UIMP
Universidad Internacional
Menéndez Pelayo

ULTRA FAST SIMULATION ALGORITHMICS FOR INDUSTRIAL APPLICATIONS IN MUON TOMOGRAPHY USING GENERATIVE ADVERSARIAL NETWORKS

Trabajo de Fin de Máster

para acceder al

MÁSTER INTERUNIVERSITARIO (UC-UIMP) EN FÍSICA DE PARTÍCULAS Y DEL COSMOS

Autor: Rubén López Ruiz

Director: Pablo Martínez Ruiz del Árbol

Codirectora: Celia Fernández Madrazo

Julio - 2022

Abstract

Muon tomography is a Non-Destructive Testing technique that consists in using cosmic muons as a probing tool, in order to generate images of objects from the information given by the interaction with the muons. The development and application of this techniques requires the production of considerable amounts of simulation data, usually generated with complex and slow particle simulation software. In this work, we explore the use of Generative Adversarial Networks (GAN) as a way of generating simulation data for muon tomography applications in a faster and less computationally expensive way. We have observed that GAN architectures can nicely reproduce the process of propagation of cosmic muons crossing material 50 times faster than other simulation software.

Key words: muon tomography, cosmic muons, generative adversarial networks, simulation, machine learning

Resumen

La tomografía muónica es una técnica de caracterización no invasiva que consiste en utilizar muones cósmicos como herramienta de sondeo, para así crear imágenes de objetos a partir de la información dada por la interacción de los muones con dicho objeto. El desarrollo e implementación de esta tecnología requiere disponer de grandes cantidades de datos de simulación, que son actualmente generados con software específico de simulación de partículas. Este software tiene la desventaja de ser complejo y lento. Por ello, en este trabajo se explora el uso de redes generativas adversarias (GAN) como método de producir datos de simulación para aplicaciones de tomografía muónica de una forma mucho más rápida y menos costosa. Se ha observado que las arquitecturas tipo GAN pueden reproducir bien el proceso de propagación de los muones cósmicos a través de materiales con un aumento en la velocidad de simulación de un factor 50.

Palabras clave: tomografía muónica, muones cósmicos, redes generativas adversarias, simulación, aprendizaje automático

Acknowledgements

Este proyecto es producto de meses de trabajo en los que he disfrutado mucho, y que no hubiera sido posible sin la ayuda de muchas personas que me han acompañado en este camino.

En primer lugar, me gustaría dar las gracias a Pablo y a Celia, por darme esta increíble oportunidad, por creer en mí, y por transmitirme todo su apoyo, conocimiento y experiencia a lo largo de estos meses. Sin duda habéis hecho de este tiempo una experiencia mucho más amena y agradable.

Agradecer también al IFCA, y a mis profesores del Máster por todas las herramientas y conocimiento transmitido, que sin duda han contribuido a mi formación como científico. Gracias también a Muon Systems por los medios y la motivación para realizar este proyecto.

Gracias por otra parte a mis padres, por guiarme siempre en lo personal y ofrecerme todo su apoyo, especialmente en los momentos más complicados. Muchas gracias también a Nayra, por ser como eres, por el cariño incondicional y por ayudarme a crecer y ser mejor como persona. No estaría aquí sin vosotros.

Por último gracias a todos mis compañeros y amigos del IFCA por hacerme sentir uno más desde el primer día. Y gracias también a mis amigos de Medina por estar siempre ahí y disfrutar juntos de los buenos momentos.

Contents

Chapter 1

Introduction

Muon tomography is a Non-Destructive Testing technique that consists in making use of the cosmic muons that reach the surface of the Earth to probe into different objects and structures. Muons are, like the electrons, a kind of elementary particles known as leptons. They are more massive than electrons, and thus have a higher penetrating power.

As cosmic muons go through different materials, they deflect and slow down due to ionization and multiple Coulomb scattering. In muon scattering tomography, we place two detectors at both sides of the object we want to inspect. With this, we can reconstruct muon trajectories and retrieve information of the Coulomb scattering, which can be used to generate three dimensional models of the object in study.

For this reason, muon scattering tomography provides a great testing technique, valid in big size and high density applications. The advantages of muon tomography make it an interesting tool for the industry.

Simulation data plays an important role in the development of muon tomography techniques. Being able to simulate the trajectories of the muons as they go through the detectors and the different objects we want to probe is crucial, for example, to develop efficient reconstruction algorithms and also to test and calibrate the equipment. The generation of realistic simulation data constitutes a large limiting factor for industrial applications of muon tomography.

Currently, the required simulation data is produced with specialized high energy physics simulation software. However, this simulation tools carry a high computational cost, and usually need a long time to obtain high quality simulation data. Also, this software requires the knowledge of the precise structure of the detection hardware, which in some occasions is not feasible, thus limiting the quality of the simulation.

This limitations may be overcome by the use of generative machine learning models. Specifically, in this work we will explore the use of Generative Adversarial Networks (GAN). GANs are a class of machine learning models based on deep neural networks that are capable (after proper training) of generating new synthetic data with the same characteristics as the training data.

GANs have been proven to be effective in many types of scenarios. The traditional case

of application is image generation. A well trained deep convolutional neural network can generate new original images resembling a certain dataset: faces, medical images, landscapes... Also, since GANs are very good at capturing details in probability distributions, they can be used for synthetic data generation. This is the case of this work, where we will explore the use of GANs to generate simulation data for muon tomography applications.

Generative Adversarial Networks have two main advantages as a simulation tool. On the one hand, the time needed for simulating muon data is reduced considerably; once a generative network is trained, the whole information about the process is captured in the weights of the network. Then, it acts as a simple function: it is fed random noise (latent variables) and outputs plausible simulated events. This computation is ultra fast, and allows us to obtain any amount of synthetic new data in almost no time.

The other main advantage is that we eliminate completely the process of 'building' the simulation. This means that all the characteristic features of the process we want to simulate can be captured by the generative network in the learning stage, even if we don't know in detail the structure of our hardware.

With all this, the motivation of this work is to test the application of Generative Adversarial Networks as a simulation tool for muon tomography industrial processes. We will train and evaluate the performance of GANs, and we will study their viability and range of application.

With respect to the structure of the manuscript, it consists of 6 chapters. Chapter 1 contains an introduction with the motivation of the work and the structure of the document.

The next two chapters are introductory to the two main topics at hand. These are intended to introduce readers who are less familiar with the topics and also to set the basic terminology. Muon tomography techniques are fully detailed in Chapter 2. This includes an introduction to the Standard Model of Particle Physics and cosmic muons, as well as an explanation of the principles behind the technique, the detection and reconstruction. Chapter 3 is also an introductory chapter dedicated to machine learning, generative modeling and Generative Adversarial Networks. I will also present there some specific GAN framework modifications relevant in this work (conditional GAN and WGAN).

The remaining part of the manuscript is devoted to detail the specific work performed and present and discuss the results obtained. In Chapter 4 I describe the two specific problems treated and their goals. Then I also describe the dataset, the best models found for each of the problems, and the key aspects involved in the process of optimizing the performance of the models (keep in mind that training machine learning models implies a long process of optimizing the models' parameters that is hard to reflect totally here). Then, the results obtained are presented and discussed in Chapter 5.

Finally, the conclusions are presented in Chapter 6, together with an insight into possible future work.

Chapter 2

Muon tomography

In this chapter I will give a brief introduction to muon tomography. Muon tomography is an non-destructive testing technique based on cosmic muons. I will explain the base concepts involved in this technique, as well as its advantages and limitations. Finally, I will present some real-world applications of muon tomography.

2.1 Particle Physics and the Standard Model

Particle Physics aims to understand the laws of nature through the study of the fundamental constituents of the Universe, the elementary particles, and the interactions between them, called forces. Currently, our whole understanding of subatomic world is contained in the Standard Model of Particle Physics (SM) [1]. This theory was developed in the decade of 1970, and gives a unified picture between the particles that constitute the ordinary matter and 3 out of the 4 interactions between them (gravity is not encompassed in the SM). In this unified picture, forces themselves are described by the exchange of particles.

The Standard Model provides a theoretical frame for many phenomena at subatomic level. Also, the theory reproduces experimental data with remarkable precision, making it one of the most accurate theories in Modern Physics.

Fundamental particles in the Standard Model are divided into two big groups: fermions and bosons. Fermions are the particles that make up the ordinary matter we see in the Universe, and have half-integer spin. They themselves are divided into leptons and quarks (Figure 2.1). The main difference between these two is that fermions are not affected by the strong force (QCD), while quarks have color charge and interact strongly.

Each of these groups contains three particle generations: first generation, formed by electron, electron neutrino and quarks up and down; second generation, formed by muon, muon neutrino and quarks strange and charm; and third generation, formed by tau, tau neutrino and quarks bottom and top.

	Leptons				Quarks			
	Particle		Q	mass/GeV	Particle		Q	mass/GeV
First	electron	(e^-)	−1	0.0005	down	(d)	−1/3	0.003
generation	neutrino	(ν_e)	0	$< 10^{-9}$	up	(u)	+2/3	0.005
Second	muon	(μ^-)	−1	0.106	strange	(s)	−1/3	0.1
generation	neutrino	(ν_μ)	0	$< 10^{-9}$	charm	(c)	+2/3	1.3
Third	tau	(τ^-)	−1	1.78	bottom	(b)	−1/3	4.5
generation	neutrino	(ν_τ)	0	$< 10^{-9}$	top	(t)	+2/3	174

Figure 2.1: The fundamental fermions in the Standard Model [2].

Bosons, on the other hand, are the particles that mediate the fundamental forces between fermions, and they have integer spin (Figure 2.2). In the Standard Model, each of the interactions is described by a Quantum Field Theory (QFT). In QFT, bosons arise naturally as mediators of the interactions. In the case of the electromagnetic interaction, the associated QFT is Quantum Electrodynamics (QED), and interactions between charged particles are mediated by virtual photons. On the other hand, the strong interaction is described by Quantum Chromodynamics (QCD) and color-charged particles exchange virtual non-massive gluons in the interaction. Finally, the weak interaction is mediated by the W^-, W^+ and Z bosons (massive).

Force	Strength	Boson		Spin	Mass/GeV
Strong	1	Gluon	g	1	0
Electromagnetism	10^{-3}	Photon	γ	1	0
Weak	10^{-8}	W boson	$W^\pm$	1	80.4
		Z boson	Z	1	91.2

Figure 2.2: The fundamental bosons in the Standard Model [2].

The Standard Model is completed with a new particle, the Higgs boson, which is responsible of the mass of the rest of the particles. The Higgs boson was discovered in 2012 at the Large Hadron Collider, and it has a mass of around 125 GeV [3].

2.2 Muons and cosmic rays

Muons are one of the second generation fermions described in the Standard Model. They have a mass of 105.7 MeV (about 200 times the mass of the electron). Like electrons, muons have spin one half, and can be positively or negatively charged. Also, muons have a life time of 2μs, and they typically decay into an electron (or positron) and a pair of neutrino-antineutrino:

$$\mu^- \longrightarrow e^- + \overline{\nu}_e + \nu_\mu \qquad\qquad \mu^+ \longrightarrow e^+ + \overline{\nu}_\mu + \nu_e$$

Muons that we detect at the surface of Earth are the product of the collisions of cosmic rays coming from outer space with the nuclei of the atmosphere. Cosmic rays are defined as beams of high energy particles that come into the Earth's atmosphere from outer space.

They are mainly constituted by protons (89%), but also contain nuclei of helium (10%) and heavier nuclei up to uranium. When this particles reach the Earth, they collide with the atoms in the upper atmosphere, creating a shower of particles, mainly pions. Pions belong to a group of particles called mesons, and consist of a first generation pair quark-antiquark. There are three types of pions: $\pi^+ : u\overline{d}$, $\pi^0 : u\overline{u}$ or $d\overline{d}$, and $\pi^- : \overline{u}d$.

