Corso di Laurea triennale in Fisica (L-30)

UNIVERSITÀ DEGLI STUDI DI MILANO

FACOLTÀ DI SCIENZE E TECNOLOGIE

Dipartimento di Fisica “Aldo Pontremoli”

Generative Adversarial Networks for the simulation of cosmic ray glitches in LiteBIRD timelines

Relatore:
Prof. Maurizio TOMASI
Correlatrice:
Prof. Samantha STEVER (University of Okayama)

Tesi di Laurea di:
Matteo BARATTO
Matricola: 886813

Anno Accademico 2019/2020

Contents

Abstract

The aim of the thesis is the creation of an algorithm to generate simulated time-ordered data of cosmic ray noise on the LiteBIRD space telescope. For this purpose I have created a Generative Adversarial Network trained to reproduce the data coming from a physically-motivated simulator (based on the Monte Carlo methods) developed by Prof. Samantha Stever (University of Okayama) and Dr. Tommaso Ghigna (Kavli IPMU, Tokyo).

LiteBIRD is a space mission approved by JAXA: the launch of the satellite is scheduled for 2028 and will remain in space, more precisely at the Lagrangian point L_2, for 3 years of observations. The goal is to measure a specific pattern in the polarization of the Cosmic Microwave Background, the B-mode polarization. These patterns are very important because they are produced by primordial gravitational waves, and observing their existence would provide direct observational evidence of cosmic inflation.

Unlike previous ground-based missions, which were shielded from the Earth's magnetic field, L_2 space missions that use bolometers, such as Planck/HFI, are very susceptible to cosmic rays: these space missions maximize their sensitivity using bolometric detectors which detect signal as a change in heat, and cosmic rays heating these bolometers, generate systematic errors in observational measurements. It is thus necessary, during the design phase, to develop software capable of simulating the effect of cosmic rays on timelines in order to use this data to improve the deglitching process that will be carried out on the final collected data.

The most common approach is to use Monte Carlo techniques which are very time-consuming due to the fact that they have to iterate over a large number of hits per second (over 400 in our case) in the wafer, and that a statistically-relevant sample is on the timescale of years, and over a large number of detectors.

I tried a different approach, creating a software based on neural networks, which have the advantage of being much faster. In particular, I created a GAN, which is a generative model, based on two neural networks that collaborate in order to produce new realistic data. The two neural networks are called Generator and Discriminator: the generator has the task of generating realistic samples while the discriminator has the task of judging the generated data and sending feedback to the generator in order to improve its performance.

Once the two neural networks are trained, the generator can be used to simulate new samples (where a single sample is composed by twelve bolometers timeline of 20 s); I then checked the quality of the data by comparing them with the data generated classically: I analyzed the distribution of the data, the power spectral density, and the correlations both between bolometers and between consecutive samples. The results obtained led me to conclude that the generated samples are consistent with those generated classically.

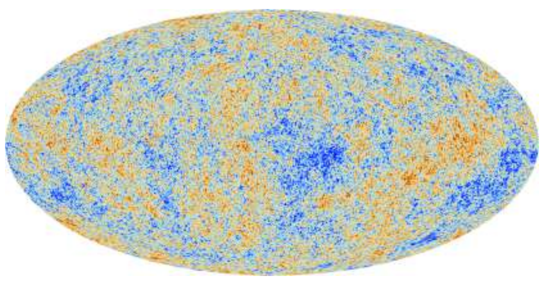
The most important aspect, as mentioned above, is the computation speed, which was measured by extrapolating the time needed to simulate a three-year timeline corresponding to the time of the mission: classic code on a single CPU would take tens of years, while the model I created completes the simulation in about 8 hours. A result that can be further improved by parallelizing the simulation on multiple CPUs or GPUs.

Chapter 1

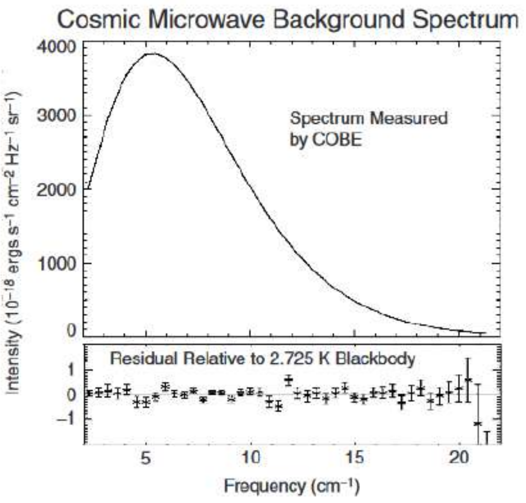
A physical overview

1.1 The Cosmic Microwave Background

Predicted in 1948 by R. Alpher et al., the Cosmic Microwave Background (CMB), also called *relic radiation*, was first observed by A. Penzias and R.W. Wilson in 1965 using the antenna of the Holmdel observatory in New Jersey. The CMB is an electromagnetic radiation produced in the early stage of the universe and permeate all the observable Universe (Fig. 1.1a). It is a quasi-isotropic radiation with a black-body spectrum at a temperature of $T \approx 2.73\,\mathrm{K}$ and a number density of photon $n_\gamma \approx 400\,\mathrm{cm}^{-3}$ [18].



(a)



(b)

Figure 1.1: (a) The anisotropies of the Cosmic Microwave Background (CMB) as observed by Planck. Source: https://sci.esa.int/s/WLGmGdw. (b) CMB's spectrum measured by COBE, the lower part show the errors bar on an expanded scale. Taken from [11]

1.1.1 Origin of the CMB

To understand the origins of the relic radiation we have to take a look at the history of our Universe. According to the Standard Cosmological Model, immediately after the Big Bang, the Universe underwent a rapid expansion called inflation. During this period (which lasted some $10^{-33}s$) any small regions of space were exponentially expanded and all the inhomogeneities was flattened out.

After the inflationary epoch, the expansion of the Universe continued with a slower rate. This expansion was followed by a progressive temperature drop of the plasma, and the reduced kinetic energy permitted the formation of particles like protons and neutrons ($t \approx 10s$). During this period, thanks to the high temperature, matter and radiation were in equilibrium and electrons interacted with photons through Thomson scattering. When temperature dropped below 3000 K nuclei and electrons began to combine, forming the first neutral atoms; the number density of electrons n_e decreased and the equilibrium between radiation and matter was thus broken.

The mean free path of photons because of Thomson scattering is:

$$\lambda_\gamma = \frac{1}{\sigma_T n_e}, \quad with \quad \sigma_T = \frac{8\pi}{3(\alpha\hbar c/m_e c^2)^2}.$$

When the electron density dropped to zero, the mean free path of photons diverged. The moment when λ_γ became larger than the the scale of the observable Universe is called *decoupling* ($t_{dec} \approx 380\,000\,\text{yr}$), at this moment the Universe became transparent to radiation: the photons began to freely propagate through space. The Cosmic Microwave Background is composed of these primordial photons.

The decoupling is assumed to have occurred everywhere at the same time in the Universe, clearly we can only detect the photons that have been scattered toward the Earth. More precisely today we observe the radiation produced by a spherical shell with radius[1] $d_{LSS} = c(t_{now} - t_{dec})$ called *Last Scattering Surface.*

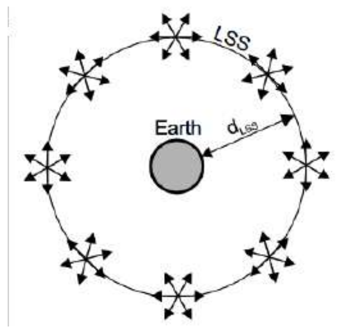


Figure 1.2: Representation of the Last Scattering Surface. Source: [18]

[1]To simplify the discussion, a non-expanding universe has been hypothesized.

1.1.2 Polarization

Thomson scattering, as shown in Fig. 1.3, preserves the polarization direction of the incident wave. In a realistic case, with a superposition of many plane waves, we expect that the scattered radiation is not polarized because the polarization induced by each wave mode is averaged out. More precisely, if the incoming radiation field were isotropic, orthogonal polarization states from incident directions separated by 90° would balance so that the outgoing radiation would remain unpolarized.

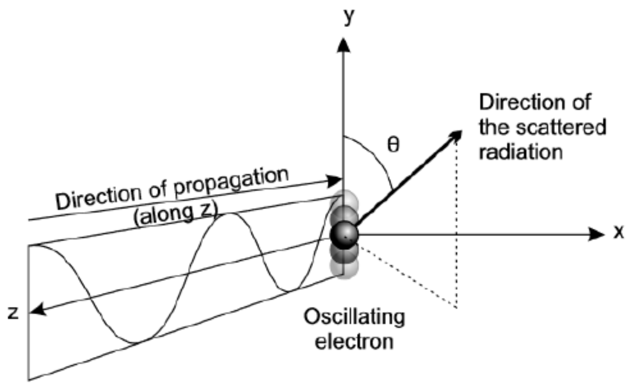

y
Direction of the scattered radiation
θ
Direction of propagation (along z)
x
z
Oscillating electron

Figure 1.3: Example of Thomson Scattering. Taken from [18]

If the radiation field has a quadrupolar component (a variation on an angle of 90°), polarisation can arise from scattering. Quadrupoles are defined mathematically as the Laplace's spherical harmonics $Y_l^m(\theta, \phi)$ with $l = 2$ and $m = 0, \pm1, \pm2$.

There are different perturbations that can generate quadrupole pattern in the radiation: as shown in [10, 18] this perturbation corresponds to a different value of $|m|$ according to their physical origin:

- Abiabatic pressure oscillation ($|m| = 0$)
- Vorticity in velocity field ($|m| = 1$)
- Gravitational waves ($|m| = 2$)

The degree of polarization depends on the thickness of last scattering surface. As a result, the polarized signal for standard models at angular scales of tens of arcminutes is about 10% of the total intensity (even less at larger scales) [2]. If decoupling were an instantaneous process, there was no time for photons in a quadrupolar configuration around a point P to scatter in P to produce polarization. On the other hand, if decoupling were too slow, quadrupolar anisotropies would smooth out while reaching thermal equilibrium, and thus polarization would not occur [18].

1.1.3 Polarization modes

To quantify the polarization we use the *Stokes parameters* I, Q, U, V: I is the average intensity of the radiation, Q and U measure the linear polarization and lastly, V measure the circular polarization (in the CMB, $V = 0$).

In particular, Q and U are the components of a second rank, traceless tensor P:

$$P_{ab} = \begin{pmatrix} Q & U \\ U & -Q \end{pmatrix}. \tag{1.1}$$

It is convenient to represent the polarization tensor in terms of two scalar field, the gradient field E (corresponding to E-type polarization) and the curl field B (corresponding to B-type polarization). These field can be expanded into spherical harmonics:

$$P_{ab}(\theta, \phi) = \sum_{l=0}^{\infty} \sum_{m=-l}^{l} (a^E_{lm} Y^E_{(lm)ab} + a^B_{lm} Y^B_{(lm)ab}). \tag{1.2}$$

Since E and B modes are defined in terms of Q and U using a non-local relationship, no detector can produce a direct measurement of E and B modes, regardless of the technology: the analysis of polarization data requires to measure Q and U and produce a sky map, and then to convert the map to E and B (typically in the harmonic space of the a_{lm} coefficients) [18].

