who provided help with the finer intricacies of coding whenever I struggled finding a minute error in my code.

References

[1] V. D. Elvira, "Impact of detector simulation in particle physics collider experiments," *Physics Reports*, vol. 695, pp. 1–54, 2017.

[2] R. Brun, R. Hagelberg, M. Hansroul, and J. C. Lassalle, *Simulation program for particle physics experiments, GEANT: user guide and reference manual.* Geneva: CERN, 1978.

[3] G. Amadio, J. Apostolakis, M. Bandieramonte, A. Bhattacharyya, C. Bianchini, R. Brun, P. Canal, F. Carminati, L. Duhem, D. Elvira, *et al.*, "The geantv project: preparing the future of simulation," in *Journal of Physics: Conference Series*, vol. 664, p. 072006, IOP Publishing, 2015.

[4] J. De Favereau, C. Delaere, P. Demin, A. Giammanco, V. Lemaitre, A. Mertens, M. Selvaggi, D. . Collaboration, *et al.*, "Delphes 3: a modular framework for fast simulation of a generic collider experiment," *Journal of High Energy Physics*, vol. 2014, no. 2, p. 57, 2014.

[5] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, pp. 2672–2680, 2014.

[6] H. Salakhutdinov, "Deep boltzmann machines," *Proceedings of the International Conference on Artificial Intelligence and Statistics.*, 2009.

[7] D. Rotman, "10 breakthrough technologies 2018," *MIT Technology Review*, 2018.

[8] L. de Oliveira, M. Paganini, and B. Nachman, "Learning particle physics by example: location-aware generative adversarial networks for physics synthesis," *Computing and Software for Big Science*, vol. 1, no. 1, p. 4, 2017.

[9] M. Paganini, L. de Oliveira, and B. Nachman, "Calogan: Simulating 3d high energy particle showers in multi-layer electromagnetic calorimeters with generative adversarial networks," *arXiv preprint arXiv:1705.02355*, 2017.

[10] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv preprint arXiv:1411.1784*, 2014.

[11] N. Kodali, J. Hays, J. Abernethy, and Z. Kira, "On convergence and stability of gans," 2018.

[12] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan," *arXiv preprint arXiv:1701.07875*, 2017.

[13] W. Fedus, M. Rosca, B. Lakshminarayanan, A. M. Dai, S. Mohamed, and I. Goodfellow, "Many paths to equilibrium: Gans do not need to decrease adivergence at every step," *arXiv preprint arXiv:1710.08446*, 2017.

[14] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein gans," in *Advances in Neural Information Processing Systems*, pp. 5769–5779, 2017.

[15] C. Fabbri, "Conditional wasserstein generative adversarial networks," *University of Minnesota*, 2018.

[16] A. Giammanco, "The fast simulation of the cms experiment," in *Journal of Physics: Conference Series*, vol. 513, p. 022012, IOP Publishing, 2014.

[17] S. Ovyn, X. Rouby, and V. Lemaitre, "Delphes, a framework for fast simulation of a generic collider experiment," *arXiv preprint arXiv:0903.2225*, 2009.

[18] B. Siddi, "Fully parameterised fast monte carlo simulation in lhcb," *2016 IEE Nuclear Science Symposium*, 2016.

[19] F. Chollet *et al.*, "Keras," 2015.

[20] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[21] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[22] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.

[23] M. Selvaggi, "Github repositry, https://github.com/selvaggi/delphes," 2016.

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[25] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[26] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee, "Generative adversarial text to image synthesis," *arXiv preprint arXiv:1605.05396*, 2016.

[27] J. Brownlee, "Gentle introduction to the adam optimization algorithm for deep learning," 2017.

[28] S. Selseng and B. Gambäck, "Improving discriminator-generator balance in generative adversarial networks," 2018.

[29] T. Arici and A. Celikyilmaz, "Associative adversarial networks," *arXiv preprint arXiv:1611.06953*, 2016.

[30] S. Selseng, "Guiding the training of generative adversarial networks," 2017.

Appendix A: Wasserstein GAN Theory Extended Version

The original WGAN paper referenced above ([12]) was met with much excitement in the ML community. This was in part due to the fact that the proposed GAN produced positive results, but mainly because there was a theoretical reason why. In the main body of this report I tried to summarise the surrounding theory as clearly and succinctly, but also as briefly, as possible. I believe a longer explanation (which will therefore cover some ideas touched upon in the main body) is appropriate, for the interested reader, as well as for completeness of this report.