Charged pions ($\pi^{\pm}$) have a mass of 139.6 MeV and have a life time of 0.26 ps [4], and they decay through the weak interaction. The main mode of decay is to a (anti)muon and a muon (anti)neutrino:

$$\pi^+ \longrightarrow \mu^+ + \nu_\mu \qquad\qquad \pi^- \longrightarrow \mu^- + \overline{\nu}_\mu \qquad\qquad (BR = 99.988\%)$$

This global process that takes place in the upper atmosphere results in a particle shower where the main final product is muons (Figure 2.3). The average flux of muons that reaches the surface of Earth is about 1 muon per cm^2 per minute, with a mean energy of 4 GeV.

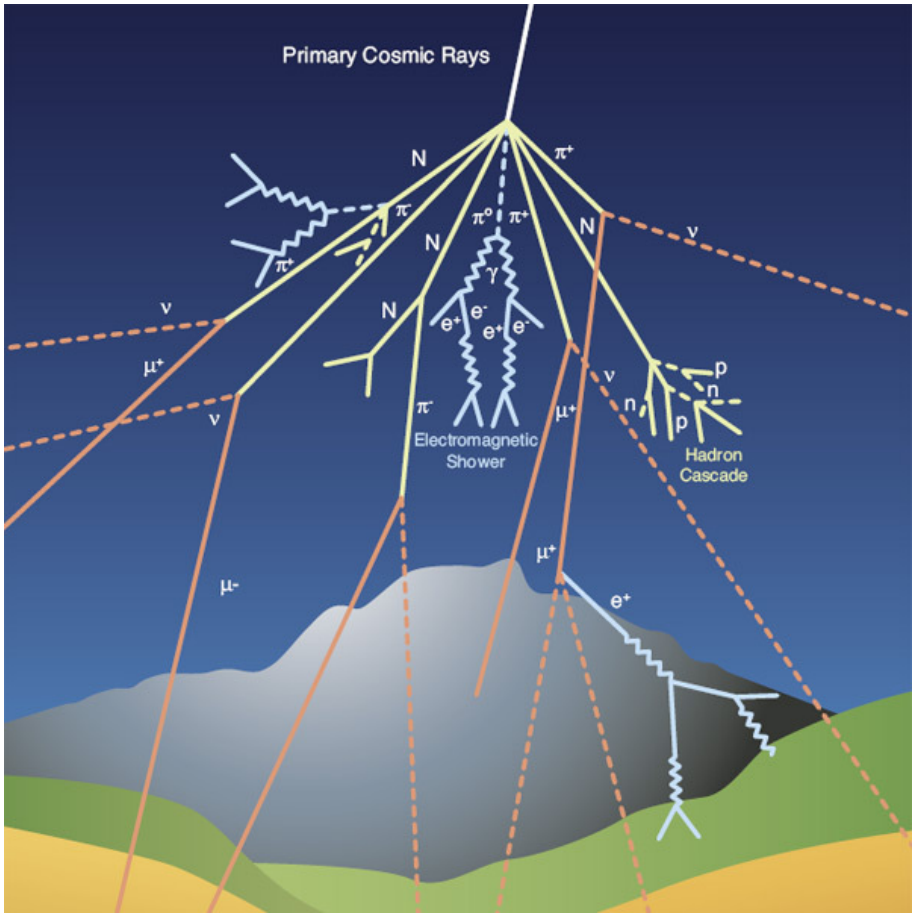

Figure 2.3: Particle shower resulting from cosmic rays.

2.3 Muon tomography: cosmic muons as a probing tool

Muon tomography, also called muography, is a Non-Destructive Testing (NDT) technique that employs cosmic muons as the main source of information. Muons interact with matter mainly through the ionization process (energy loss) and through Coulomb scattering. Both of them depend strongly on the composition, density and size of the material. This effects allow us to probe objects by measuring muons before and after the material.

There are two big groups of muography technologies, according to the principles behind the measurement of the muons. These are absorption muography and scattering muography. Absorption muography is performed by measuring the incident muon flux as a function of the direction of observation. It is based in the different transmittance of the materials that compose the object. Absorption muography is applied to large scale objects and requires only one detector, although long exposure times. On the other hand, scattering muography works by analyzing the scattering position and angular shift of the muons. Therefore, it requires at least two detectors, and not so long exposure times. It is used mainly for small to medium objects.

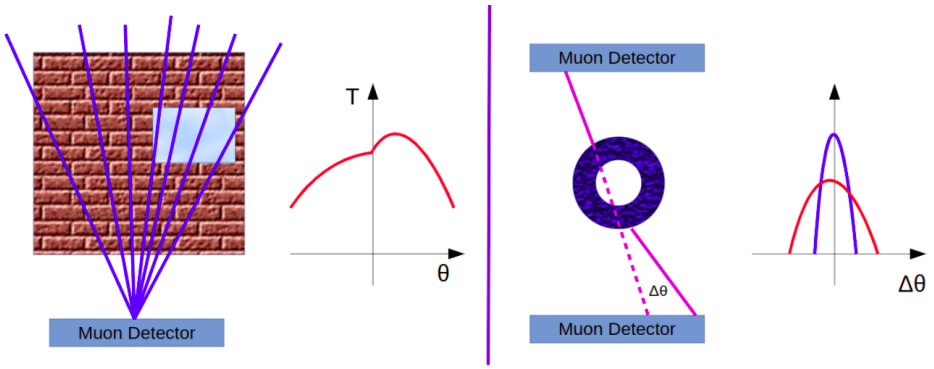

Figure 2.4: Absorption muography (left) and scattering muography (right).

There are numerous fields of application of this techniques. In particular, scattering muography is interesting in industrial processes [5]. It can be exploited to perform preventive maintenance and quality control inside the industry. The large penetrating power of muons and the possibility of operating the detectors without physical contact make the technique more convenient and efficient than other NDTs.

As a concrete example, scattering muography is being tested to estimate the wear of thermally insulated pipes. By placing detectors on top and below of a section of pipe one can generate a density map of the pipes. Then, feeding the image to a Convolutional Neural Network (CNN) we can estimate the radius of the pipe, and infer from that the quality of the pipe. This application of muon tomography is closely related to the problem studied in this work. The details of the process are explained later in Chapter 4.

Another example outside the industry is the monitoring of inaccessible places. In this sense, muon tomography can be used to make a density map of large scale structures like volcanoes [6] and pyramids [7]. These studies are performed with absorption muography.

With respect to the muon detection technology, there are multiple options, but probably

the detectors with the best ratio between cost and resolution and robustness are multiwire proportional chambers (MPCs). MPCs are radiation detectors specialized for charged particles that can provide spatial information about their trajectory. The specifications of the MPCs may vary depending on the application. For example, the detectors we have simulated for this work are those belonging to the company Muon Systems [8], which was interested in this project. Particularly, these detectors consist of an array of parallel wires at high voltage, which are inside a chamber with walls at ground voltage. The wires have a separation of 4 mm and are embedded in a mixture of argon and CO_2. Usually two layers are placed perpendicularly in order to measure X and Y coordinates. In Figure 2.5 we can see this particular set up, designed for industrial scattering muography. It has 3 MPC layers with a sensitive surface of 1 m^2.

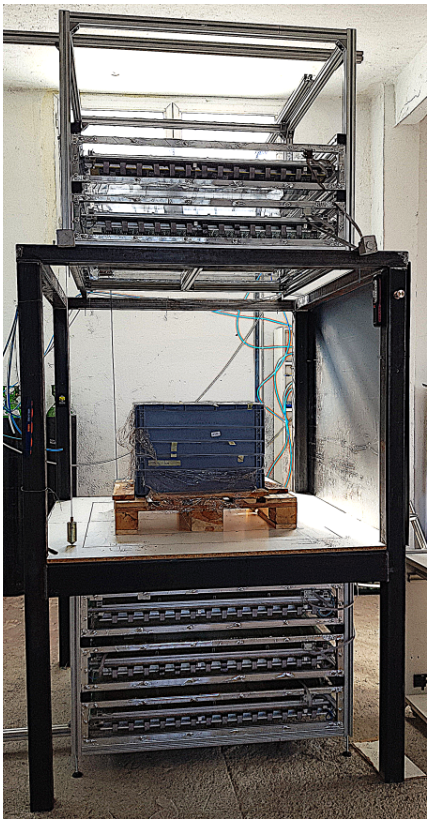

Figure 2.5: MPC-based muon detectors for scattering muography. Picture provided by Muon Systems [8].

In order to obtain valuable information from muon scattering tomography, we also need an effective reconstruction algorithm to obtain the scattering angles of the tracks. The most simple approach is the Point-of-closest-approach algorithm (POCA). This algorithm makes a simplistic assumption: the multiple scattering of a muon can be reduced to a single point of scattering, thus obtaining for each track a scattering angle and a point. It takes as input the incoming and outgoing tracks of the muon (assuming the electronics can associate both tracks). These tracks are likely to not be coplanar. Therefore, for each line, the closest point to the other line is computed using a simple algebraic formula. Then the middle point is taken as POCA, and is assumed the point where the scattering

is produced. Also from the lines we can obtain the scattering angle. In Figure 2.6 we can see an illustration of the principle behind this algorithm.

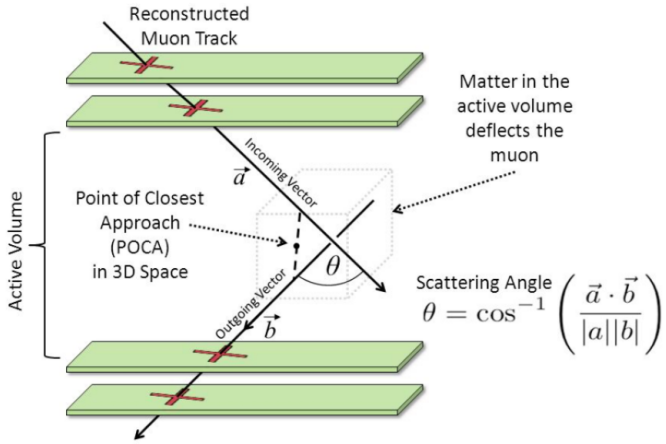

Figure 2.6: Point-of-closest-approach algorithm (POCA).

The POCA algorithm is useful in scenarios that are essentially empty with some high density chunks. However, in other situations it is too simplistic. There are other more complex algorithms based on maximum likelihood estimation (MLE) and machine learning. These depend critically on having large amounts of simulation data for parameter fitting (in the case of MLE) and training (in the case of machine learning). For this reason, this project is aimed towards exploring ways of obtaining the needed simulation data in an ultra fast way.

Chapter 3

Generative Adversarial Networks

Machine learning algorithms construct models that, by looking at a certain set of data, can perform tasks without being explicitly programmed to do so. This is currently a rapidly growing field that has proven to be very effective in diverse real-world applications. As explained in the introduction, machine learning algorithms are also at the heart of this work. Therefore, this Chapter is meant to serve a brief introduction to the field.

Here, I will explain in detail the main concepts behind machine learning, neural networks and, in particular, Generative Adversarial Networks.

3.1 Introduction to machine learning and neural networks

Machine learning is a branch of computer science which focuses on the development of algorithms that imitate the way that humans learn, by looking at data and gradually improving its accuracy. The term was first introduced in 1959.

Artificial Neural Networks (ANN) [9] are computational models whose structure tries to mimic the human brain. They consist of interconnected nodes called neurons. These neurons are typically structured in layers, and can transfer the information from one layer to the next. Each neuron receives an input signal from all the neurons in the previous layer and computes an output signal as a nonlinear function of the inputs.

Also, the interconnexions between the neurons have associated a weight which determines its importance inside the network. These weights are updated during the training process.

The computation of the neurons is called activation function, and it can be expressed as

$$a = \sigma(w^T x + b), \tag{3.1}$$

where x is the vector of inputs, w are the weights of the input connexions, b is a bias term,

and σ is a non linear function like the sigmoid or the ReLU and Leaky ReLU[1] function.