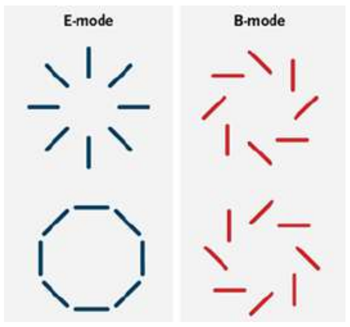


Figure 1.4: Representation of E-mode and B-mode polarization. Note that E-modes remain the same when reflected, while B-modes flip.

Polarization anisotrophies of the B-mode type can only be produced by primordial gravitational waves, which are such that $|m| = 2$. Therefore, measurements of the B-mode spectrum can prove the existence of relic Gravitational Waves during inflation.

The main problem is that the amount of power associated with B mode is smaller than the power in E modes. To quantify this problem it is common to use the r parameter (the ratio between the power of primordial B modes and the power of primordial E modes). Today the accepted upper limit for r is 0.06 with a 95% confidence. [13]

1.2 LiteBIRD

1.2.1 Overview

LiteBIRD (Lite (Light) satellite for the studies of B-mode polarization and Inflation from cosmic background Radiation Detection) is a satellite whose purpose will be the detection of B-mode polarization of the CMB radiation with an uncertainty in r of the order of 10^{-3}.

In May 2019, it was selected by the JAXA as a strategic large class mission. The launch is scheduled for the year 2028, and it will orbit the second Sun-Earth Lagrangian point L_2, for three years of observations. The Sun–Earth point L_2 has been selected because the Sun, the Earth, and the Moon are all located in almost the same direction, which makes it easier to avoid facing them in terms of optical and thermal aspects. [17].

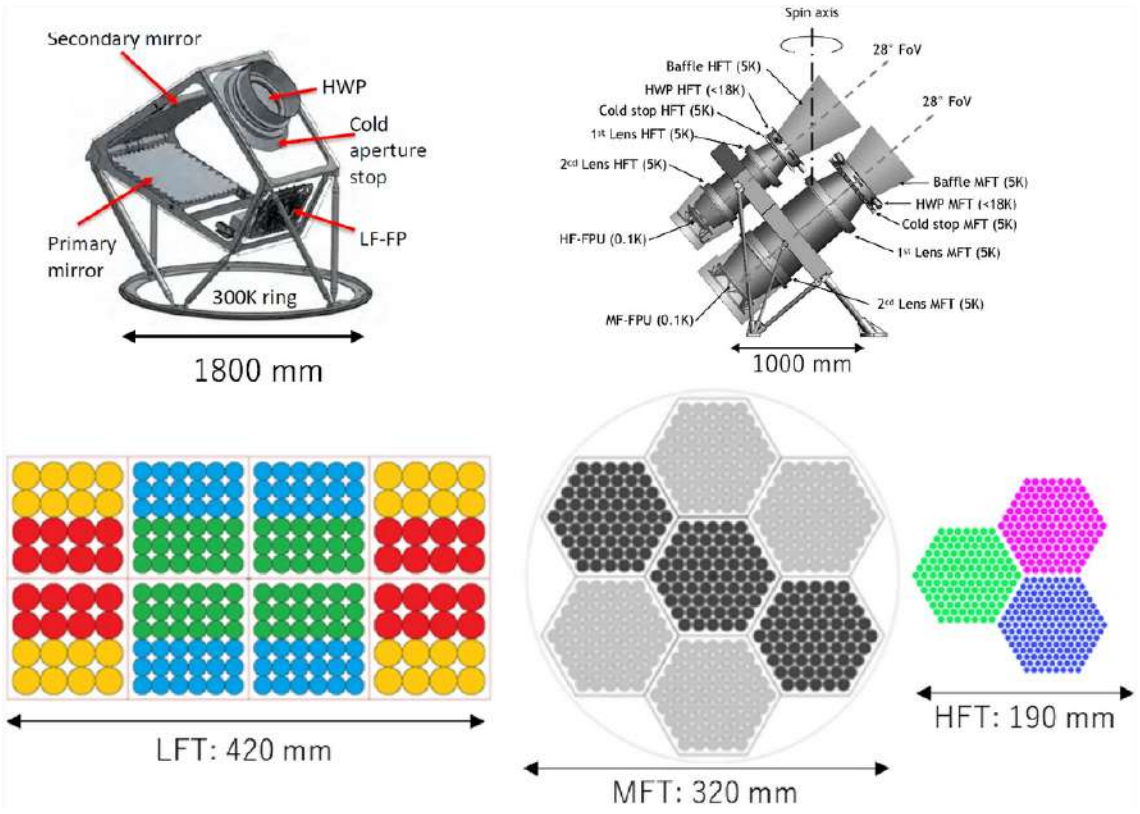


Figure 1.5: Design of the two telescope of LiteBIRD: *left* the Low Frequency Telescope, *right* the Medium and the High Frequency Telescope. Below is the pixel distribution. Taken from [17]

LiteBIRD is expected to observe 15 frequency bands covering a wide range from 34 to 448 GHz. This will be achieved with two telescopes (Fig. 1.5). The first uses a crossed-Dragone reflective optics, to minimize effects from multireflection among refractive surfaces, and mounts a low frequency instrument (LFT) to cover a frequency range of 34 GHz to 161 GHz. The second uses lens optics with the advantage of compactness, and mounts two different instruments: a medium frequency instrument (MFT, from 89 GHz to 224 GHz) and a high frequency instrument (HFT, from 166 GHz to 448 GHz).

The detectors will be TES (Transition-Edge Sensor) bolometers: in the current baseline design $\approx$ 4000 TES bolometers will be divided between the three instruments.

1.2.2 Detectors

A bolometer is a device created for measuring the power of the electromagnetic radiation incident on it using a material with a temperature-dependent electrical resistance. By absorbing the incoming photon, the detector heats up and this causes a change in electrical resistance which is measured as a signal. A bolometer is composed by an absorber having a heat capacity C connected to a heat reservoir at stable temperature T_0 by a weak thermal link (with thermal conductivity G) and by a thermometer that measures the ΔT when some electromagnetic power is absorbed.

The power P_{opt} of the incident radiation is related to the change of temperature by the following relation [6]:

$$C\frac{dT}{dt} = -P_b + P_{opt} + P_{bias}, \tag{1.3}$$

in which P_{bias} is a thermal dissipation due to an electrical bias and P_b is the total power flowing out of the bolometer through a link to a thermal bath, which is kept at a constant temperature:

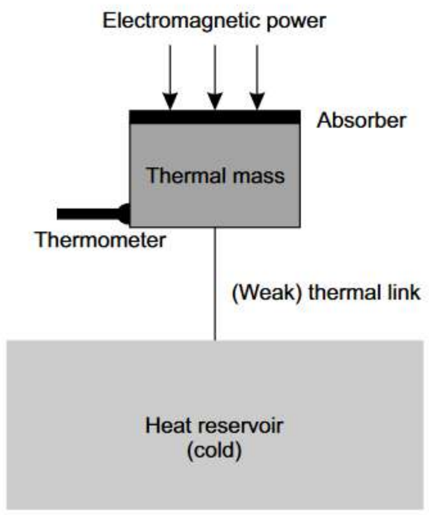
$$P_{bias} = \frac{V_{bias}^2}{R}, \quad P_b = \frac{G}{nT^{n-1}}(T^n - T_0^n), \tag{1.4}$$

where n depends on the primary thermal carrier.

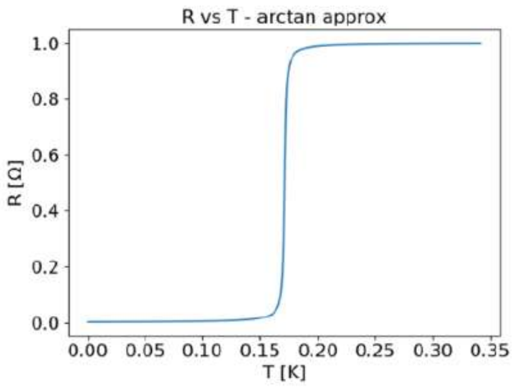
The power absorption heats up the bolometer and causes a change in the electrical resistance R; knowing the relation $R(T)$ one can measure the change in the resistance. This change causes a drop in the biased current, and the relation between incident radiation and the current is dependent on the gain $S_I = I/(P_{bias} + P_{opt})$, with $S_I \sim 1/V_{bias}$. Therefore, by measuring the current and using the previous formulas, it is possible to obtain P_{opt}. Refer to [6] for a more detailed discussion.

Transition-Edge Sensors (TES) are superconductive bolometers characterised by a small resistance ($R \approx 1\,\Omega$) and a positive logarithmic derivative α [6] (indicating the amount by which a change of temperature results in the change of resistance of the bolometer):

$$\alpha = \frac{d\log R}{d\log T} \geq 100 \tag{1.5}$$



(a)



(b)

Figure 1.6: (a)Scheme of a bolometer. Taken from [18]. (b) Temperature-Resistance relation for a TES bolometer at $T_0 \approx 0.17\,\mathrm{T}$ (LiteBIRD like). Taken from [6]

This technology has the advantage of being very sensitive and therefore allows for precision measurements of the CMB. However, this high sensitivity leads the instrument to be more subject to systematic errors caused by the presence of other heat sources that can impact on the detector, such as cosmic rays, which we will consider in the next section.

1.3 Cosmic Rays

Before LiteBIRD many other CMB observations were made, most of them were ground based missions and therefore not affected by the presence of cosmic rays, as they were shielded by the Earth's magnetic field. Previous space missions, like RELIKIT-1, COBE, WMAP and Planck/LFI, did not use bolometers and were therefore much less subject to cosmic ray disturbance.

The emblematic case is that of Planck/HFI (Sec. 1.3.2) which utilised bolometers (as will be LiteBIRD): at L_2 a bolometer will be exposed to Cosmic Rays because of the significant radiative environment and the lack of significant magnetic field that can shield the sensors. It is therefore vital to study the theme of cosmic rays and the effect they generate in order to account for this effect in future missions.

1.3.1 CR classification and features

Cosmic rays at L_2 are composed of about 89% protons, 10% α-particles, and 1% nuclei of heavier elements, and less than 1% are electrons [4]. We can distinguish two types of cosmic rays:

- Galactic Cosmic Rays (GCRs) and extragalactic cosmic rays: originated from outside the solar system, from within or beyond the local Galaxy.
- Solar Energetic Particles (SEPs): originated in the Sun during solar flares or coronal mass ejection.

The ratio between the two populations is influenced by the solar behavior (which oscillates in an 11 year cycle). There is a significant anticorrelation between solar activity and the CR's flux with energies below 10 GeV [4]. During so called solar maxima, the presence of the solar magnetic field is at its strongest, and this magnetic field significantly attenuates the flux of GCRs at L_2. Conversely, during solar minima, the GCR flux is strongest due to being the least inhibited by solar magnetic fields. On the other hand the Sun is a CR source, the typical energy reached in solar particles is in the keV range. It should be noted, however, that during a solar flare, the highest energy reached is typically 10 to 100 MeV, occasionally reaching 1 GeV (roughly once a year) to 10 GeV (roughly once a decade) [4].

