NOVA IMS Information Management School

MAA

Mestrado em Métodos Analíticos Avançados
Master Program in Advanced Analytics

GAN Hyperparameters search through Genetic Algorithm

Umberto Tammaro

Project Work presented as partial requirement for obtaining the Master's degree in Advanced Analytics

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

NOVA Information Management School

Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

GAN hyperparameters search

through genetic algorithm

Author:

Umberto Tammaro (M20190806)

Nova Information Management School

Master's in advanced Analytics

Supervisor:

Prof. Mauro Castelli

Nova Information Management School

November 2021

DEDICATION

Dedicato a Mamma, Papà e Federica.

Nonostante la distanza, il vostro supporto e affetto si fa sentire forte come sempre.

Grazie di tutto.

INDEX

LIST OF FIGURES

ABSTRACT

Recent developments in Deep Learning are remarkable when it comes to generative models. The main reason for such progress is because of Generative Adversarial Networks (GANs) [1]. Introduced in a paper by Ian Goodfellow in 2014 GANs are machine learning models that are made of two neural networks: a Generator and a Discriminator. These two compete amongst each other to generate new, synthetic instances of data that resemble the real one. Despite their great potential, there are present challenges in their training, which include training instability, mode collapse, and vanishing gradient. A lot of research has been done on how to overcome these challenges, however, there was no significant proof found on whether modern techniques consistently outperform vanilla GAN. The performances of GANs are also highly dependent on the dataset they are trained on. One of the main challenges is related to the search for hyperparameters. In this thesis, we try to overcome this challenge by applying an evolutionary algorithm to search for the best hyperparameters for a WGAN. We use Kullback-Leibler divergence to calculate the fitness of the individuals, and in the end, we select the best set of parameters generated by the evolutionary algorithm. The parameters of the best-selected individuals are maintained throughout the generations. We compare our approach with the standard hyperparameters given by the state-of-art.

1. INTRODUCTION

Artificial neural networks have successfully been used in a large number of applications, in recent years their growth has led to extraordinary findings. Among them, the Genetic Adversarial Networks are those that most captured the attention of the public. GANs have been used for a variety of generative tasks, varying from images to time-series data leading to impressive results. All these algorithms, unfortunately, create several problems since it is necessary to establish several parameters for their network design. For instance, in the case of multilayer perceptrons, it is necessary to establish learning rate, initial weights, number of hidden layers, and so on. The search for the right parameters has become one of the main challenges to getting these tools to work. These problems usually are approached using methods as random search and grid search. Although they usually work fine, they are time-consuming and often do not lead to the best results. Among the possible alternatives to tackle these problems, there is one that is leading to great results in little time: the evolutionary algorithms (EA). Inspired by biological evolution, EAs imitate evolution via reproduction, mutation natural selection, and survival of the fittest. They search more or less randomly in the space of solutions, paying special attention to those zones where the evaluation function is at maximum leading to faster and better results compared to traditional methods. In this thesis we present a possible way to apply EAs to identify the best set of hyperparameters for a WGAN-GP, trying to outperform the standard set.

Our work has been inspired a lot by the research done by academics in recent years. A lot of techniques have been studied on how to apply a genetic algorithm to the parameters research of neural networks, some focused more on the structure of the architecture and others on the tuning of the hyper-parameters. We have decided to focus mainly on the search for hyper-parameters. Using a predefined network, we wanted to see how much the results would change just with genetic research for the best set of values. Fundamental to our algorithm, has been the research published on [17]. Although our approach has been similar to this one, we have focused on a different set of parameters to tune. Also, most of the research is focusing on using and improving GAN using convolutional layers. These architectures happen to be the most used because the images are the most common data to work with when dealing with GANs. In our case, we wanted to check if the research would apply also to tabular data, that although can seem easier to work with, often shows way more complicated distributions.

The thesis is organized in the following way:

In section (2) we discuss the basic concepts of the GANs, giving the background study conducted for the thesis.

In section (3) we explain the idea behind Evolutionary Algorithms and how they are constructed, presenting the basic operators with which they are built.

In section (4) we present a review of the various approaches found in the bibliography and the different aspects that are possible to evolve in neural networks.