In Figure 3.1 we can see the basic structure of a Deep Neural Network (DNN), which is a type of ANN that contains several hidden layers.

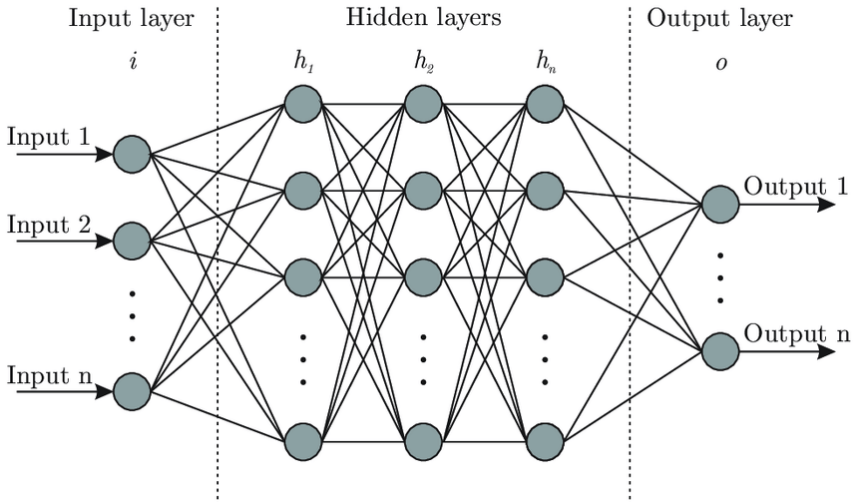

Figure 3.1: Structure of a DNN [10]

What makes neural networks so powerful is their ability to learn. There are two main modes of machine learning according to the training process: supervised and unsupervised learning.

Supervised learning models are trained on data that has been labelled for a particular purpose. The model then tries to learn the underlying patterns in the data to correctly map the input data (X) to the output label (y), in order to obtain accurate labelling results in never-before-seen data. Supervised learning is useful for two big groups of problems: classification and regression. These problems are also referred to as predictive learning, since the model has to make a prediction given some input data.

The learning task consists in looking at labelled data and make predictions (at first random) and then progressively adjust the weights until the labelling error is minimized. For this, we define a cost function, $f(y, \hat{y})$, which is an statistic that measures the error between the real and predicted labels $(y, \hat{y})$, giving an idea of how close the network predictions are to the real values. Therefore, the minimum of the cost function is identified with a network weight configuration that yields the best results. Some examples of cost functions are the mean square error or the binary crossentropy. Once we have the cost function, the network optimization is done in an iterative way by evaluating it on subsets of the entire dataset, called batches. At each iteration, the value of the cost function depends on the current weight configuration and it is used to compute the necessary update of the

[1]Leaky ReLU function:

$$f(x) = \begin{cases} x & x > 0 \\ \alpha x & x \leq 0 \end{cases}, \qquad \alpha \geq 0 \ \text{ (small)}.$$

The ReLU function is obtained for $\alpha = 0$.

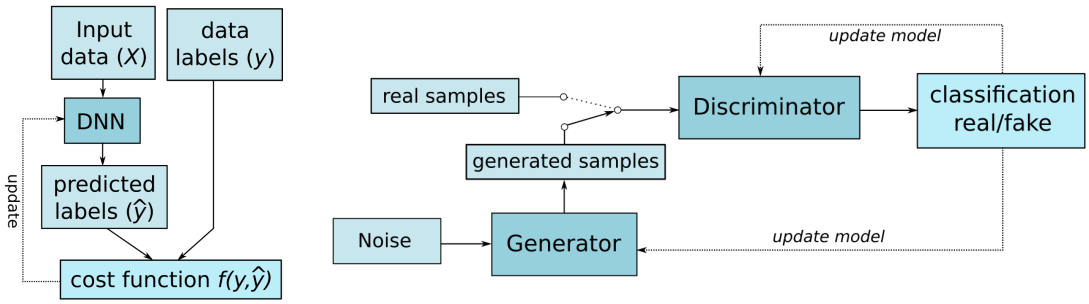

Figure 3.2: Supervised learning scheme (left). Example of GAN framework (right).

weights to minimize the cost, through an optimization algorithm like Stochastic Gradient Descent (SGD) or Adam [11]. Then, the weights are updated for the next iteration using backpropagation [12], an algorithm that efficiently updates the weights from the last layer to the first. The amount that the weights are updated in each iteration is called learning rate, and it is a crucial parameter: if it is too high, the network may skip the global minimum, and if it is too low, the training process may take too long to converge.

In supervised learning the training dataset is divided into two groups, the training set and the test set. The first one is sampled in batches and is used for the training of the network. The second one is reserved and used after the training process is complete to evaluate the quality of the network.

Every iteration of the complete training set is called epoch. Machine learning models are trained during several epochs, meaning that the process goes through the training dataset several times.

The second type of machine learning, unsupervised learning, does not make use of labelled data. Therefore, there is no iterative correction of the model as the model is not predicting anything. Instead, the goal of the model is to find underlying patterns in the input data, and make useful representations of these patterns. Some examples of unsupervised learning models are clustering and generative modeling.

In this work, we are treating the problem of generative modeling for muon tomography applications. As explained before, unsupervised models can summarize very well the distribution of input variables. Therefore, we can later use them to generate new examples that plausibly match the distributions of input data. This technique is known as generative modeling, and can be very useful to generate images with the same characteristics as other ones, or to generate new synthetic data. Generative Adversarial Networks are an example of generative models.

3.2 Generative Adversarial Networks

Generative Adversarial Networks (GAN) address to the problem of generative modeling using deep learning methods. They are a new framework of training generative models based on framing the problem as a supervised learning task. This is done by training two sub-models: a generator model that learns to synthesize new samples, and a discriminator

that has to tell apart real samples from the fake ones created by the generator. With this, we create a supervised learning framework by labeling the samples as either real or fake. The two models can then be trained together in a zero-sum game, eventually reaching an equilibrium where the generator produces plausible samples that fool the discriminator about half of the time.

This clever approach to generative modeling was introduced by Ian J. Goodfellow *et al.* in 2014 [13]. Since then, many improvements and modifications have been made to the original architecture, mainly to stabilize the training process and extend the framework to other problems. For example, Deep Convolutional GAN (DCGAN) [14] make use of convolutional neural networks and are widely used for image generation. Also, we have conditional GAN and Wasserstein GAN [15] frameworks, which I will detail later due to the application in this work.

But first, let's talk about how the generator and the discriminator work, and how they are trained together.

The generator is usually modelled by a deep neural network. It takes as input a fixed-length vector of random values and outputs samples in the problem domain. These input variables are called latent variables. They are typically gaussian numbers and have no intrinsic meaning, they are just used by the generator to seed the generation process. Therefore, by generating random noise we can produce new samples each time.

The discriminator model is a binary classifier that takes as input a sample in the problem domain, and predicts a binary class label (0 if the sample is real and 1 if it is generated). The real samples come from the training data, while the fake ones are given by the generator. Usually, the discriminator is also modelled by a deep neural network, although any classification model is suitable for the task. After the training, the discriminator is discarded since we are only interested in keeping the generator.

The training of the generator and discriminator models is done simultaneously, in an adversarial way. This means that the two models compete in a zero-sum game: when one wins the other one losses. As a typical example, we can think of the generator as a counterfeiter trying to make fake money, and the discriminator being the police. When the discriminator identifies the fake money, the generator is penalized with a great update of its parameters while the discriminator remains untouched. On the opposite situation, when the generator fools the discriminator, it is the discriminator the one that is penalized with an update in its parameters. In game theory terms, the discriminator D and generator G play a two-player minimax game with value function $V(D, G)$

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}}[\log D(x)] + \mathbb{E}_{x \sim p_z(z)}[\log(1 - D(G(z)))]. \tag{3.2}$$

In the original GAN paper [13], the authors present an algorithm to implement the minimax game between discriminator and generator, i.e., the training algorithm for Generative Adversarial Networks. As a remark, the authors also perform a theoretical analysis of adversarial networks. First they show that the minimax game has a global optimum where the generated probability distribution is equal to the data distribution. In addition, they also show that the proposed algorithm optimizes equation (3.2), giving the desired result.

This training procedure is detailed in Algorithm 1. Note that we alternate between k optimization steps for the discriminator and one step for the generator. This is done to keep D near its optimal performance during the training.

Algorithm 1 GAN training iteration. The "update_weights()" steps include computing the loss function, computing the optimal weight update and backpropagating the result.

```
for number of epochs do
  for number of batches do
    for k steps do
      latent_vectors ← sample_noise()              # Sample batch of latent vectors
      generated_samples ← generator(latent_vectors)
      real_samples ← sample_batch()                # Sample batch of real samples
      fake_labels ← discriminator(generated_samples)
      real_labels ← discriminator(real_samples)
      discriminator.update_weights(real_labels, fake_labels)
    end for
    latent_vectors ← sample_noise()                # Sample batch of latent vectors
    generated_samples ← generator(latent_vectors)
    fake_labels ← discriminator(generated_samples)
    generator.update_weights(fake_labels)
  end for
end for
```

3.2.1 Conditional GAN

There are some times when it is interesting to generate images or data conditioned to a certain class or subgroup of the training data. To do this, we can train our generative model by feeding it an additional vector of information in addition to the latent vector. This additional input can be a class value, such as dog or cat when generating animal images, or some more complex data, like an image when performing image translation.

To adapt this framework, we also need to condition the discriminator. This is done by providing it with the sample (real or fake) plus the additional input. It is important to note that the discriminator needs to receive consistent data. In the case of generated samples, the additional input fed to the discriminator has to match the one given to the generator beforehand. In Figure 3.3 we can see this extended architecture.

3.2.2 Wasserstein GAN

Wasserstein GAN (WGAN) is an improvement on the original framework that provides a more stable training process and also a new loss function that correlates to the quality of the generated samples. This extension was introduced in 2017 by Arjovsky et al. [15]. There is a quite dense mathematical motivation for this modification of the GAN architecture. However, here I will simply remark the most relevant changes and the aspects where they improve the model.

The idea behind the WGAN is that the generator should be trained to minimize the

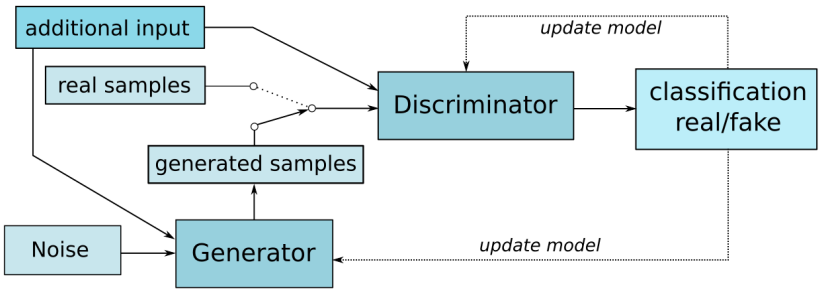

Figure 3.3: Example of conditional GAN framework.

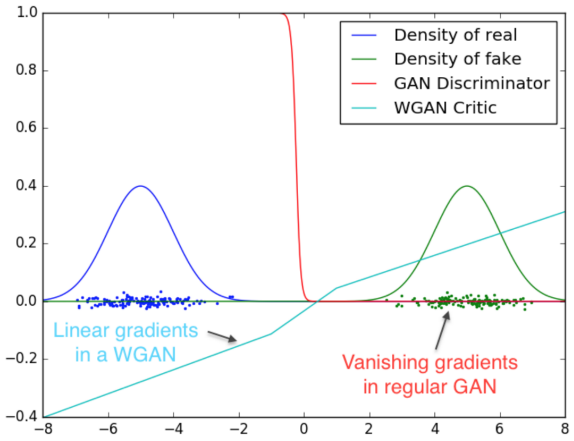

Figure 3.4: Generator gradients. WGAN has linear gradients everywhere, avoiding gradient vanishing.

distance between the real distribution of data and the generated one. For this purpose, the discriminator is replaced by a critic, a neural network that scores the realness of a given sample. The critic uses the Wasserstein distance as loss function, a distance that measures the minimum cost of transforming a data distribution q into a distribution p. The advantage of this cost function is that it provides smooth gradients everywhere, allowing the generator to learn even when the critic is optimized (Figure 3.4). This also makes the training less sensitive to hyperparameter choices.