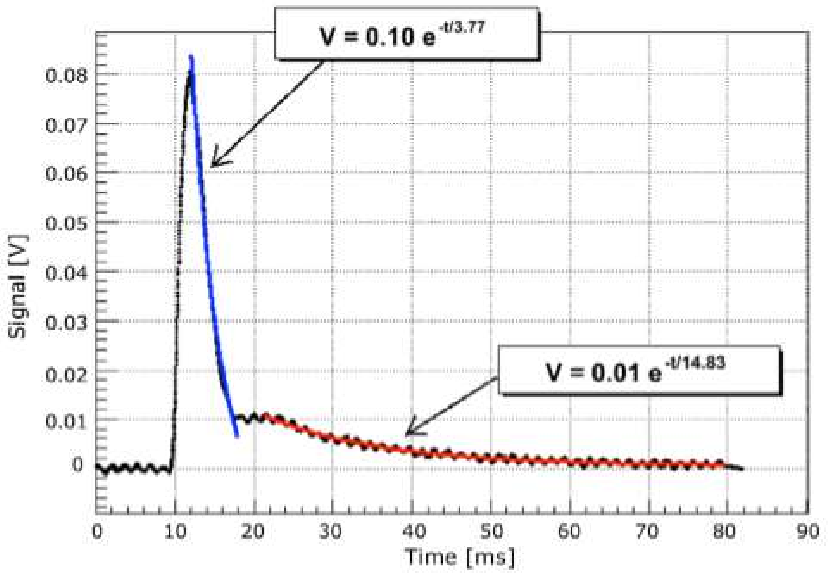


Figure 1.7: Plot of 466 event representing an hit on a silicon wafer. The two fits performed with an exponential model are also shown. Taken from [4]

The glitch caused by a cosmic ray in an isolated bolometer has a typical exponential shape (as show in Fig. 1.7). More precisely the shape is composed of two different component: a fast peak arising from ballistic phonon propagation and a slower component from thermal diffusion. The initial energy 'spike' followed by a slow dispersion is what gives the glitch its characteristic shape. The quickest mechanism of energy deposition is propagation via ballistic phonons which propagate to the sensor and are thermalised within or close to it and this can be seen in the 'fast' part of the glitch. The remaining component of the glitch, which is usually slower and of lower rise / decay times, is the thermal component [16]. This answer is valid for an isolated bolometer, as in the case of Planck/HFI, but in general the answer can be more complex and depends on the layout of the focal plane.

1.3.2 The case of Planck/HFI

The Planck mission has observed the sky between August 2009 and August 2013 in the frequency range from 30 GHz to 1 THz from the second Sun-Earth Lagrange point L_2 (the same position from which LiteBIRD will make its observations). The Planck satellite mounted two instruments on board: the High Frequency Instrument (HFI) and the Low Frequency Instrument (LFI). The HFI detectors exhibited a strong coupling with CR radiation that produced transient glitches in the raw time-ordered information. [4]

The presence of glitches in the collected data represents a percent-level uncertainty in the results obtainable from those data; in particular, there is a potential confusion between glitch residuals and the non-Gaussian features caused by topological defects or inflation processes [4]. In Fig. 1.8 we can observe and example of Planck's raw data from three different bolometers, it is evident that the CRs cannot be ignored.

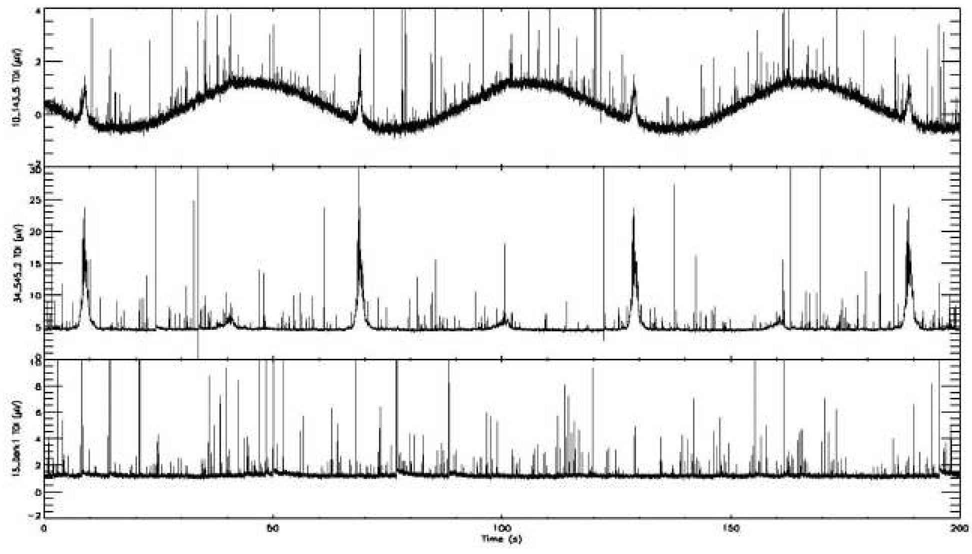


Figure 1.8: Raw TOIs for three bolometers illustrating the typical behaviour of a detector at 143 GHz, 545 GHz, and a blind detector ($t \in [0s, 200s]$). The typical maximum amplitude of a spike is between 100 and 500 mV depending on the bolometer. Taken from [3]

The data cleaning process (*deglitching*) has been also very complicated: due to the high event rate it was not possible to simply flag and remove all affected data. Multiple measurements of the same part of the sky were used to identify the presence of glitches, which were then removed. The deglitching algorithm took an estimated sky signal which is subtracted from the time-ordered data, and detected glitches with more than a 3σ variation. Of the selected data, an algorithm found fast changes (high derivatives) in the signal, indicating a cosmic ray impact. The glitches were then fit to using the same function that was used to analyse them (a variable template), and their signal was removed. The removed parts of the signal were then reconstructed using unflagged redundant data [16].

For future missions it is necessary that a study on the impact of cosmic rays on instruments is carried out in advance during the development phase. Hence the need to create programs able to simulate the effect of cosmic rays on the timelines measured by the detectors of space missions orbiting the L_2 point in order to ascertain the cosmic rays effect on final science outcome, as we will see in Chapter 3.

Chapter 2

Generative Adversarial Networks

We concluded the previous chapter by emphasizing the importance of removing the systematic effects caused by cosmic rays on the timelines of instruments such as those used by LiteBIRD. To optimize these removal techniques, it is first necessary to develop codes that simulate the presence of these systematic effects on the timelines, on which to test the removal algorithms. Unfortunately, simulating these events takes long computation times due to the large number of event that must be generated to accurately model cosmic ray propagation.

In my thesis I dealt with Generative Adversarial Networks (GANs) that are a set of machine learning models created by Ian Goodfellow, then a PhD student at the University of Montreal, in 2014 [8]. This technique enables a computer to generate new data using two separate neural networks. The potential contained in these algorithms is very high, and in recent years impressive results have been achieved (for example in the field of image generation).[1]

In this chapter we will see a first introduction on neural networks (Sec. 2.1), then we will see specifically the GANs (Sec. 2.2) and finally some advanced topics about stability and convergence of the algorithm (Sec. 2.3).

2.1 Neural Networks

A Neural Network (NN) is a computational model loosely inspired to the structure of biological networks of neurons in the brain. It is based on layers and neurons (see Sec. 2.1.1): every layer transforms the input data in a different representation trying to extract and learn as much information as possible. It is like a sieve for data processing, made of succession of increasingly refined data filters (the layers): some data goes in, and it comes out in a more useful form [5].

Typically a network can be composed of a restricted number of layers (as in the case of the model presented in Ch. 3) up to a very high number of them (*Deep*

[1]See for example `https://thispersondoesnotexist.com` .

Learning).

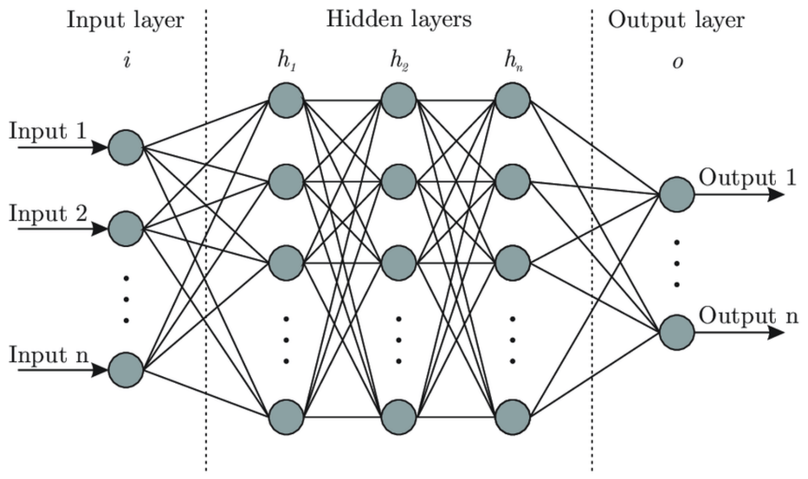


Figure 2.1: Example of a feed forward Neural Network composed of Dense layers. Taken from `https://bit.ly/3lSrPcl`

2.1.1 Layers and neurons

As we have previously stated, NNs are composed by a certain number of layers stacked successively; each single layer is a set of basic units, called *neurons*: a neuron is a mathematical function applied to the input data:

$$\text{output} = f(w \cdot x + b), \tag{2.1}$$

where x is the input, and w, b two parameters. A neuron is therefore composed by a linear combination followed by the application of a non-linear function f called *activation function*.

The choice of a specific activation function depends mainly on the type of problem that the neural network will have to solve. For example if we want the output to be between 0 and 1 (a probability) we will choose the function *sigmoid* whose image coincides with that range.

Typically a neural network is composed of several layers and each layer is composed of a certain number of neurons. If we consider, for simplicity, the layers to be *fully connected* (Dense layers, see Fig. 2.1) then the neural network is a composition of non linear function where the output of a layer of neurons is used as an input for the following layer [15]:

$$\text{output} = f(w_n \cdot f(...f(w_2 \cdot f(w_1 \cdot x + b_1) + b_2) + ...)... + b_n). \tag{2.2}$$

The network is therefore used to represent very complex nonlinear functions; the main feature, however, is the ability to learn, i.e. the ability to change the internal parameters to better perform the task for which it was created.