Vanilla GANs minimise the Jensen-Shannon (JS) divergence, where this divergence is a measure of the similarity between two probability distributions, say p_G and p_{data}. The JS divergence is a symmetrical and smoothed version of the Kullback-Leibler divergence and is defined by:

$$D_{JS}\left(p||q\right) = \frac{1}{2}D\left(p||\frac{p+q}{2}\right) + \frac{1}{2}D_{KL}\left(q||\frac{p+q}{2}\right). \tag{A1}$$

As we know, a GAN consists of two models, a discriminator D and a generator G, who play a non-cooperative minimax game. Since each model updates its cost independently, irrespective of the other, convergence cannot be guaranteed. However, it can be shown that once the generator is trained properly such that $p_G \rightarrow p_{\text{data}}$, then $D(\text{x}) \rightarrow 0.5$. In this case the global optimal for the loss function is equal to -2log2, where this loss function quantifies the similarity between p_G and p_{data}.

It is, however, hard to achieve this Nash equilibrium and as such training GANs is not easy and the process is prone to be slow and unstable. The main problem of training arises because both p_G and p_{data} lie on low dimensional manifolds. This means that p_G and p_{data} are most certainly going to be disjoint. With these disjoint supports it is always possible to train a discriminator that outputs:

$$D(x) = 1, \forall x \in p_{\text{data}}; \quad D(x) = 0, \forall x \in p_G. \tag{A2}$$

We meet a problem at this point, however. If the discriminator doesn't converge properly it cannot supply *useful* gradients for the generator to learn from. If the discriminator, on the other hand, converges such that its loss function falls to 0, as would be the case in equation A2, there is *no* gradient to back-propagate through the generator and the learning stops completely. If we further add the problem of mode collapse and the lack of a proper evaluation metric, the problems with vanilla GANs becomes a concern. These problems all arise from the trouble of minimising the JS divergence given in equation A1.

The motivation of the Wasserstein GAN implementation is to avoid the problem of vanishing gradients by minimising another distance metric; the Earth Mover (EM) distance. This is again a measure of the distance between two probability distributions and can be thought of as the minimum amount of work required to move "some" (probability) mass a distance "x" from one probability distribution, p_G, to another, p_{data} located at y. It may be written down as:

$$W(p_{\text{data}}, p_G) = \inf_{\gamma \sim \Pi(p_{\text{data}}, p_G)} \mathbb{E}_{(x,y)\sim\gamma}[\|x - y\|]. \tag{A3}$$

Where $\Pi(p_{\text{data}},\ p_G)$ is the set of all possible joint probability distributions between p_{data} and p_G, where any one of these satisfy $\gamma \in \Pi(p_{\text{data}},\ p_G)$.

We can, however, only compute an approximation to equation A3 since computing the Wasserstein distance *exactly* is intractable. A result from Kantorovich-Rubinstein duality shows that W above is equivalent to:

$$W(p_{\text{data}}, p_G) = \sup_{\|f\|_L \leq K} \mathbb{E}_{x\sim p_{\text{data}}}\left[f(x)\right] - \mathbb{E}_{x\sim p_G}\left[f(x)\right]. \tag{A4}$$

The supremum is taken over all 1-Lipschitz functions, where supremum is the opposite of infimum meaning we wish to find the least upper bound.

The requirement that $\|f\|_L \leq K$ means that f must be K-Lipschitz continuous. f is called K-Lipschitz continuous if there exists a real constant $K \geq 0$ such that:

$$|f(x_1) - f(x_2)| \leq K|x_1 - x_2|. \tag{A5}$$

As a result, functions that are everywhere differentiable are Lipschitz continuous. If this function f comes from a family of K-Lipschitz continuous functions parameterised by w, then the discriminator in the WGAN model uses w to find a good f_w, where w in this context refers to the weights of the network. The loss function we therefore arrive at is given by:

$$L(p_{\text{data}}, p_G) = W(p_{\text{data}}, p_G) = \max_{w\in W} \mathbb{E}_{x\sim p_{\text{data}}(x)}\left[f_w(x)\right] - \mathbb{E}_{z\sim p_{\text{data}}(z)}\left[f_w(g_\theta(z))\right]. \tag{A6}$$

The discriminator, or more correctly, the critic, is therefore trained to learn a K-Lipschitz continuous function in order to compute the Wasserstein distance. In order to maintain the K-Lipschitz continuity of f_w during the training, the original paper suggests clamping the weights w of the critic to within a range of [-d,d], where d is some hyper-parameter.