Also, another great advantage of using the Wasserstein distance as loss function is that it is a direct measure of the quality of the generated images. This allows us to monitor the quality just by looking at the loss function plot.

However, there is an issue with the Wasserstein distance: it is highly intractable. There is a simplification of its calculation, but we have to assume that the critic is a 1-Lipschitz function. This is, the critic f satisfies

$$|f(x_1) - f(x_2)| \leq |x_1 - x_2| \qquad \forall x_1, x_2. \tag{3.3}$$

One way of enforcing this constraint is to clip the weights of the critic, i.e., enforcing $w \in [-c, c]$ for all weights w, where c is a small value like 0.01. But weight clipping will

diminish the learning abilities of the network, inducing instability and non-convergence. Another clever way of enforcing the Lipschitz constrain was proposed by Gulrajani et al. [16]. It is based on Theorem 1.

Theorem 1 *A differentiable function f is 1-Lipschitz if and only if it has gradients with norm at most 1 everywhere.*

With this result, we can enforce Lipschitz constraint by penalizing the critic if the gradient norm moves away from 1. This combined framework is known as Wasserstein GAN with Gradient Penalty (WGAN-GP).

For all of the advantages mentioned, the use of WGAN-GP has been generalized, and most of the up-to-date works with GANs use this extended framework (for example [17], [18]).

Chapter 4

Ultra fast simulation

This Chapter is dedicated to explain in detail the work that I've done in exploring the use of Generative Adversarial Networks in muon tomography applications. I will present the problem at hand, the dataset I have worked with and the goal of the experiments. Then I will go through the process of optimization, highlighting the key aspects that lead to the best results. Keep in mind that working with GANs (and machine learning models in general) usually entails a long process of optimization (architecture, hyperparameters...), that cannot be fully reflected here.

4.1 Problem description

As we mentioned before, in this work we consider a multiple scattering muon tomography configuration (Figure 4.1 left). This consists in two detectors placed at opposite sides of the object we want to study. Each of the detectors is capable of reconstructing the position and velocity of the incoming muons both in the X and Y directions. So, for each muon that goes through our object we measure 8 variables.

The goal is to train a generative network that can produce, in specific given scenarios, plausible muon events that match the probability distribution of real measurements.

Looking at the configuration of Figure 4.1 left, we notice that the first detector measures directly cosmic muons. The distribution of these muons is very well known, and in practice they are reasonably cheap to simulate, for example using programs like CRY [19]. With this, we can assert that the measurements of the second detector are the ones that we are more interested in generating, since they alone contain all the physics behind the multiple scattering process. So we will focus on training the model to generate the information provided by the second detector.

In particular, we will train the GANs to generate 4 modified variables, which are more convenient since they have a gaussian-like distribution centered in zero. The first two are the displacement in the trajectories of the muons in X and Y (Figure 4.1 right). And the other two variables are simply the difference in initial and final velocities of the muons, in X and Y. These 4 variables can be calculated from the original measurements as:

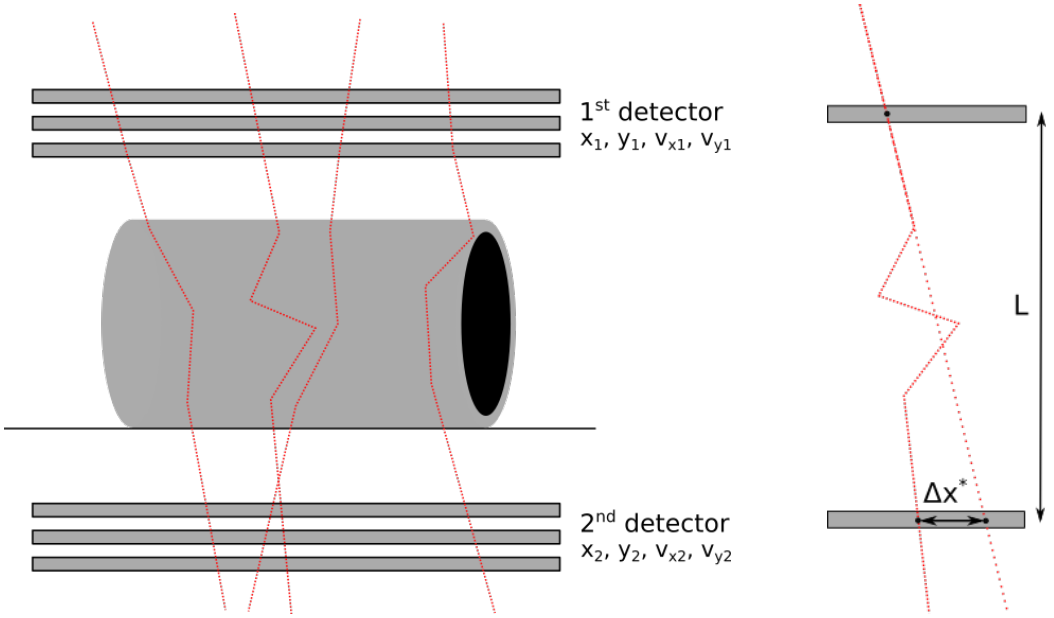

Figure 4.1: Multiple scattering tomography configuration (left). Scheme of variable Δx^* (right).

$$\begin{aligned} \Delta x^* &= x_2 - x_1 + L v_{x_1} \\ \Delta y^* &= y_2 - y_1 + L v_{y_1} \\ \Delta v_x &= v_{x_2} - v_{x_1} \\ \Delta v_y &= v_{y_2} - v_{y_1} \end{aligned} \tag{4.1}$$

In order to tackle this problem, we will use the Conditional GAN framework presented in Section 3.2.1, to also accommodate it for the use of first detector information. The specific architectures used will be explained in Section 4.2.

The analysis performed in this work is made up of two parts. First, we will study if it is possible to train a GAN to produce muon samples for a certain fixed pipe thickness value. This problem will be detailed in Section 4.2.1. In the second part, we will go further and train a GAN conditioned on the thickness value of the pipe. This means that we will use data from different pipe thickness to train the network in order to later produce samples tuning the input thickness. This will be detailed in section 4.2.2.

4.1.1 Dataset

Although the ultimate goal of this study is to work with real measurements, the data used in this experiment has been obtained through simulation, using Geant4 [20]. We have simulated a process similar to the one pictured in Figure 4.1, which consists in the propagation of cosmic muons through metal pipes of different thickness. Likewise, the interaction with the detectors has also been simulated. The result of the simulation gives us, for each muon, the exact value of the 8 relevant variables.

Each training sample represents a muon, and is made up of 9 values: the initial variables

of the muon, variables (4.1) and the thickness (r) of the pipe involved with that muon. For reference, these variables are

$$x_1, y_1, v_{x_1}, v_{y_1}, \Delta x^*, \Delta y^*, \Delta v_x, \Delta v_y, r \tag{4.2}$$

Figures 4.2 and 4.3 shows the distribution of the above variables. It should be noted that the variables in the first detector depend only on the natural distribution of cosmic muons. On the contrary, the variables obtained in the second detector depend on the variables measured in the first one and on the material in between the detectors.

For the first experiment, we have selected only one thickness value of the pipe, while for the second experiment we have used data from 8 different thickness. In Table 4.1 there is a summary of the data composition, with the number of samples used for training and evaluation of the models.

Simple GAN		
Pipe thickness (mm)	Number of training samples	Number of evaluation samples
16	306707	307352
Conditional GAN		
Pipe thickness (mm)	Number of training samples	Number of evaluation samples
4	619605	300000
6	618798	300000
8	617951	300000
10	616700	300000
14	614944	300000
16	615216	300000
18	614109	300000
20	613692	300000
12*	-	300000

Table 4.1: Dataset composition. *Only used to evaluate interpolation capabilities of the network.

In supervised learning it is common to preprocess the dataset, this is, to manipulate or transform the data before feeding it to the network, in order to enhance and stabilize the training process. Here, I have applied a transformation to standardize the variables by removing the mean and scaling to unit variance. This means, that for each of the variables in (4.2), except r, the values u_i $(i = 1, ..., N)$ are replaced with z_i calculated by

$$z_i = \frac{u_i - u}{s} \qquad (i = 1, ..., N), \tag{4.3}$$

where u and s are the weighted mean and weighted standard deviation, respectively, of the u_i $(i = 1, ..., N)$. The weights employed have been

$$w_i = \frac{1}{\sqrt{\Delta x_i^{*2} + \Delta y_i^{*2}}} \qquad (i = 1, ..., N). \tag{4.4}$$

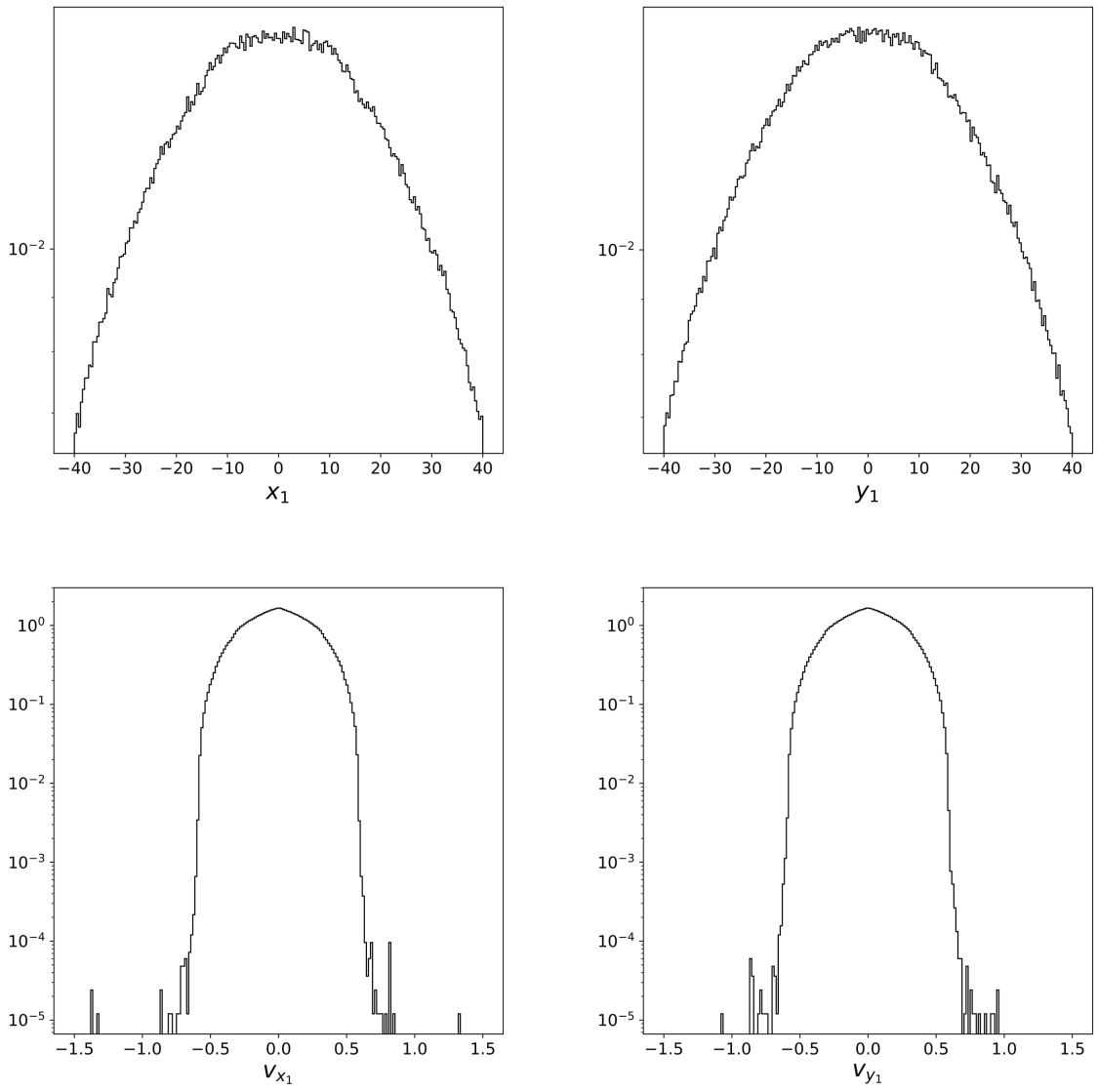

Figure 4.2: Distribution of kinematic variables measured by the first detector. Note that these variables do not depend on the thickness of the pipe.