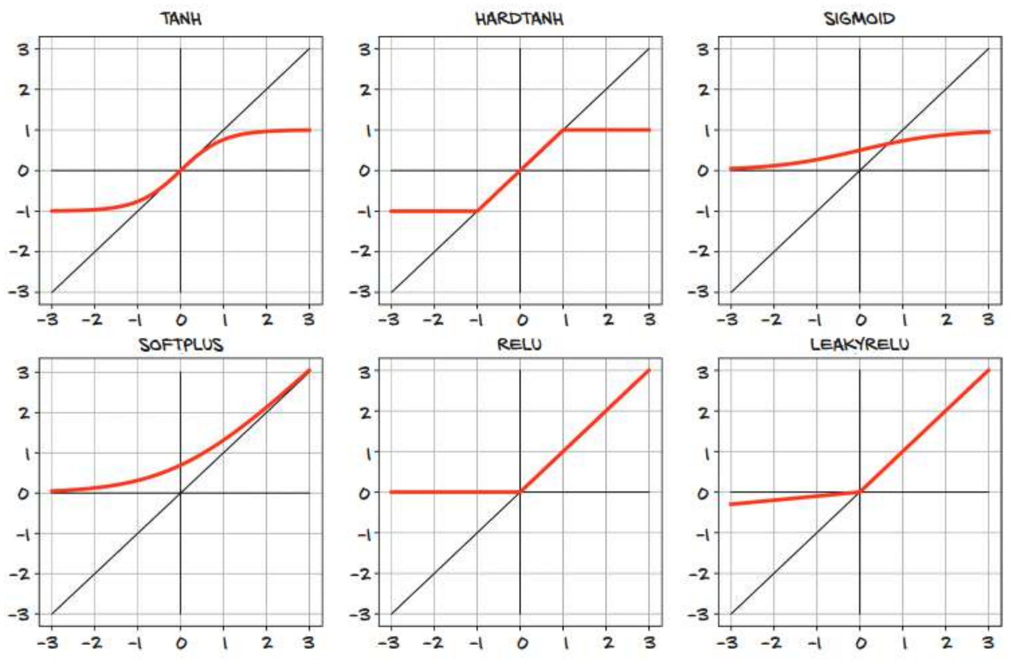


Figure 2.2: A collection of typical activation functions. Taken from [15]

2.1.2 Learning process

To obtain a usable neural network we will have to train it, so that it learns to set its own internal parameters (weights). The training has two main characters:

- A *loss function* to compare the desired output and the NN's output.
- An *optimizer* that updates the weights on the base of the loss function

The loss function is defined on $\mathbb{R}^{m\times n}$ and produces values in $\mathbb{R}^m$, where m is the number of samples that we are passing through the NN and n is the dimension of the output (the last layer). It computes a single numerical value for every sample that the network, through the optimizer, tries to minimize. There are many types of loss function, depending on the problem that are solving, for example the most straightforward could be the *mean absolute error* or the *mean squared error* that are usually used in regressional problem.[2]

Once the loss function has been applied and the loss score has been calculated, the optimizer uses this value to update the network's weights. In general this process is called *backpropagation*. In this case too, there are different optimizers to chose from. The most intuitive to analyze is the one based on the *Stochastic Gradient Descent*: the algorithm calculates the gradient of the loss function with respect to the internal parameters of the network:

$$\nabla_{w,b}(\mathcal{L}oss) = \left(\frac{d\mathcal{L}}{dw}, \frac{d\mathcal{L}}{db}\right). \tag{2.3}$$

[2]For a more detailed list of loss functions see: `https://keras.io/api/losses/`

Once the gradient is calculated, the optimizer moves the parameter in the opposite direction from the gradient (for example, $w = w -$ learning rate $\cdot$ gradient).

The learning process can be therefore summarized in the following steps:

1. Take a batch of input data and use the NN to process them.
2. Compare the obtained results with the desired output using a *Loss Function*.
3. Use the *optimizer* to update the network's weights seeking to minimize the loss (*backpropagation*).

The process is iterated until it reaches, in the best of cases, the global minimum of the loss function. The main problem of this implementation is the risk of get stuck in a local minimum and not reach the global minimum. To overcome this situation new optimizers implementations have been proposed (like RMSprop, Adam, Adagrad...) which consider not only the current value of the gradient but also the previous parameter update.

2.2 GANs

Generative Adversarial Networks are machine learning algorithms based on two neural networks: the goal is to generate new data similar to that of a training dataset. They can therefore be used to generate the effect that cosmic rays will have on LiteBIRD detectors, which as seen in section 1.3.2, is an important study that must be carried out in the development phase of the mission.

An intuitive way to understand GANs is to start from the following analogy that sees as main characters a forger and an art critic. Imagine a beginner forger trying to recreate a famous painting; at first he is pretty bad at the task. He then decides to call an art critic, of questionable moral, to analyse the fake paintings: the art critic sees the forger's work and tells him where the mistakes are. Over time, using the art critic's feedback the forger manages to improve his artistic skills and the critic becomes increasingly expert at spotting fakes. At the end the forger achieves such a skill that the art critic can no longer distinguish whether a painting is real or not, resulting in fakes very faithful to the originals.

2.2.1 Core Elements

In analogy with the example just examined, a GAN is composed of two different neural networks that work in symbiosis:

- A *Generator* takes a random vector as input and tries to produce a sample of fake realistic data.

- A *Discriminator* takes a sample (real or generated) as input and tries to estimate whether it is false or not.

The discriminator is a *classifier*, therefore typically the output associated with a sample is a single value in a range (e.g. $[0, 1]$ or $[-1, 1]$) where the upper bound represents a "perfect" real sample and the lower bound a fake sample (or vice versa). In Fig. 2.3 we can see an architecture diagram explaining the flow of data through the two neural networks.

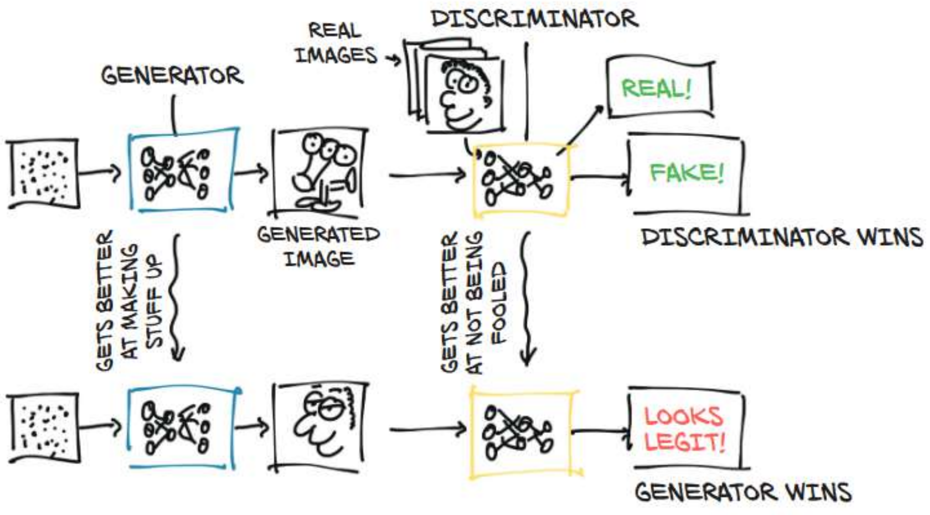


Figure 2.3: GAN's architecture. Taken from [15]

Once the two networks have been created, the next step is to train them (see Sec. 2.2.2): the discriminator is trained to more precisely discern the false from the true, while the generator is trained to produce more and more realistic images (trying to fool the discriminator). Lastly, when the training is finished and convergence is reached (see Sec. 2.3), we take the generator aside and finally use it to start producing new data.

2.2.2 Training process

The training process, mentioned above, is explained in Fig. 2.4. The discriminator and the generator (witch we will indicate respectively $D(x)$ and $G(x)$) are trained separately in the following way [12]:

1. Discriminator training:
   (a) Take a mini-batch of data x from the *real training dataset*.
   (b) Use a mini-batch of random noise vectors z to generate fake examples $G(z) = x^*$.

   (c) Use the discriminator to classify both the real sample x and the fake sample x^*.

   (d) Calculate the classification error[3] and backpropagate the error to update only the Discriminator's trainable parameters, seeking to *minimize* the error.

1. Generator training:

   (a) Get a new random noise vector z and generate a new mini-batch of samples x^*

   (b) Use the Discriminator to classify x^*.

   (c) Calculate the classification error and backpropagate to update only the Generator's trainable parameters, seeking to *maximize* the Discriminator's error.

As reported in [7], GAN training can be better described as a game,[4] rather than an optimization. One player is the discriminator $D(x)$, characterised by the set of parameters $\boldsymbol{\theta}^{(D)}$; the other player is the generator $G(x)$ having the set of parameters $\boldsymbol{\theta}^{(G)}$. Both players have a cost function $J(\boldsymbol{\theta})$ dependent on both the set of parameters. The discriminator tries to minimize the function $J^{(D)}(\boldsymbol{\theta}^{(G)}, \boldsymbol{\theta}^{(D)})$ tuning only $\boldsymbol{\theta}^{D}$; the generator instead tries to minimize $J^{(G)}(\boldsymbol{\theta}^{(G)}, \boldsymbol{\theta}^{(D)})$ tuning only $\boldsymbol{\theta}^{G}$. The fact that the two functions depend on both $\boldsymbol{\theta}(D)$ and $\boldsymbol{\theta}(G)$, and that a model can only control its own parameters and not those of the other, makes the GANs better framed as a game rather than as an optimization problem.

The training stops when the GAN converges, i.e. when it reaches a point of equilibrium that is a local minimum of $J^{(D)}$ with respect to $\boldsymbol{\theta}(D)$ and a local minimum of $J^{(G)}$ with respect to $\boldsymbol{\theta}(G)$. In practice this point is reached when:

- The generator produces fake samples that are indistinguishable from the real sample.

- The discriminator can only randomly guess whether a sample is real or fake.

Finding the right point of convergence of a GAN is complex because as the generator improves, the discriminator gets worse: it is therefore necessary to pay attention to when to end the training. If the training continued beyond the point of equilibrium, the feedback from the discriminator would become meaningless because

[3]The method by which it is calculated depends on the chosen loss function. A naive implementation could be the following: when the Discriminator's output is between 0 and 1 (where 0 is fake and 1 is real), if we classify a real sample and get 0.7 then the error is 0.3 i.e. the expectation value less 0.7.

[4]See: https://en.wikipedia.org/wiki/Game_theory

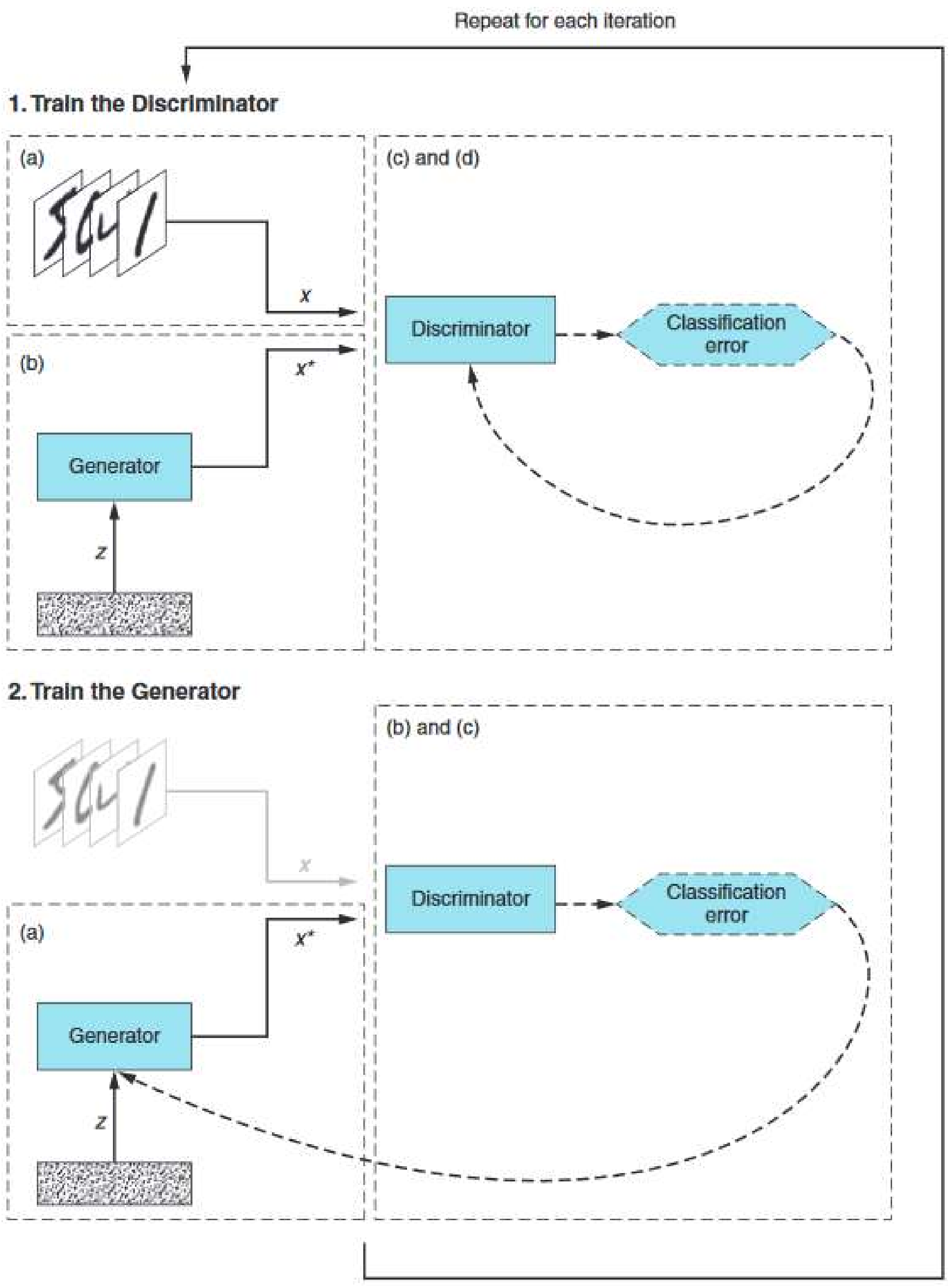


Figure 2.4: Training scheme. Taken from [12]

it would no longer be able to distinguish a true sample from a false one. The generator would then be trained on random judgments, leading to worse result.