In section (5) we present the work project that has been done, introducing our proposed approach. We further describe the experimental setups and the detailed results of the proposed methods.

2. GENERATIVE ADVERSARIAL NETWORK

According to Yann LeCun, Chief AI scientist of Facebook, a generative adversarial network is *"the most interesting idea in the last 10 years of Machine Learning".*

Most of the algorithms in deep learning have been focused on the development of discriminative models, but with the introduction of Generative Adversarial Networks (GANs), a big step on generative models has been done. GANs were introduced in a paper by Ian Goodfellow and other researchers at the University of Montreal in 2014 [1]. GANs are a deep learning framework qualified to capture the distribution of true examples for new data example generation.

Figure 2.1. Human faces generated by GAN (2018)

Figure 2.1 shows images of human faces created by GANs, since 2014 the progress done on this architecture has flourished and has led to generating unbelievable realistic outputs. Today, GANs are used to create many different types of data, being in the form of media, time series or tabular data. The different synthetic outputs can be used for different tasks, such as training different machine learning models or overcoming the new privacy challenges of data. GANs belong to the class of generative algorithms. These algorithms can be divided into two classes: *explicit density model* where the distribution of the data is assumed, or *implicit density model* where the data instances are produced without explicit hypothesis. GANs belong to the second class and they were proposed to overcome all the disadvantages of the other generative algorithms.

Behind the GANs there is the adversarial learning idea, where there is a generator that tries to create as realistic an example as possible to deceive the discriminator [2]. This adversarial idea exists in different machine learning algorithms and is used for different objectives. This idea has provoked the birth of a new field of study: adversarial machine learning [3].

2.1 STRUCTURE OF GAN

Figure 2.2 shows a schematic representation of GAN. The structure of GANs is composed of two deep neural networks (a Generator and a Discriminator) playing a minimax game with each other. The Generator tries to create samples of data as close as possible to reality, in the meanwhile, the discriminator network tries to distinguish whether the samples are real or fake. This figure represents the vanilla GAN, so the first model ever developed. To today, given the fact that GANs can be used to generate a variety of different data, a lot of new architectures have been developed.

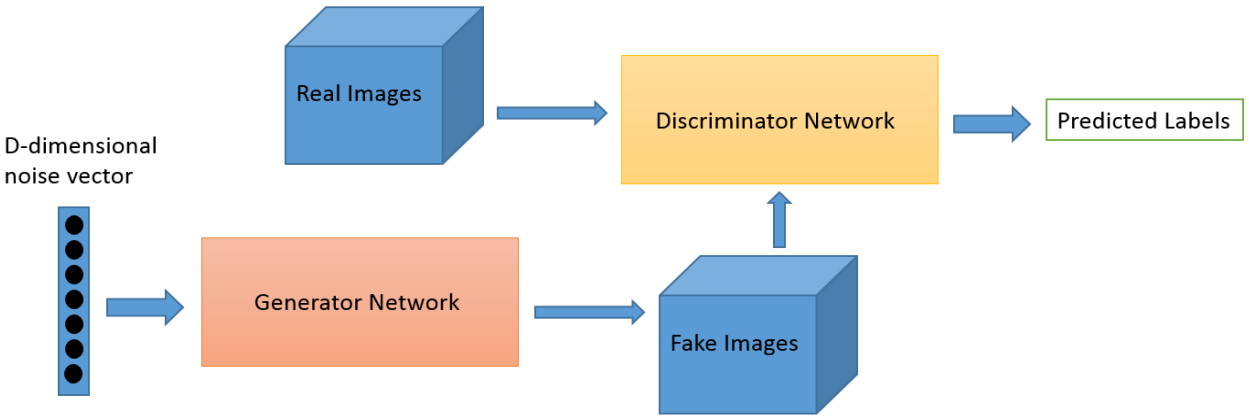


Figure 2.2. Schematic representation of a GAN (O'Reilly)

As we can see from figure 2.2 the generator never sees the real data, therefore, in order to learn how to generate realistic samples it receives feedback from the discriminator, in this way, it learns to make the discriminator classify its output as real. The more the generator and the discriminator play this game, the more they advance each other's skills. The training proceeds until the generator becomes good enough at creating synthetic data that it is possible to use it for the industry. Neural networks need some form of input, in this case, while the discriminator is fed with real samples, the generator takes random noise as its input so that updating its weights transforms this noise into a meaningful output.