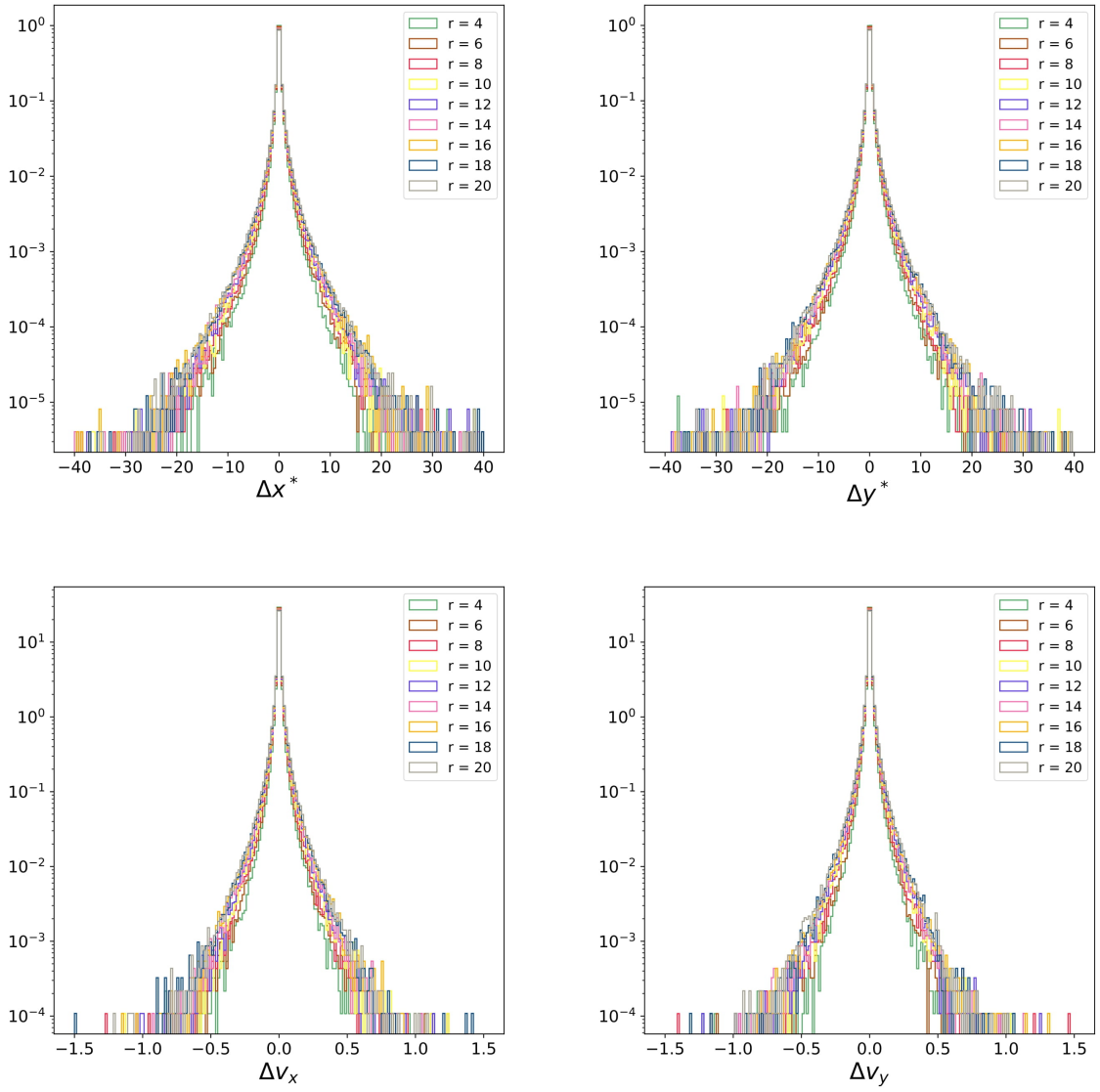

Figure 4.3: Distribution of variables (4.1). Note the dependence with the thickness of the pipe.

The use of this weights has been crucial to obtain good results that reproduce the tails of the desired distributions.

4.2 Experiments

In this section I will describe in detail the two experiments I have performed. This includes the specific objective of each one, the architecture of the GANs and the metrics and criteria to evaluate the performance of the resulting model. I will also remark some key aspects in the process of training and optimization of the models.

4.2.1 Simple GAN for muon propagation

The first problem consisted in training a GAN to generate variables (4.1). For this purpose, the generator takes two vectors as input: a vector of n_l random variables, and a vector of size 4 containing initial variables of the muon $(x_1, y_1, v_{x_1}, v_{y_1})$. On the other hand, the discriminator takes as input two vectors: one of size 4 containing the initial variables of the muon, and another vector of 4 values representing variables (4.1) (can be real samples or the generator output vector). Figure 4.4 shows this framework.

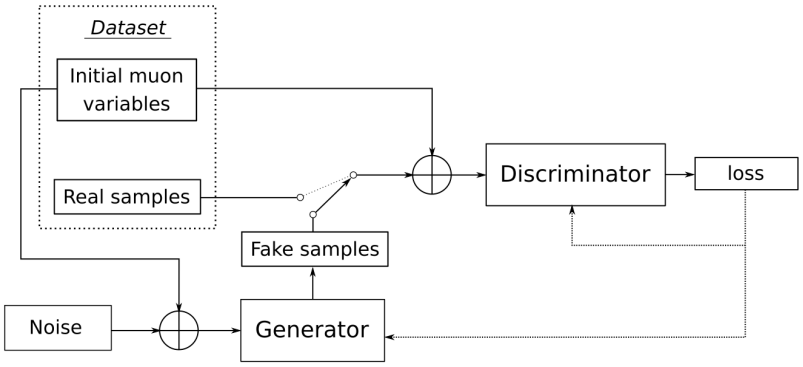

Figure 4.4: Scheme of the simple GAN framework.

The process of optimizing the parameters of this framework has been laborious. There has been a lot of fine tuning involved to get the best possible results.

One key aspect of this optimization process has been the dropout. Dropout is a standard regularization technique that consists in randomly ignoring some of the neurons in a layer at each training step. This makes the training process noisy, and works well to avoid overfitting[1]. Applying dropout is recommended in the GAN framework to stabilize the learning process and avoid converging to local minima. However, in my case I have experienced the exact opposite: removing the dropout layers from both discriminator and generator has been decisive to obtain acceptable results. Therefore, the final architecture of the model does not contain dropout.

[1]Overfitting refers to the condition of a machine learning model that works very well for a particular subset of data (usually the training data) but fails on the rest of the dataset. This is because the model learns features that are too specific from the training data, and not general to the whole dataset.

Other important aspect has been the use of an appropriate loss function. The original GAN framework [13] features a discriminator trained using the binary cross-entropy between the real labels and the predicted ones. This function is defined as

$$\frac{1}{N}\sum_{i=1}^{N} y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i) \tag{4.5}$$

where N is the total number of samples, and $y, \hat{y}$ are the real and predicted labels, respectively. The binary cross-entropy function has the limitation that it is only concerned with whether the predictions are correct or not, and not about how incorrect they are. In our case, using the binary crossentropy as loss function was not proving effective.

For this reason, I tried a modified framework, the Least Squares GAN (LSGAN) [21], [22]. This uses the Mean Squared Error (MSE) as loss function, defined as

$$\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2. \tag{4.6}$$

The MSE penalizes samples according to their distance to the boundary decision of the discriminator, thus providing more information to the generator about the "goodness" of the samples. In our case, LSGAN has worked way better than the original GAN framework.

The other parameters that needed to be tuned are: the optimizer and learning rate, the number of neurons in the hidden layers, and the latent space dimension.

The final model corresponding to the results in Chapter 5 is described now.

The discriminator has 4 hidden layers of 128, 64, 64, 64 nodes, activated with Leaky ReLU functions, and it outputs a single linear activated value. The generator has 6 hidden layers with 512, 256, 256, 128, 64, 16 nodes, activated with Leaky ReLU functions, and outputs 4 linear activated values. The dimension of the latent space is 64. For training, I have used Adam optimizer with a initial learning rate of 0.001 that halves every 50 epochs. This decrease of the learning rate is done to reach more accuracy in the later stages of the training process. The training batches had 1024 samples, and the model was trained during 200 epochs. Finally, in order to have an optimized discriminator during the whole training, I have updated the discriminator 5 times for each generator update.

4.2.2 Conditional GAN for muon propagation with interpolation capabilities

This second experiment is an extension of the previous one. The objective is to train a generative adversarial network that is able to disentangle the muons that traverse through different pipes, and can afterwards generate muon data conditioned to a given thickness.

In order to do this, the generator and discriminator have to receive as input a label indicating the thickness of the pipe that corresponds to a certain muon. This labels are given to the network as one-hot encoded vectors.

One-hot encoding is a way of representing categorical data in which we have a vector of size n (total number of categories) with all values equal to 0 except for the category we want to represent. This representation is widely used in machine learning for treating discrete data. In Figure 4.5 we can see an example of one-hot encoding.

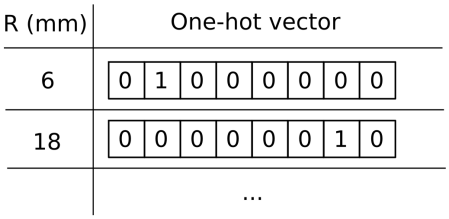

Figure 4.5: Example of one-hot encoding.

In our case, we have $n = 8$ different thickness labels. With this, the generator will take as input the latent variables, plus a vector of size 12 containing the 4 initial variables of the muons and the one-hot label. As well, the discriminator input will include the same size 12 vector, together with variables (4.1) (real or fake).

With respect to the specific GAN architecture, I first considered using the same LSGAN approach as before, since it can be adapted simply by adding the thickness label to the input of the networks. However, the results obtained were not good enough. After exploring numerous possibilities, the best models I found were using the Wasserstein GAN with Gradient Penalty framework (WGAN-GP), which I explained in detail in Section 3.2.2. Briefly resuming, WGAN-GP replaces the use of a discriminator with a critic, which gives a score to the samples it receives (the higher the score, the more real they are). Figure 4.6 shows this framework.