Finally, we briefly discuss about the possibility of *overfitting*: the Generator does not make reference to the training data directly, all the information it receives comes solely from the discriminator in the form of an estimate that varies according to how much the two models learn. This makes GANs resistant to overfitting because the generator has no opportunity in practice to directly copy training examples [7]; if the generator were to overfit, theoretically, the discriminator would begin to reject the samples and would tell the generator how to improve to overcome the problem.

2.3 Types of GAN

GANs are a very recent and continuously developing field of research, in fact, since their creation in 2014, many other articles have been published on the subject. The main research areas concern the development of new setups in order to make the algorithm as stable and efficient as possible. In the next section we will firstly see the vanilla implementation [7] on witch, part of the previous explanations were based, and then we will move on to analyze more modern implementations.

2.3.1 Min-Max setup

A GAN can be considered as a *zero-sum game*, a situation in which one player's gains equal the other player's losses. The discriminator is a standard binary classifier and therefore we can chose the following cost function:

$$J^D = \mathbb{E}_x[\log[D(x)]] + \mathbb{E}_z[\log[1 - D(G(z))]], \tag{2.4}$$

where $\mathbb{E}$ stands for the expected value (over the real data in the first case and over the random inputs of the generator in the second case), D for the discriminator function mapping data in probability (in a range between 0 and 1) and G for the generator function. This loss function derives from the *binary cross entropy*: the discriminator tries to minimize the likelihood of mistaking a real sample for a fake one in the first part or a fake sample for a real one in the second part.

Because a GAN can be seen as a zero-sum game, and there are only two players, the generator's cost function will be:

$$J^G = -J^D. \tag{2.5}$$

We can then summarize the whole model as a *min-max* process in which one contender tries to maximize the same function that the other tries to minimize. The final parameters that the generator aspires to reach are the follow:

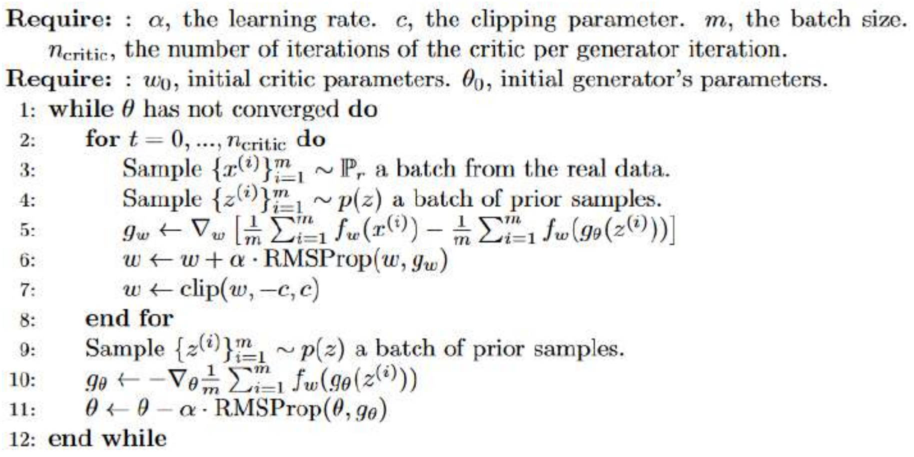
$$\boldsymbol{\theta}^G_{final} = \arg\min_{\boldsymbol{\theta}^G}\max_{\boldsymbol{\theta}^D}[J^G(\boldsymbol{\theta}^G, \boldsymbol{\theta}^D)]. \tag{2.6}$$

Although this setup is very useful, from a theoretical point of view, to fully understand the functioning of a GAN, in reality (as seen in Sec. 2.2.2) it can be very difficult to train. For this reason, over the years new and more efficient models have been developed, such as the ones we analyze below.

2.3.2 Wasserstein GAN

Wasserstein GANs (WGANs) were introduced in 2017 [1] to overcome the difficulties previously encountered, such as the problem of finding a stop criterion or the instability in the training due to "vanishing gradients"[5] ; this model is no longer based on binary cross entropy as loss function, but on the *Earth mover's distance*, also known as Wasserstein distance.[6]

Figure 2.5 shows a diagram that summarizes the main elements of a WGAN's training cycle. Note that this model introduces a new hyperparameter, the *clipping value*, because there is a technical constraint on the function represented by the discriminator [1]: $D(x)$ must be 1-Lipschitz.[7]



```
Require: : $\alpha$, the learning rate. $c$, the clipping parameter. $m$, the batch size.
    $n_{\text{critic}}$, the number of iterations of the critic per generator iteration.
Require: : $w_0$, initial critic parameters. $\theta_0$, initial generator's parameters.
while $\theta$ has not converged do
    for $t = 0, ..., n_{\text{critic}}$ do
        Sample $\{x^{(i)}\}_{i=1}^m \sim \mathbb{P}_r$ a batch from the real data.
        Sample $\{z^{(i)}\}_{i=1}^m \sim p(z)$ a batch of prior samples.
        $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^m f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^m f_w(g_\theta(z^{(i)}))\right]$
        $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
        $w \leftarrow \text{clip}(w, -c, c)$
    end for
    Sample $\{z^{(i)}\}_{i=1}^m \sim p(z)$ a batch of prior samples.
    $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^m f_w(g_\theta(z^{(i)}))$
    $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while
```

Figure 2.5: Pseudocode representing a training process. Taken from [1]

The loss functions used are the following:

$$L^D = D(x) - D(G(z)), \tag{2.7}$$

$$L^G = D(G(z)), \tag{2.8}$$

where x is an element taken from the real dataset and z a random noise vector.

In this new algorithm the discriminator does not actually classify sample as real or false using a number between 0 and 1; the output of the discriminator is no longer restricted to a range of numbers but can vary over the whole $\mathbb{R}$. Discriminator training just tries to make the output bigger for real instances than for fake instances: the aim is to maximize the gap between the judgment on a fake sample and that on a true one. At the same time it is like if the generator used the result of the discriminator as a real loss function and not as a value to be compared with the expected value through the binary cross entropy.

Simulations made using this architecture have shown that WGANs are more stable in training (less problems with vanishing gradients) and efficient in terms of

[5]When the discriminator gets too smart, the gradient of the generator vanishes. Basically, an optimal discriminator doesn't provide enough information for the generator to make progress. See `https://arxiv.org/abs/1701.04862`

[6]For a detailed mathematical discussion see [1].

[7]A function $\mathbf{f} : \mathbf{\Omega} \subseteq \mathbb{R}^{\mathbf{n}} \to \mathbb{R}^{\mathbf{m}}$ is L-Lipschitz if there is a constant $L \geq 1$ such as: $\|\mathbf{f}(\mathbf{x}) - \mathbf{f}(\mathbf{y})\| \leq K\|\mathbf{x} - \mathbf{y}\|, \quad \forall \mathbf{x}, \mathbf{y} \in \mathbf{\Omega}$

realism of results; moreover, perhaps the most revolutionary element is that the loss function correlates with the quality of the results, thus providing a very useful criterion to understand when to stop training. In fact, we remember that in the case of classic GANs, only the analysis of results after training and the accuracy of the discriminator (which had to reach 50%) were used as criteria, with all the problems we have already discussed in Sec. 2.2.2.

The only obstacle, already pointed out by the creators of the model themselves [1], is the sensitivity to the *clipping value*, which as we will see can create great instability in training.

2.3.3 WGAN with Gradient Penalty

Weight clipping in WGAN, in some cases, could leads to optimization difficulties such as vanishing or exploding gradients. In [9] we can find a new method no longer based on clipping value but on penalizing the norm of gradient of the discriminator with respect to its input.

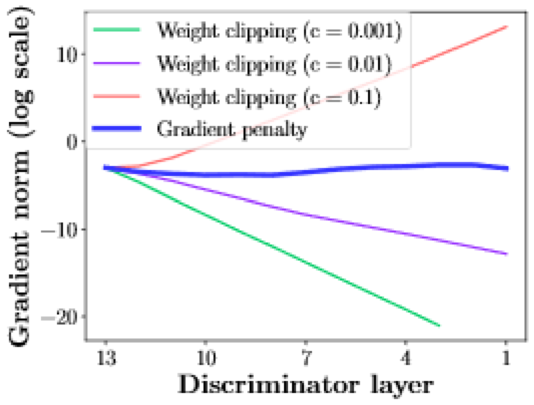


Figure 2.6: Gradient norm of a 12 layers discriminator during training with respect to successive layers, using different clipping values and gradient penalty. Taken from [9]

To avoid that the gradient either grows or decays exponentially as we move further back in the network, a solution could be use a gradient penalty, modifying the discriminator's loss function in the following way:

$$L = L_{\mathrm{W}} + \lambda(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2, \tag{2.9}$$

where L_W is the standard discriminator loss function (2.7), λ the weight we want the penalty to have, and $\hat{x}$ is a sample interpolated between a true and a false sample $\hat{x} = \epsilon x_{\mathrm{true}} + (1 - \epsilon)x_{\mathrm{false}}$ with ϵ a random number between 0 and 1.

Chapter 3

Results

After having seen the theoretical foundations of the CMB and having dealt with the problem represented by cosmic rays, we can move on to the heart of the thesis: create a Generative Adversarial Network (Chapter 2), which, starting from a training dataset, will learn to simulate cosmic rays impacts on the LiteBIRD bolometers in order to better study their effects and facilitate the "deglitching" of the data that will be collected by the satellite.

In the first section we will have a look at the training dataset: how it is generated and how it is manipulated before it is passed to my algorithm, which will be described in section 3.2. In the other two sections we will compare the data generated by the neural network with those produced classically, paying particular attention, in the last section, to the efficiency of the code in terms of execution speed.