Improved Wasserstein GANs [14] were developed to deal with the problems of using weight clipping of the critic to enforce the Lipschitz constraint. Problems that arise from this involve capacity under-use and exploding and vanishing gradients *without* careful tuning of the hyper-parameter d. The paper instead proposes a gradient penalty to constrain the norm of the critic's output with respect to its input. This works because a function is 1-Lipschitz only if it has gradients with a norm of at most 1 everywhere. This penalty is applied on a linear interpolation between data points and samples, and acts therefore to smooth out the manifold between p_{data} and p_G. The final revised critic loss function for this Improved WGAN, and the one used in this project, but in a conditional setting, may be written:

$$W(p_{\text{data}}, p_G) = -\mathbb{E}_{x\sim p_{\text{data}}(x)}\left[f_w(x)\right] + \mathbb{E}_{z\sim p_z(z)}\left[f_w(g_\theta(z)\right] + \lambda \mathbb{E}_{\hat{x}\sim p_{\text{data}}}[(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2]. \tag{A7}$$

Appendix B: Delphes Source Code

Two Delphes paramaterisations were simulated as part of this report.

The first of these is **electron efficiency** defined by:

```
set EfficiencyFormula {

(pt <= 0.1)    * (0.00) +
(abs(eta) <= 1.5) * (pt > 0.1   && pt <= 1.0)    * (0.73) +
(abs(eta) <= 1.5) * (pt > 1.0   && pt <= 1.0e2) * (0.95) +
(abs(eta) <= 1.5) * (pt > 1.0e2)                 * (0.99) +
(abs(eta) > 1.5 && abs(eta) <= 2.5) * (pt > 0.1 && pt <= 1.0) * (0.50) +
(abs(eta) > 1.5 && abs(eta) <= 2.5) * (pt > 1.0 && pt <= 1.0e2) * (0.83) +
(abs(eta) > 1.5 && abs(eta) <= 2.5) * (pt > 1.0e2) * (0.90) +
(abs(eta) > 2.5) * (0.00)

}.
```

The second of these is **momentum smearing**, defined by:

```
 set ResolutionFormula {

(abs(eta) <= 0.5) * (pt > 0.1) * sqrt(0.03^2 + pt^2*1.3e 3^2) +
(abs(eta) > 0.5 && abs(eta) <= 1.5) * (pt > 0.1) * sqrt(0.05^2
                                                  + pt^2*1.7e 3^2) +
(abs(eta) > 1.5 && abs(eta) <= 2.5) * (pt > 0.1) * sqrt(0.15^2
                                                  + pt^2*3.1e 3^2)

}.
```

This momentum resolution formula is then applied in the following way, for an electron with properties (p_T, η, ϕ), where we note η and ϕ remain unchanged:

```
res = ResolutionFormula >Eval(pt, eta, phi)
res = ( res > 1.0 ) ? 1.0 : res
pt = LogNormal(pt, res * pt ).
```

Where the "LogNormal" function is defined by:

```
LogNormal(Double_t mean, Double_t sigma)
{
  Double_t a, b;
  if(mean > 0.0)
  {
    b = TMath::Sqrt(TMath::Log((1.0 + (sigma*sigma)/(mean*mean))));
    a = TMath::Log(mean) [minus] 0.5*b*b;
    return TMath::Exp(a + b*gRandom >Gaus(0.0, 1.0));
  }
  else
  {
    return 0.0;
  }
}.
```

N.B. These are all taken from the source code, which is written in C++; they were therefore adapted for Python accordingly.

Appendix C: Image Based Noise Generation

This scheme of generating noise samples from the training data to help guide the generator to the correct convergence is inspired from [30].

The prior random noise, taken from a Gaussian distribution of mean 0 and standard deviation 1, may be represented by a 10×10(×1) array, shown in Figure 1.

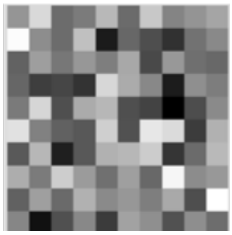


Figure C1: *Example prior random noise for the generator.*

The first step in the scheme takes an image from the training set and compresses it from 32×32×3 to 10×10×3.

Figure C2: *The first transformation of the training set image of class 7 of the CIFAR10 dataset.*

The next step takes the average of the pixel values from each of the Red, Green and Blue (RGB) colour channels to produce an image of the form in C1; a 10×10(×1) array.

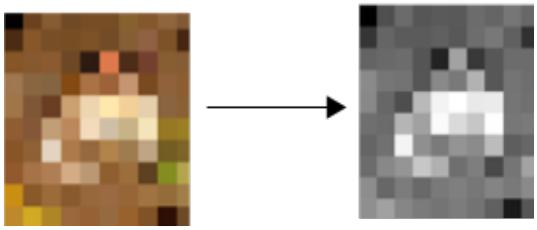


Figure C3: *The second transformation of the training set image from class 7 of the CIFAR10 dataset.*

As a result we have a static set of noise samples (the generator sees the same samples at each epoch), where these noise samples are derived from the training set. This formulation helps the GAN to generate better images and is shown not to reduce diversity of the generated samples, which is important.