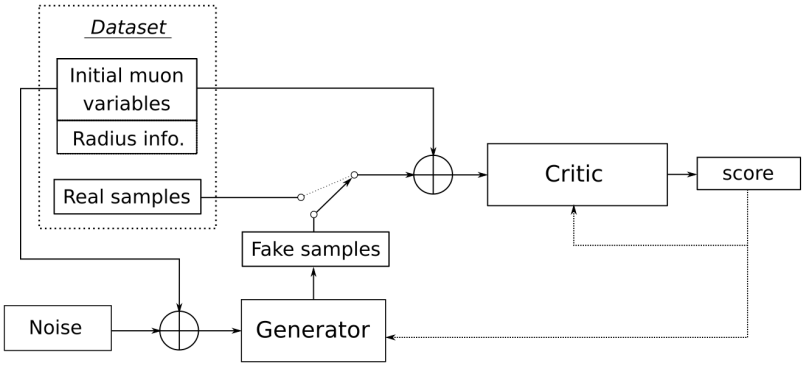

Figure 4.6: Scheme of the WGAN-GP framework.

Again, I have followed a process of optimization of the parameters involved in the training. Mainly it has been try and error of different architectures and learning rates until I obtained good enough results. The final model is described now.

The critic has 3 hidden layers of 128, 64, 32 nodes, activated with Leaky ReLU functions, and it outputs a single linear activated value. The generator has also 3 hidden layers with 32, 64, 128 nodes, activated with Leaky ReLU functions, and outputs 4 linear activated values. The dimension of the latent space is 16. For training, I have used Adam optimizer with a fixed learning rate of 0.0001. The training batches had 5000 samples, and the model was trained during 1000 epochs. Finally, in order to have an optimized discriminator during the whole training, I have updated the discriminator 5 times for each generator update.

The performance of the generative network will be evaluated in two ways. First, we will review the performance in generating muon data for the thickness labels used in the training. Then, we will also test the interpolation capabilities by generating samples corresponding to $r = 12$ mm, for which the GAN was not trained.

To generate this interpolated data, we need to feed the generator with a one-hot label representing a thickness of 12 mm. This is done by taking the intermediate vector between $r = 10$ and $r = 14$.

$$\begin{aligned} r = 10 &\longrightarrow [0\ 0\ 0\ 1\ 0\ 0\ 0\ 0] \\ r = 12 &\longrightarrow [0\ 0\ 0\ 1\ 1\ 0\ 0\ 0] \cdot \frac{1}{2} \\ r = 14 &\longrightarrow [0\ 0\ 0\ 0\ 1\ 0\ 0\ 0] \end{aligned}$$

Chapter 5

Results and Discussion

In this Chapter I will present and discuss the most relevant results from experiments described in Section 4.2. The results presented here correspond to the best model found for each of the two experiments, i.e., the ones detailed in Sections 4.2.1 and 4.2.2.

The evaluation of the generative models is done based on two types of results: visual and numerical results. Visual results include histograms plotting the distribution of the generated variables, as well as some correlations between the variables. Numerical results include statistical parameters of the distributions: mean, covariance matrices and skewness.

The mean of a certain variable is simply defined as the sum of the samples divided by the total number of samples:

$$\overline{x} = \frac{1}{N}\sum_{i=1}^{N} x_i. \tag{5.1}$$

The covariance matrix is a square matrix C that contains in each entry c_{jk} the covariance between variables j and k. The covariance gives a measure of the joint variability of two variables, which in our case is very interesting to evaluate the sample-to-sample generation capabilities of the networks; if the covariances between real and generated distributions match, we can infer that the generation of each individual sample is consistent.

The sample covariance between variables x_j and x_k having N samples of each variable is defined as

$$C = (c_{jk}) = (\mathrm{cov}(x_j, x_k)), \qquad \mathrm{cov}(x_j, x_k) = \frac{1}{N-1}\sum_{i=1}^{N}(x_{ij} - \overline{x_j})(x_{ik} - \overline{x_k}). \tag{5.2}$$

From equation (5.2) note that $\mathrm{cov}(x_j, x_k) = \mathrm{cov}(x_k, x_j)$ for all j, k. So the covariance matrix is always symmetric. Also note that $\mathrm{cov}(x_j, x_j) = \sigma^2(x_j)$, meaning that the elements in the diagonal of C are the square of the standard deviation of each variable.

In the last place we have the skewness. The skewness is a variable that measures the degree of asymmetry of a certain distribution about its mean. The sample skewness of a set of N values from variable x is computed as

$$g_1 = \frac{m_3}{m_2^{3/2}}, \tag{5.3}$$

where m_j is the biased jth central moment

$$m_j = \frac{1}{N}\sum_{i=1}^{N}(x_i - \overline{x})^j. \tag{5.4}$$

As a remark, keep in mind that there are plenty of evaluation metrics and tests to compare variable distributions. But in our case we consider that the parameters described above are enough to evaluate the quality of the results.

5.1 Simple GAN for muon propagation

In Figure 5.1 we can see the distributions of the generated variables compared to the real distributions, for a simulation of a pipe with thickness 16 mm. We can observe that the generator is able to produce samples that match very well the real probability distributions. Next, in Figure 5.2 we can see two 2-D histograms that plot the correlation between some of the variables (the two strongly correlated variables). We can observe that in both cases the 2-D distributions match correctly, only observing some little discrepancies in the extreme points, mainly due to the randomness of the generation process.

Summing up, we can conclude from the visual results that the generator is capable of producing samples that match the original dataset with remarkable accuracy. Also, the correlations are reproduced very well, meaning that the generation sample-to-sample is good and the 4 variables are generated in a consistent way.

Moving on to the numerical parameters, Table 5.1 compares the mean and skewness of the real and generated distributions. We can observe that, in accordance with the plots, the means of the distributions are very close to 0, both for real and generated samples. About the skewness, we can observe that it is close to 0, indicating that the distributions are mostly symmetric. Again, the skewness is in the same order of magnitude for real and generated samples. This tells us that the distributions have similar degree of symmetry.

Note that we wouldn't want the parameters presented to match exactly to the real dataset, even with an ideal generator. This is because the objective is to generate new original samples never seen before, seeding the process with random noise.

Table 5.2 shows the covariance matrices of the real and generated samples. First, let's look at the elements in the diagonal, which represent the variances of the 4 variables. We can see that there is a good agreement between both matrices, suggesting that the model is successfully generating distributions with the correct variance. Looking outside the diagonal, these elements measure the correlations between the variables. We can see

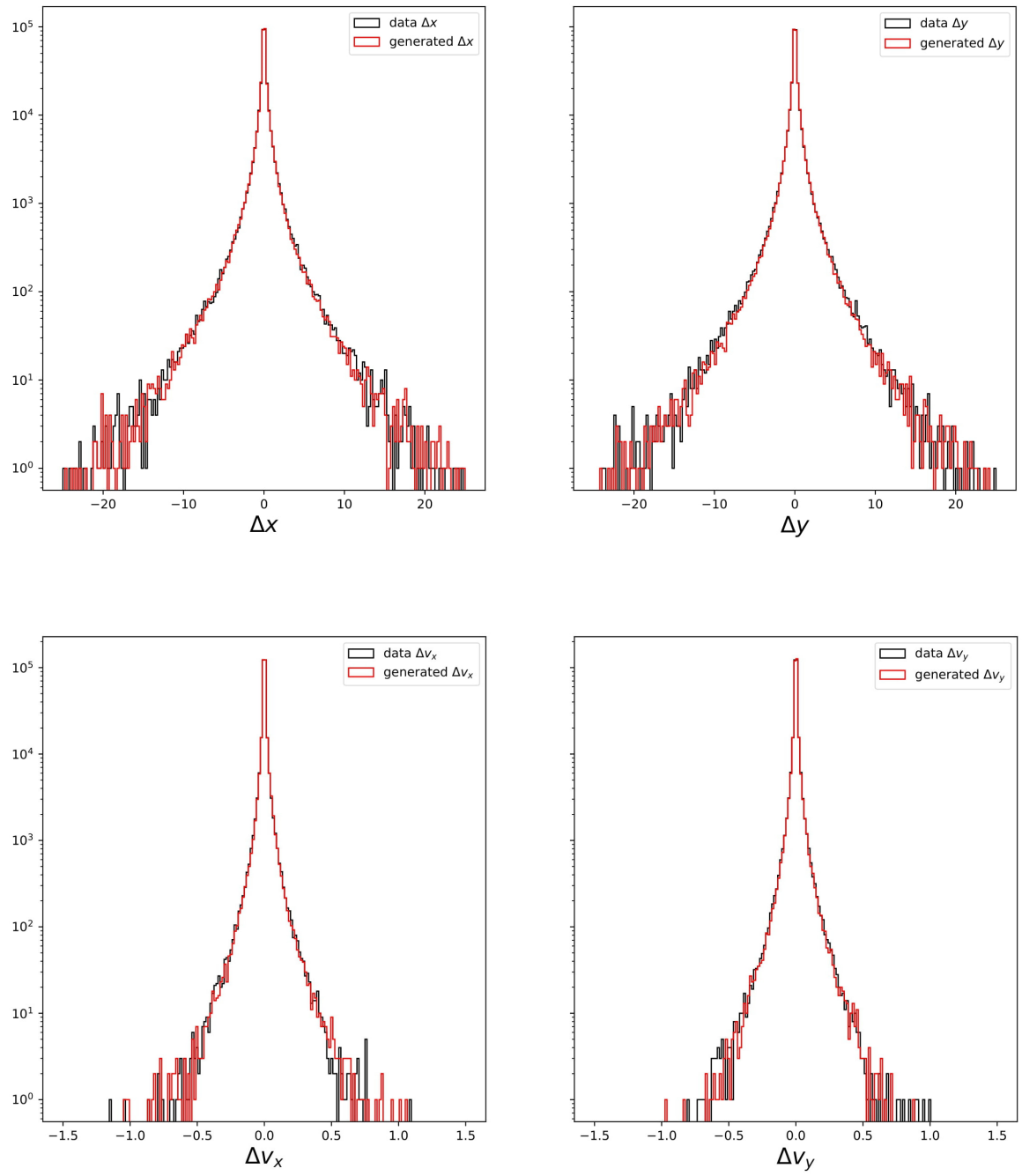

Figure 5.1: Distributions of real (black) and generated (red) variables (4.1) corresponding to a simulation of a pipe with wall thickness 16 mm.