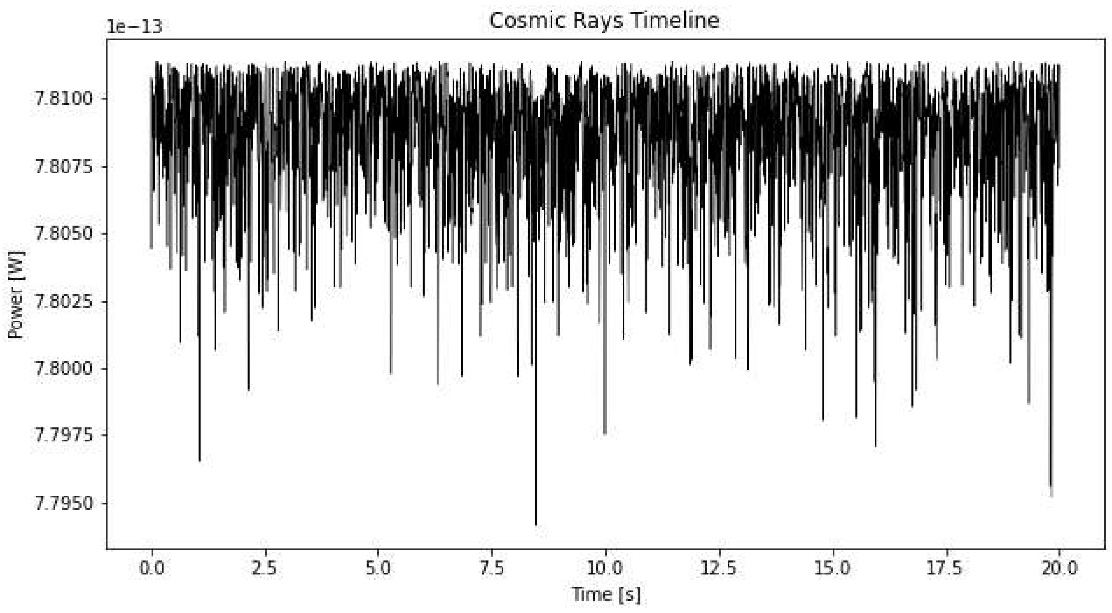


Figure 3.1: Example of a 20 s timeline relative to a single bolometer

3.1 Training set

The training dataset is generated using a Python algorithm based on Monte Carlo methods, developed by Prof. Samantha Stever and Dr. Tommaso Ghigna. The program is divided into two parts. In the first part, following a Poisson time distribution, it randomly generates a set of collision points: for each collision, the position on the wafer, the angle of incidence and the energy of the cosmic ray are sampled; finally calculating which bolometers will be influenced. In the second part all these information are used to solve the equations that rule the bolometers response [6], obtaining a timeline of the powers that will be measured.

In order to train the neural networks, a batch of 1000 samples was generated; each sample is a 20 s timeline of 12 distinct bolometers whose data are sampled at 156.3 Hz. This data is saved in a single NumPy tensor of shape (1000, 2, 12, 3126). The generated bolometers belong to three neighboring pixels of the LFT, where each pixel contains four bolometers.

3.2 My Generative Adversarial Network

Data Preparation

First of all, I split the dataset into *training set* (896 samples) and *test set* (104 samples): the first will be used to train the GAN while the second will be used to carry out the comparative tests reported in Sec. 3.3. To reduce the size of the training set I decided to eliminate the time axis: the data is sampled at a constant frequency, so it is of little interest to know the exact time of each data. The final shape of the training set is therefore (896, 12, 3126).

The data are not yet ready to be fed to the GAN, because they must be normalized first. Given the structure of the generator, which provides the hyperbolic tangent as activation function applied to the last layer, I chose to normalize also the input data in an interval $[-1, 1]$ using the following transformation:

$$x_{\text{normalized}} = 2\left(\frac{x - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}}\right) - 1. \tag{3.1}$$

The results generated by GAN will then have to be converted back into the original range of values using the inverse formula.

Discriminator and Generator

Following the model described in Chapter 2 I implemented two neural networks using *Keras*,[1] a deep learning framework for Python, that provides simple high-level features to create networks architectures.

[1] `https://keras.io/`

Both the generator and the discriminator are feed-forward neural networks based on 1D convolutional layers. This layer creates a convolution kernel that is convolved with the layer input over a single temporal dimension to produce a tensor of outputs, witch dimension is represented by the chosen number of filters.

The following code define a function that creates a discriminator using the keras.Sequential() tool.

```
def get_discriminator_model():

  model = Sequential()

  model.add(layers.Input(shape = data_shape))

  model.add(layers.Conv1D(16, 12, strides = 1,
        data_format='channels_first',
        activation = 'swish'))
  model.add(layers.LayerNormalization())

  model.add(layers.Conv1D(32, 12, strides = 2,
        data_format='channels_first',
        activation = 'swish'))
  model.add(layers.LayerNormalization())

  model.add(layers.Conv1D(64, 12, strides = 3,
        data_format='channels_first',
        activation = 'swish'))
  model.add(layers.LayerNormalization())

  model.add(layers.Flatten())
  model.add(layers.Dropout(0.4))
  model.add(layers.Dense(1))

  return model
```

The discriminator takes as input a tensor of shape (n, 12, 3126) where n depends of the *batch size* used in the training and return a single value (see Sec. 2.2.2). I used three 1D convolutional layers, each followed by a normalization layer that maintains the mean of the sample close to zero and the standard deviation close to to facilitate the training. As suggested in [9] I decided to use the *LayerNormalization* layer that normalize every sample independently, rather than across a batch like *BatchNormalization*. At the end I use a *Flatten* layer followed by a *Dense* layer (without any activation function) to obtain a single value.

The generator is a deconvolutional network defined in the following way:

```
def get_generator_model():

```

```python
    model = Sequential()

    model.add(layers.Input(shape = noise_dim))
    model.add(layers.Reshape((1, noise_dim)))

    model.add(layers.Conv1DTranspose(4 , 12, strides = 1,
            padding = 'same',
            data_format = 'channels_first',
            activation = 'swish'))
    model.add(layers.BatchNormalization())

    model.add(layers.Conv1DTranspose(8 , 12, strides = 2,
            padding = 'same',
            data_format='channels_first',
            activation = 'swish'))
    model.add(layers.BatchNormalization())

    model.add(layers.Conv1DTranspose(12, 12, strides = 2,
            padding = 'same',
            data_format='channels_first',
            activation = 'swish'))

    model.add(layers.Dense(data_shape[1], activation='tanh'))

    return model
```

It takes as input n vector of 500 random[2] points and generate a tensor of shape (n, 12, 3126). In this case I used three deconvolutional layers *Conv1DTranspose* which apply a transformation that goes in the opposite direction of a normal convolution. The first two are followed by a *BachNormalization* layer, while the third is directly connected to a dense layer that brings the sample to the final dimension and applies the hyperbolic tangent.

Following the tips suggested in [5, Sec. 8.5.2] instead of using *MaxPooling* or *AveragePooling* layers to downsample or upsample the data, I used strided convolution in order to avoid sparse gradients. For the same reason, despite it is recommended to prefer *LeackyReLU*[3] as activation function, I decided to use the *swish* activation function [14]

$$f(x) = x \cdot \text{sigmoid}(x) = x \cdot \frac{e^x}{1 + e^x}, \tag{3.2}$$

because I found that in my model it gives better results than LeackyReLU.

[2]Sampled from a Gaussian distribution with mean 0 and standard deviation 1

[3]`https://www.tensorflow.org/api_docs/python/tf/keras/layers/LeakyReLU`

Architecture

After trying to generate cosmic ray timelines with a vanilla GAN (Sec. 2.3.1), failing to make the networks converge, I switched to a more stable WGAN with gradient penalty (Sec. 2.3.3). For both the generator and the discriminator I chose as optimizer *Adam* with a learning rate of 0.0005. Furthermore during a single training iteration I trained the discriminator four times more than the generator, so that it converges sooner. The training lasted approximately ~ 100 iterations, until the loss function of the discriminator settled in a neighborhood of 0.

3.3 Generated data and comparisons

Once the GAN has been trained, I saved the generator model and started using it separately to produce new cosmic ray timelines for the twelve bolometers. First of all, I went to carry out various tests to check the quality of the generated samples.

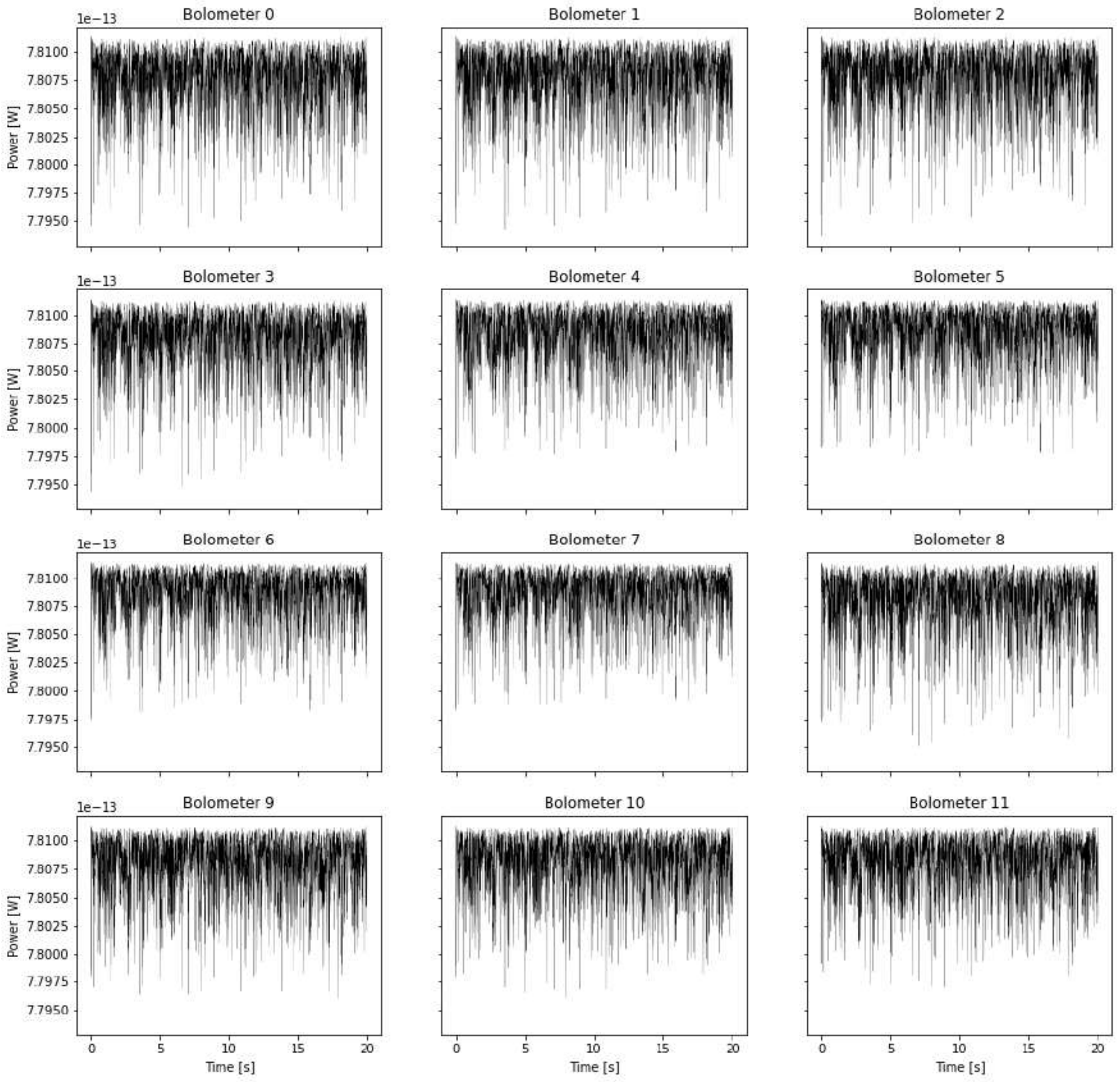


Figure 3.2: Sample of generated data, representing all the twelve bolometers.