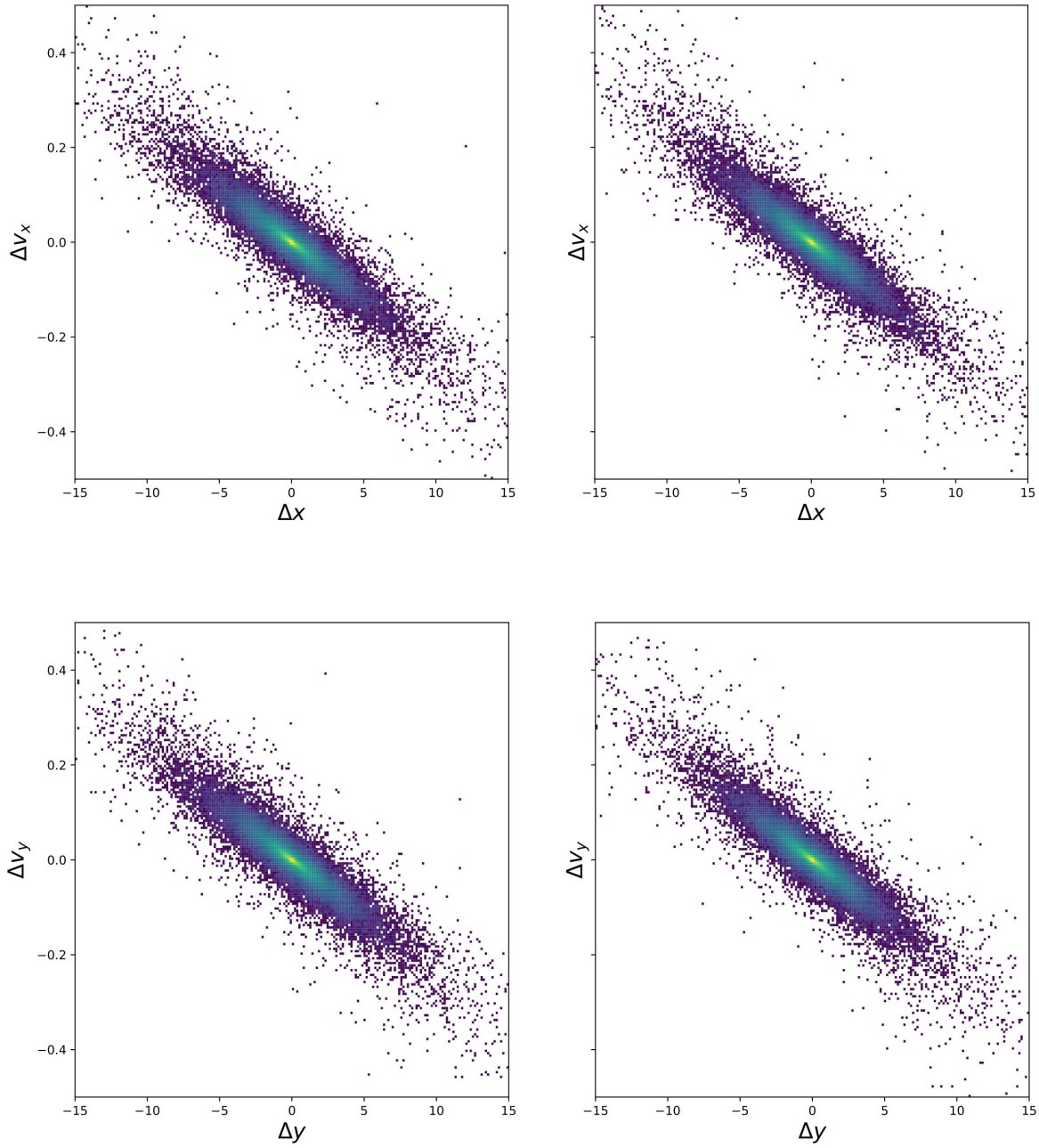

Figure 5.2: 2-D histograms that represent the correlations between Δx^*-Δv_x and Δy^*-Δv_y. The plots on the left correspond to real samples and the ones on the right to generated samples. Again, this is for a simulation of a pipe of wall thickness 16 mm.

that the strongest correlations are between variables Δx^*-Δv_x, Δy^*-Δv_y, Δx^*-Δy^*, while the rest are essentially zero. Again, the network is reproducing correctly the covariance between these variables. This is remarkable, and gives more proof about the generation process: the 4-D distribution is being generated in a consistent way, reproducing all the relations between the individual variables.

	Variable	Real samples	Generated samples
Mean	Δx^*	$5.9 \cdot 10^{-5}$	$-6.0 \cdot 10^{-3}$
	Δy^*	$-1.2 \cdot 10^{-3}$	$4.2 \cdot 10^{-3}$
	Δv_x	$3.4 \cdot 10^{-5}$	$2.3 \cdot 10^{-4}$
	Δv_y	$1.5 \cdot 10^{-5}$	$-8.0 \cdot 10^{-5}$
Skewness	Δx^*	1.10	0.55
	Δy^*	0.10	0.60
	Δv_x	−0.05	0.56
	Δv_y	0.24	−0.66

Table 5.1: Mean and skewness.

Real samples

	Δx^*	Δy^*	Δv_x	Δv_y
Δx^*	1.71	$-6.98 \cdot 10^{-2}$	$-3.60 \cdot 10^{-2}$	$6.38 \cdot 10^{-4}$
Δy^*		1.70	$9.18 \cdot 10^{-4}$	$-3.43 \cdot 10^{-2}$
Δv_x			$9.82 \cdot 10^{-4}$	$-9.60 \cdot 10^{-6}$
Δv_y				$9.34 \cdot 10^{-4}$

Generated samples

	Δx^*	Δy^*	Δv_x	Δv_y
Δx^*	1.48	$-0.34 \cdot 10^{-2}$	$-3.47 \cdot 10^{-2}$	$1.28 \cdot 10^{-4}$
Δy^*		1.35	$0.63 \cdot 10^{-4}$	$-3.14 \cdot 10^{-2}$
Δv_x			$9.65 \cdot 10^{-4}$	$-3.90 \cdot 10^{-6}$
Δv_y				$8.74 \cdot 10^{-4}$

Table 5.2: Covariance matrices of real and generated samples.

With this results we can determine that the model is able to reproduce very well the multiple scattering process at hand, and can generate plausible samples that follow the real probability distributions.

Additionally, we have compared the simulation speed between this model and the Geant4 software used to generate the training data. When generating 10000 events, Geant4 takes about 37 s, while the GAN model takes around 0.7 s. This constitutes a decrease in the computation speed of a factor of about 50. In other words, simulations that take a whole day can be done now in about 27 minutes.

5.2 Conditional GAN for muon propagation with interpolation capabilities

We have already seen that it is possible to train a GAN to produce good samples that simulate a multiple scattering process through a metal pipe. Now, as we explained in the previous Chapter, we want to go a little bit further. In this sense, we want to obtain a generative model capable of producing muon samples scattered through metal pipes of different thickness. Then, we will evaluate the interpolation capabilities of the model, this is, we will ask the model for samples corresponding to a thickness value it never saw and check if the generated samples are good. The pipe thickness values involved in the training have been 4, 6, 8, 10, 14, 16, 18 and 20 mm.

First, as a way to check that the model has correctly learned, we have produced samples corresponding to pipes with thickness value contained in the training dataset. The results of this are shown in Figure 5.3. In this plots appear the real and generated distributions corresponding to the largest and smallest thickness. The difference between said distributions is that the greater the thickness, the wider the distribution. As we can see, the generator is capable to distinguish between both thickness values and tune the generated samples according to this condition. This is a great result, because it implies that we can use the same generator model for different case scenarios just by tuning one input parameter.

Then, let's see if the model is able to interpolate well. In Figure 5.4 we can see the resulting distributions for a thickness of 12 mm, which was not in the training dataset. This new samples have been generated by feeding the network with an intermediate label between 10 and 14 mm. In the plots we can also see the distributions corresponding to these thickness values (pink). It can be observed (although quite faintly due to the similarity of the distributions) that the samples generated for 12 mm thickness lie in between the ones of 10 and 14 mm, which is precisely the effect observed in real data. Therefore, we can affirm that the model is able to interpolate between thickness labels, and tune the generation even to a thickness value that it never learned.

Finally, to assess the quality of the interpolated samples, I will present some numerical results corresponding to the distribution of 12 mm samples. In Table 5.3 we show the values of mean and skewness of the distributions. With respect to the mean, both real and generated data have means very close to zero, so we can say the agreement in that sense is good. Following with the skewness, we observe that the real variables Δx^* and Δy^* have a certain asymmetry, while the other two are fairly symmetric. However, the skewness of the all the generated distributions are close to zero, indicating that they are symmetric. Nevertheless, the differences are not relevant, and the results tell us that the model tends to generate symmetric distributions.

Then, Table 5.4 shows the covariance matrices of the real and generated samples. Looking at the diagonal, we can see that the variances are in good agreement for real and generated data. This indicates that the dispersion of the distributions is quite similar. Outside the diagonal, we find the covariances between the variables. We observe that the strongest correlations are, as before, between Δx^*-Δy^*, Δx^*-Δv_x, Δy^*-Δv_y. In the case of the first one, we see a notable difference, and thus the generator is showing a deficiency reproducing

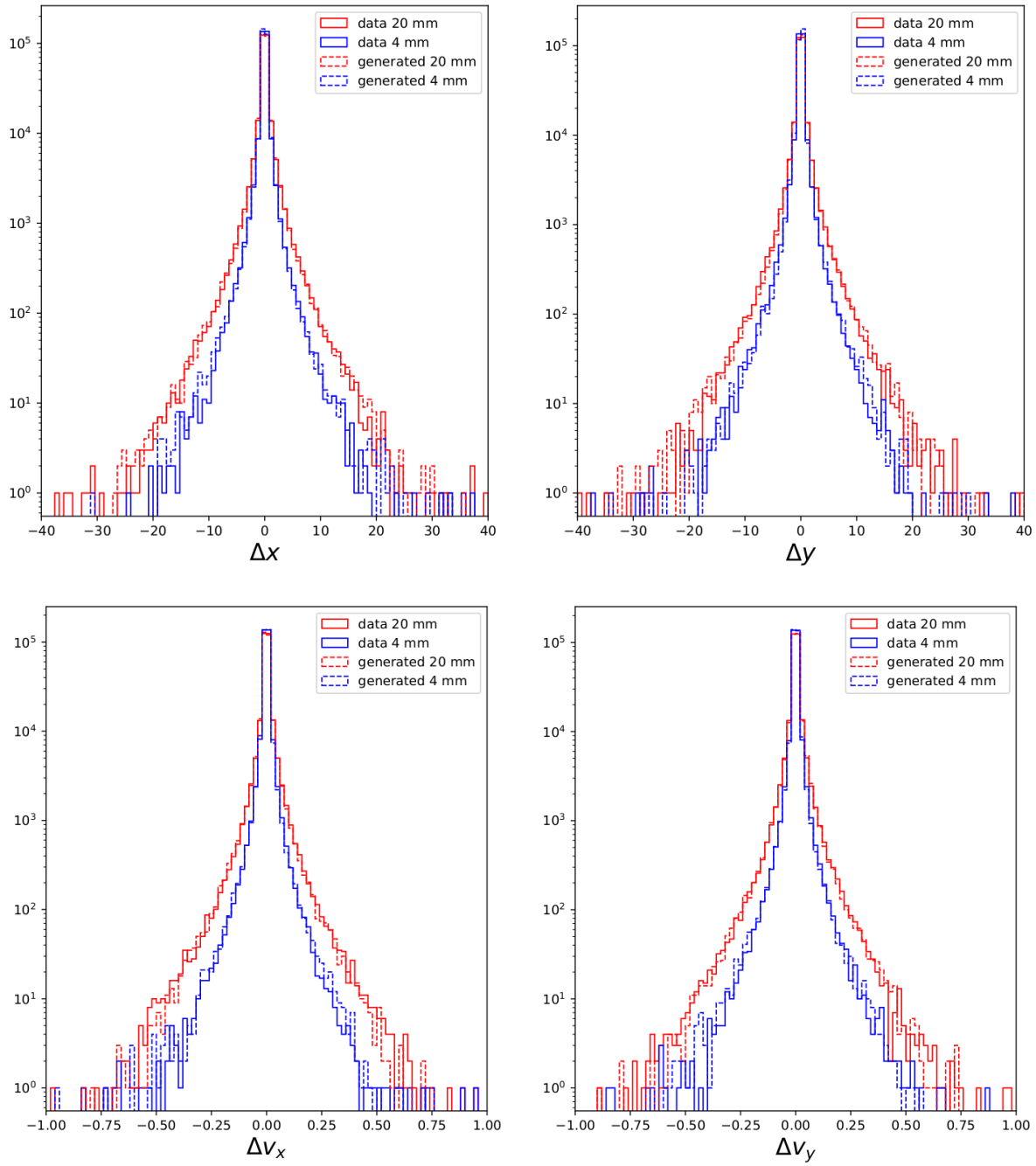

Figure 5.3: Variable distributions of real and generated samples, for two different values of the pipe thickness.