Power distribution and power spectral density

After having visually verified that the timelines were generated in the right power range and that they did not present obvious anomalies, I carried out a qualitative analysis by studying the power distribution. The comparison was made by comparing multiple generated data with random sample taken from the test dataset, as shown in Fig. 3.3 the two distributions are comparable and, apart from some fluctuations, very similar. Since the samples are different from each other there could be ”worse” comparisons than others with slightly more pronounced deviations (e.g. the bolometer 1 in the figure).

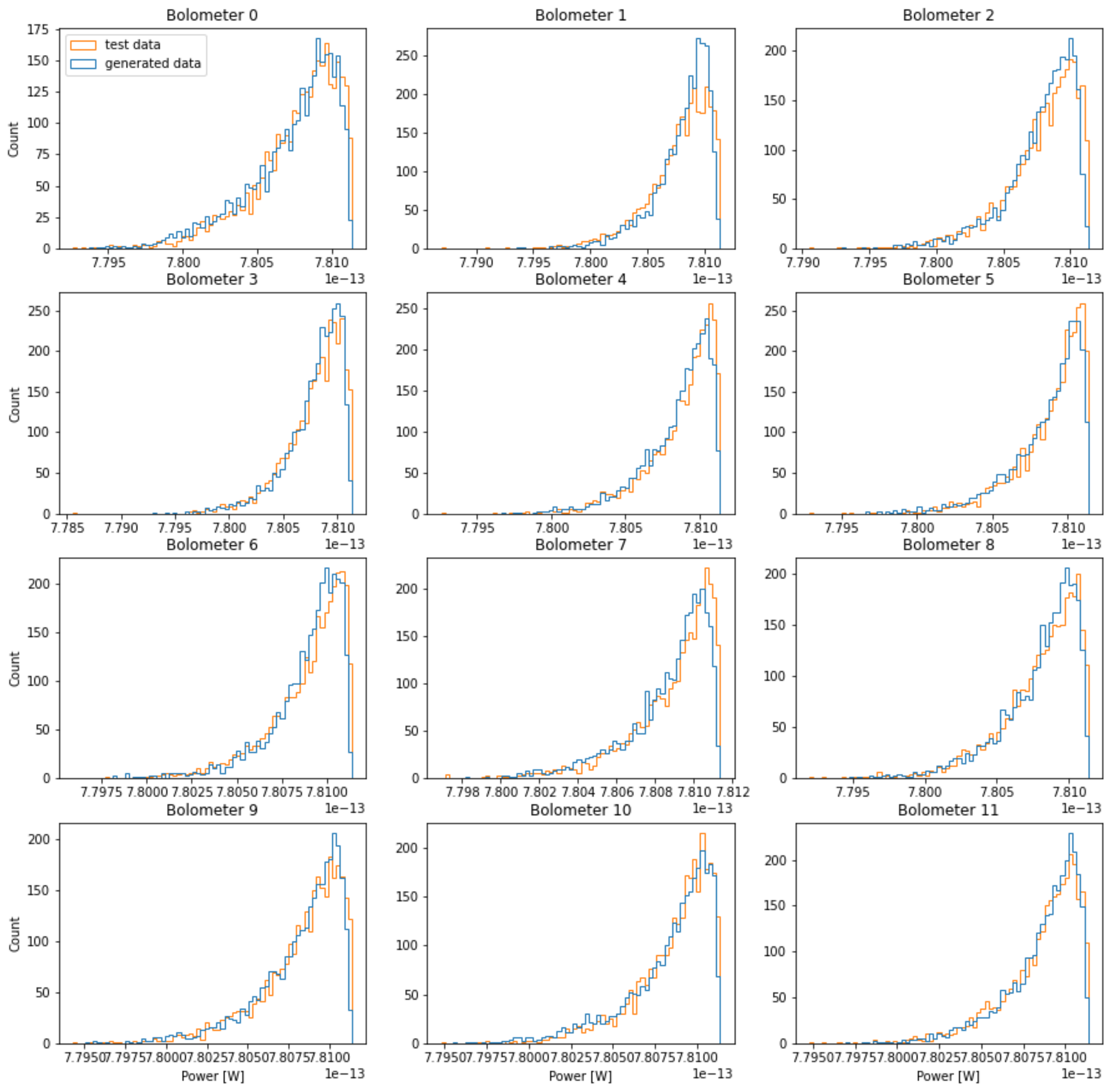


Figure 3.3: Comparison between the power distribution of a generated sample with a sample taken from the test dataset.

I therefore decided to also perform a comparison in the frequency domain, comparing the power spectral densities of the generated data with the test ones. In this case, for all the tests carried out the two distributions are not only similar but, at low frequencies, they were always found to be coincident; with increasing frequency, although an increase in variability is observed, the two curves remain comparable.

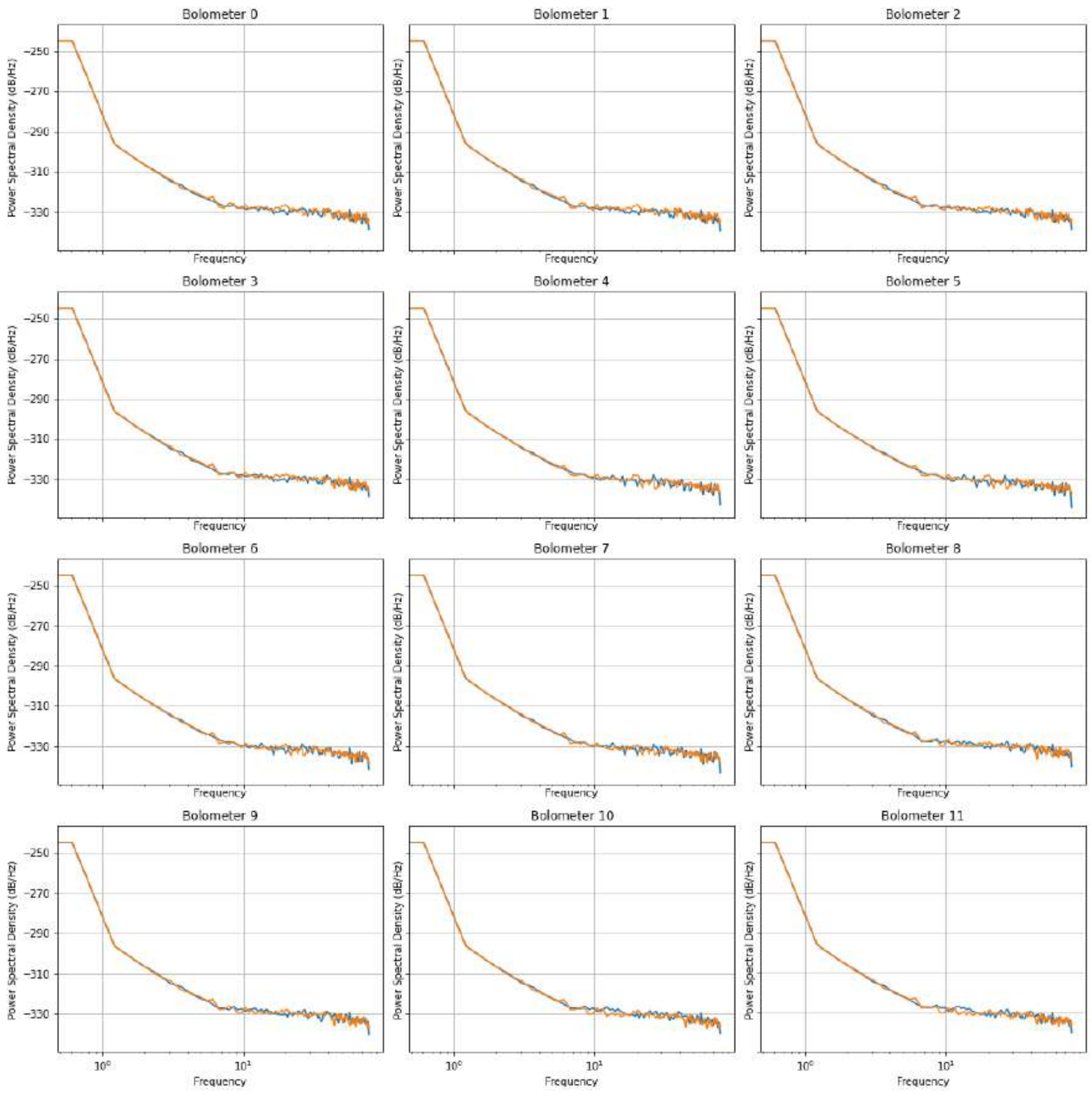


Figure 3.4: Comparison between the power spectral density of a generated sample and the one of a sample taken from the test dataset.

Correlation matrix

Since bolometers, when installed in a wafer, will be close together, one would expect that if a cosmic ray hits a bolometer, some of the power will also be propagated to nearby ones. For this reason I decided to study the correlation between the twelve bolometer. The elements of the matrix are the Pearson correlation coefficients:

$$c_{j,k} = \frac{\text{cov}[X_j, X_k]}{\sigma_{X_j}\sigma_{X_k}}, \tag{3.3}$$

where X_i is the i^{th} bolometer.

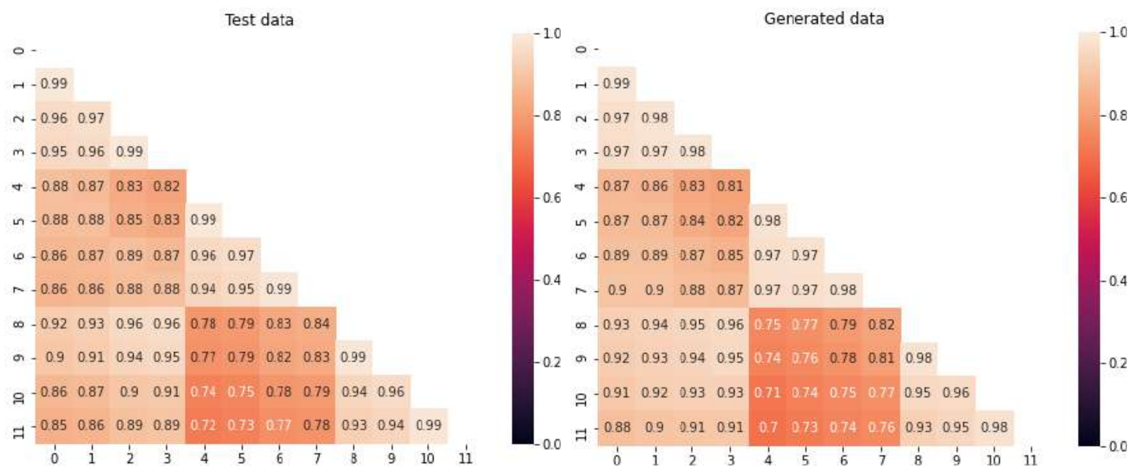


Figure 3.5: Mean correlation matrices for test data and generated data.