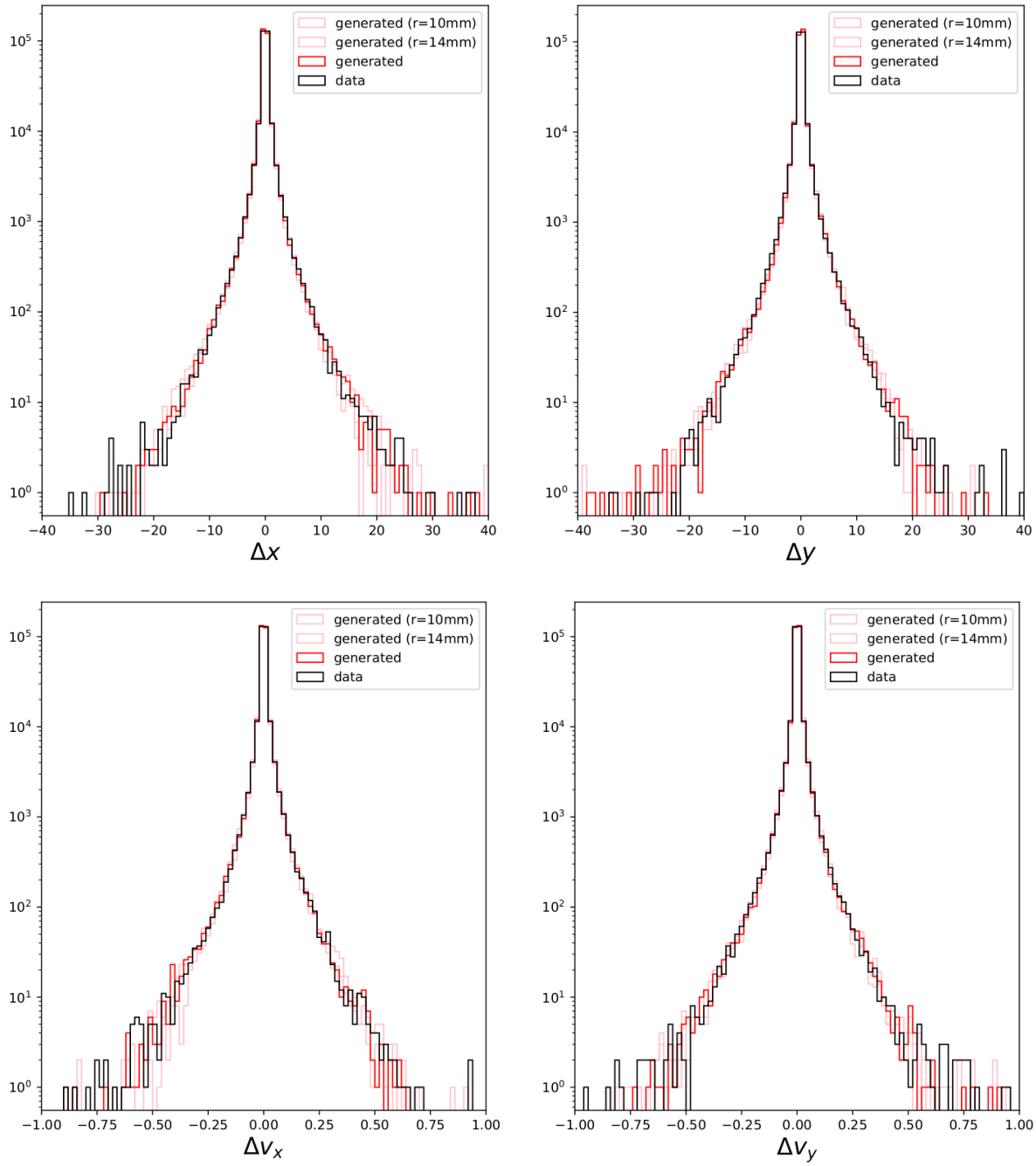

Figure 5.4: Real and generated distributions corresponding to interpolated samples ($r = 12$ mm). For visualization, in pink appear the distributions for 10 and 14 mm, used to perform the interpolation.

that relation. However, it is still in the same order of magnitude. Looking at the other two important correlations, we see that these are very well reproduced by the model. Again, this circumstance gives us proof that the generation is being well done individually sample by sample, meaning that all the properties of the joint 4D distribution can be reproduced in a consistent way.

	Variable	Real samples	Generated samples
Mean	Δx^*	$-6.1 \cdot 10^{-4}$	$-1.4 \cdot 10^{-2}$
	Δy^*	$2.7 \cdot 10^{-3}$	$1.8 \cdot 10^{-2}$
	Δv_x	$2.5 \cdot 10^{-5}$	$-2.2 \cdot 10^{-4}$
	Δv_y	$-3.9 \cdot 10^{-5}$	$-7.0 \cdot 10^{-6}$
Skewness	Δx^*	-2.84	0.76
	Δy^*	4.82	-0.34
	Δv_x	0.01	-0.70
	Δv_y	0.13	0.02

Table 5.3: Mean and skewness results for $r = 12$ mm data.

Real samples

	Δx^*	Δy^*	Δv_x	Δv_y
Δx^*	1.55	$4.29 \cdot 10^{-2}$	$-2.98 \cdot 10^{-2}$	$5.67 \cdot 10^{-4}$
Δy^*		1.52	$-4.52 \cdot 10^{-4}$	$-3.01 \cdot 10^{-2}$
Δv_x			$8.02 \cdot 10^{-4}$	$3.60 \cdot 10^{-6}$
Δv_y				$8.01 \cdot 10^{-4}$

Generated samples

	Δx^*	Δy^*	Δv_x	Δv_y
Δx^*	1.23	$-0.78 \cdot 10^{-2}$	$-2.79 \cdot 10^{-2}$	$1.12 \cdot 10^{-4}$
Δy^*		1.23	$0.65 \cdot 10^{-4}$	$-2.76 \cdot 10^{-2}$
Δv_x			$7.75 \cdot 10^{-4}$	$-1.90 \cdot 10^{-6}$
Δv_y				$7.50 \cdot 10^{-4}$

Table 5.4: Covariance matrices of real and generated samples for $r = 12$ mm data.

As a summary of this second experiment, we can confidently say that it is possible to train a conditional GAN to produce samples conditioned to a given input thickness label. Also, we can assert that the model has interpolation properties. This means that it is possible to generate data corresponding to certain thickness value that the network has never learned.

Chapter 6

Remarks and Conclusions

In this work we have explored the application of Generative Adversarial Networks for muon scattering tomography data simulation. Precisely, the goal of the study has been to analyze whether GANs are able to produce ultra fast simulation data with the same characteristics as the data obtained with classic simulation software, which is computationally very expensive.

In this sense, we have attacked two different problems. First, we have trained a GAN to produce data corresponding to muons scattered through a metal pipe. In this first case, we have determined, by looking at the generated variable distributions and correlations among them, that the model is capable to produce plausible muon data that resembles the original simulation.

For the second part, we have increased the level of complexity to test further capabilities of GANs. In this case, we have trained a model with data from muon scattering through metal pipes of different thickness values, with the objective of having a model that can be tuned with a parameter (thickness) to produce modulated samples. We have seen that this is indeed possible, that the generator produces plausible samples for each thickness value. Also, we have tested the interpolation capabilities of the network, observing that it also produces good data for never-seen-before thickness values that lie between the training labels.

With this study it has been demonstrated that generative models based on machine learning, particularly Generative Adversarial Networks, constitute a very powerful tool to obtain ultra fast simulation for muon tomography applications. The speed increase achieved with respect to traditional simulation software is a factor of about 50, meaning that with the same computational cost, we can obtain around 50 times more simulation data. Also, the framework for training GANs is very flexible, allowing us to obtain, for example, user-modulated data with the use of additional labels.

6.1 Future work

The work presented here is a good introduction to the use of GANs in muon tomography simulation, and the results obtained are very good by themselves. However, there are a

few other extensions that are worth remarking here, for future work reference.

First, note that the training data involved in this study comes from Geant4 simulation software. But the final goal is to replace traditional simulation software with a properly trained model. For this reason, the next step would be to use training data obtained from real measurements.

But to do this we would need to take into account that the real data is slightly different to the simulation currently being used. This is because the detection process is complex. As we have said, the detectors consist of MPC layers, and these are made of an array of wires. This means that in real measurements we do not obtain the exact position and velocities of the muons, we just have, for each wire, whether it was activated or not. This measurement process yields a discrete measurement structure from where we can reconstruct the muon trajectories with some uncertainty.

To include the detection mechanism, we would have to modify the GAN framework to accept these new data structures, something that remains for future development.

Bibliography

[1] S. Weinberg, "A model of leptons", Phys. Rev. Lett. **19**, 1264–1266 (1967).

[2] M. Thomson, "Modern Particle Physics", in (Cambridge University Press, 2013) Chap. 1.

[3] ATLAS Collaboration, "Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC", Physics Letters B **716**, 1–29 (2012).

[4] R. Workman et al. (Particle Data Group), "Review of Particle Physics", to be published (2022).

[5] P. Martinez Ruiz del Arbol, A. Ocio Alonso, C. Diez, and P. Gomez Garcia, "Applications of Muography to the Industrial Sector", Journal of Advanced Instrumentation in Science **267** (2022).

[6] R. D'Alessandro, F. Ambrosino, G. Baccani, L. Bonechi, M. Bongi, A. Caputo, R. Ciaranfi, L. Cimmino, V. Ciulli, M. D'Errico, F. Giudicepietro, S. Gonzi, G. Macedonio, V. Masone, B. Melon, N. Mori, P. Noli, M. Orazi, P. Passeggio, and L. Viliani, "Volcanoes in italy and the role of muon radiography", Philosophical Transactions of The Royal Society A Mathematical Physical and Engineering Sciences **377**, 20180050 (2019).

[7] L. W. Alvarez, J. A. Anderson, F. E. Bedwei, J. Burkhard, A. Fakhry, A. Girgis, A. Goneid, F. Hassan, D. Iverson, G. Lynch, Z. Miligy, A. H. Moussa, M. Sharkawi, and L. Yazolino, "Search for hidden chambers in the pyramids", Science **167**, 832–839 (1970).

[8] *Muon Systems*, `https://muon.systems` (visited on 07/15/2022).

[9] I. C. Education, *Neural Networks*, (Aug. 2020) `https://www.ibm.com/cloud/learn/neural-networks` (visited on 06/29/2022).

[10] F. Bre, J. Gimenez, and V. Fachinotti, "Prediction of wind pressure coefficients on building surfaces using artificial neural networks", Energy and Buildings **158**, `10.1016/j.enbuild.2017.11.045` (2017).

[11] J. Brownlee, *Gentle introduction to the Adam optimization algorithm for deep learning*, (July 2017) `https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/` (visited on 06/29/2022).

[12] I. J. Goodfellow and Y. Bengio, "6.5 back-propagation and other differentiation algorithms", in (MIT Press, 2013) Chap. 6, pp. 200–220.

[13] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks", `10.48550/ARXIV.1406.2661` (2014).

[14] A. Radford, L. Metz, and S. Chintala, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", `10.48550/ARXIV.1511.06434` (2015).

[15]M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein GAN", 10.48550/ARXIV.1701.07875 (2017).
[16]I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved Training of Wasserstein GANs", 10.48550/ARXIV.1704.00028 (2017).
[17]M. Walia, B. Tierney, and S. Mckeever, "Synthesising tabular data using Wasserstein Conditional GANs with Gradient Penalty (WCGAN-GP)", in AICS (2020).
[18]J. Li, K. Niu, L. Liao, L. Wang, J. Liu, Y. Lei, and M. Zhang, "A generative steganography method based on WGAN-GP", in Artificial Intelligence and Security (2020), pp. 386–397.
[19]C. Hagmann, D. Lange, J. Verbeke, and D. Wright, *Cosmic-ray shower library (cry)*, (Mar. 2014) https://nuclear.llnl.gov/simulation/doc_cry_v1.7/cry.pdf (visited on 07/01/2022).
[20]S. Agostinelli et al., "Geant4—a simulation toolkit", Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **506**, 250–303 (2003).
[21]J. Brownlee, *Generative Adversarial Networks with Python: Deep Learning Generative Models for image synthesis and image translation* (Machine Learning Mastery, 2019).
[22]X. Mao, Q. Li, H. Xie, R. Y. K. Lau, Z. Wang, and S. P. Smolley, "Least Squares Generative Adversarial Networks", 10.48550/ARXIV.1611.04076 (2016).