I calculated the matrix for all the test samples and for 100 generated samples, then averaging the coefficients. I found that actually there is a strong correlation between the bolometers of the test dataset and, even more curiously, there is a particular correlation pattern characterized by 4x4 sub-matrices, most likely caused by the position of the simulated bolometers in the wafer. As reported in Fig. 3.5 the generator has learned to reproduce the pattern in a very precise way, with differences in the order of magnitude of 10^{-2}.

Correlation between samples

To study if the GAN produces different sample or or has correlations between one sample and another, I generated a batch of 104 samples and I calculated the mean Pearson correlation coefficient for the twelve bolometer separately in three ways:

- among all the possible pairs present in the test dataset,
- among all the possible pairs composed of a generated element and a test element,
- among all the possible pairs present in the generated dataset.

I calculated the average and the standard deviation of all the values found and then, the average on the twelve bolometer, getting the following values:

	Test-Test	Test-Generated	Generated-Generated
Mean	$-3.16 \cdot 10^{-5}$	$4.66 \cdot 10^{-4}$	$4.99 \cdot 10^{-2}$
Mean Std Dev	$1.9 \cdot 10^{-2}$	$2.0 \cdot 10^{-2}$	$9.5 \cdot 10^{-2}$

Table 3.1

The correlation between the generated data is higher than that of the test data alone. Despite this, the three values are consistent with 0 within 1σ, which leads us to conclude that although there is a slight deterioration, the samples are still unrelated to each other.

3.4 Efficiency

The last aspect I went to study is the efficiency of my algorithm, in particular the execution speed. The code used to create the training set, based on the Monte Carlo methods, due to the many simulations it has to produce, employs $\approx 10\,\text{min}$ for a single timeline of $20\,\text{sec}$. The LiteBIRD mission will collect information for 3 consecutive years, which means that, wanting to simulate enough samples to cover the entire duration of the mission ($4,730,500$ samples), at worse, using a single CPU it would take us $\approx 90\,yr$.[4]

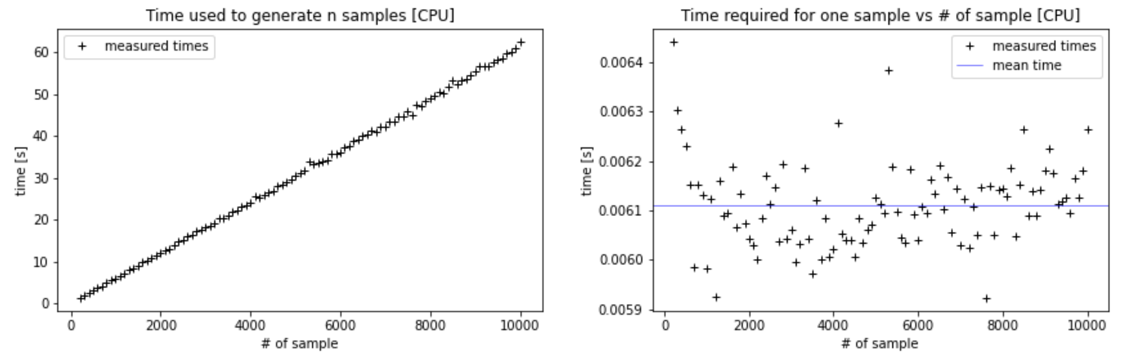


Figure 3.6: Results of the GAN simulation made on the CPU

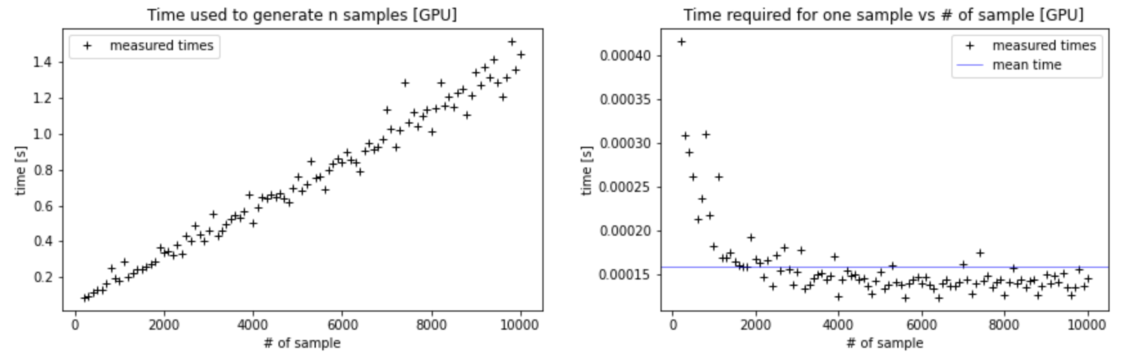


Figure 3.7: Results of the GAN simulation made on the GPU

To quantify the efficiency of my model I generated an ever-increasing number of samples (starting from 100 and arriving at 10000 with jumps of 100), in order to find a relationship between the time taken and the number of samples generated. Finally, I adopted two approaches: in the first I extrapolated the value corresponding to three

[4]It is however possible to run an end-to-end simulation using a 90 minute timeline with a "shuffling" on the TOD tensor.

years of data, in the second I calculated the average time to generate a sample (which can then be multiplied by the number of samples needed); since the two methods lead to very similar results, a single final value will be reported later.

These simulations were performed in *Google Colaboratory*, first using a single CPU (Fig.3.6) and then using a GPU (Fig.3.7), obtaining the following results:

	Mean time for one sample	Time for 3 yrs
CPU	$6.1 \cdot 10^{-3}\, s$	$\approx 8\,$hrs
GPU	$1.6 \cdot 10^{-4}\, s$	$\approx 12\,$min

Table 3.2

The results obtained confirm the expected efficiency of neural networks compared to traditional code, reducing the time it takes to produce a three-year timeline to relatively few hours. Furthermore, I have simulated the worst case, in fact the efficiency can be further increased by parallelizing the code on more CPUs or GPUs obtaining even better performance.

Conclusions

In this thesis, I simulated the effect of cosmic rays on the LiteBIRD instrument timelines creating a Generative Adversarial Network. Using as input the data generated by a classical code based on Monte Carlo methods, I trained the GAN in order to make it reproduce similar timelines. Once trained, the discriminator is set aside and the generator is used to produce new data.

A single sample of generated data consists of 12 timelines (12 bolometers) of 20 seconds each. The results obtained are qualitatively very promising, as the network is able to reproduce the power distribution and power spectral density of the original samples. Furthermore, since the bolometers will be positioned on the same wafer, a precise correlation pattern is observed in the classically generated data, which is reproduced by GAN with a mean absolute difference of the order of magnitude of 10^{-2}.

Further analysis to test the goodness of the generated data can be carried out by doing a dawn-sampling of the generated data, bringing them to the LiteBIRD sampling frequency of 19Hz, and going to perform a full end-to-end simulation that involves timeline-to-map projection, analyzing the differences between the classical data and those produced by the GAN generator.

The choice to use a code based on neural networks rather than a classic one based on Monte Carlo methods, lies in the speed of execution. As shown in Sec. 3.4, the code I created drastically reduces the time needed to produce a timeline of 3 years lasting the entire LiteBIRD mission: for a single CPU, we go from an assumed time of many years for the classic code to about ≈ 8 hours for the one based on neural networks. Using a single GPU instead, you get the most amazing results, the code generates 3 years of data in about 12 minutes, just over the same time it takes for classic code to generate 20 seconds on the CPU. The results can also be improved by switching to parallelized code between different CPUs or GPUs.

Finally, with regard to other possible future researches in the GAN field and more generally in neural networks applied to the simulation of the effects of cosmic rays, the possibilities are vast: for example, the GAN could be trained directly starting from data sampled at a lower frequency, in such a way to reduce the number of points or, by keeping the number of data constant, simulate longer times; furthermore, improvements in the architecture of my network are not excluded in order to increase data quality. It could also be thought of drastically changing the architecture of the generator using, for example, recurrent neural networks; or trying to approach the problem using different generative models, for example by combining GANs and Variational Autoencoders such as in VAE-GAN or BEGAN.

Bibliography

[1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. *Wasserstein GAN.* 2017. arXiv: 1701.07875 [stat.ML].

[2] Amedeo Balbi, Paolo Natoli, and Nicola Vittorio. “The CMB polarization: status and prospects”. In: (July 2006).

[3] A. Catalano et al. “Characterization and Physical Explanation of Energetic Particles on Planck HFI Instrument”. In: *J. Low Temp. Phys.* 176.5-6 (Mar. 2014). ISSN: 0022-2291. DOI: 10.1007/s10909-014-1116-6.

[4] Catalano, A. et al. “Impact of particles on the Planck HFI detectors: Ground-based measurements and physical interpretation”. In: *A&A* 569 (2014), A88. DOI: 10.1051/0004-6361/201423868. URL: https://doi.org/10.1051/0004-6361/201423868.

[5] François Chollet. *Deep Learning with Python.* Manning, 2018.

[6] Tommaso Ghigna. “Development of new generation receivers for experimental cosmology with the Cosmic Microwave Background and systematic effect studies”. PhD thesis. University of Oxford, 2020.

[7] Ian J. Goodfellow. “NIPS 2016 Tutorial: Generative Adversarial Networks”. In: *CoRR* abs/1701.00160 (2017). arXiv: 1701.00160. URL: http://arxiv.org/abs/1701.00160.

[8] Ian J. Goodfellow et al. *Generative Adversarial Networks.* 2014. arXiv: 1406.2661 [stat.ML].

[9] Ishaan Gulrajani et al. *Improved Training of Wasserstein GANs.* 2017. arXiv: 1704.00028 [cs.LG].

[10] Wayne Hu and Martin White. “A CMB polarization primer”. In: *New Astronomy* 2.4 (1997), pp. 323–344. ISSN: 1384-1076. DOI: https://doi.org/10.1016/S1384-1076(97)00022-5. URL: http://www.sciencedirect.com/science/article/pii/S1384107697000225.

[11] Marc Leslie Kutner. *Astronomy: a physical perspective.* 2nd ed. Cambridge University Press, 2003.

[12] Jakub Langr and Vladimir Bok. *GANs in Action.* Manning, 2019.

[13] Planck Collaboration et al. “Planck 2018 results - VI. Cosmological parameters”. In: *A&A* 641 (2020), A6. DOI: 10.1051/0004-6361/201833910. URL: https://doi.org/10.1051/0004-6361/201833910.

[14] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. *Searching for Activation Functions*. 2017. arXiv: 1710.05941 [cs.NE].

[15] Eli Stevens and Luca Antiga. *Deep Learning with PyTorch*. Manning, 2019.

[16] Samantha Lynn Stever. “Characterisation and modelling of the interaction between sub-Kelvin bolometric detectors and cosmic rays”. Theses. Université Paris Saclay (COmUE), Jan. 2019. URL: https://tel.archives-ouvertes.fr/tel-02091039.

[17] H. Sugai et al. “Updated Design of the CMB Polarization Experiment Satellite LiteBIRD”. In: *J. Low Temp. Phys.* 199.3 (May 2020), pp. 1107–1117. ISSN: 1573-7357. DOI: 10.1007/s10909-019-02329-w.

[18] Maurizio Tomasi, Cristian Franceschet, and Sabrina Realini. “The quest for CMB B-modes”. In: (2020).