Experiments suggest that the distribution of the noise is not so important, so we can choose something easy to sample from, as a uniform distribution.

Once the generator masters to mimic the distribution of training samples, we can generate concrete outputs such as images, videos, text, numerical simulations, and just about anything else one can imagine.

2.2 TRAINING GANs

The GAN training strategy is to define a game between two competing networks. The model is most straightforward to apply when the models are both multilayer perceptron. The *generator* network maps a source of noise $p_z(z)$ to the input space as $G(z;\theta_g)$, where G is a differentiable function represented by a multilayer perceptron with parameters θ_g. The discriminator network, also represented as a multilayer perceptron $D(x;\theta_d)$, receives a generated sample and a true data sample and outputs a single scalar.

Formally, the game between the generator G and the discriminator D is the minimax objective:

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

Where $D(x)$ represents the probability that x came from the data rather than p_g. To avoid overfitting and maintain an equilibrium between the two networks the training is organized in alternating k steps of optimizing D and one step of optimizing G.

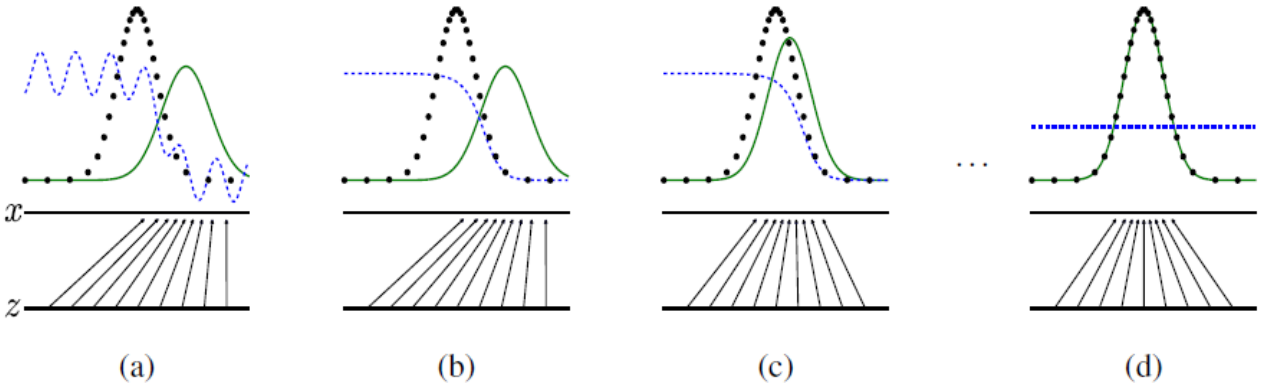


Figure 2.3. The process of learning of a GAN

In figure 2.3 is possible to visualize the process of learning of the GAN. The blue dashed line represents the discriminative distribution D that discriminates between samples from the real data distribution (black, dotted line) p_x from those of the generative distribution p_g (green, solid line). The lower horizontal line is the domain from which z is sampled. From the graphs is it possible to see how the discriminator, trained in an inner loop, in the beginning, has no difficulties whatsoever to identify which data is from which distribution. Following the updates of G, the gradient of D guides the $G(z)$ to flow to regions that are more likely to be classified as real data. After several steps, we arrive at the final graph where G and D will reach a point at which both cannot improve because the two distributions are equal. To generate such realistic outputs, GANs need to be well trained, and this task is often hard and unstable for several reasons.

2.3. COMMON PROBLEMS OF GANs

2.3.1 Mode Collapse

One of the main problems with GANs is the insurgence of mode collapse. When training a GAN, we want to produce outputs that are like the real data but at the same time they are different from each other. It could happen that the generator learns to generate samples from only a few modes of the data distribution but misses many other modes [4]. It does so by mapping several different input values $p_z(z)$ to the same output point $G(z)$.

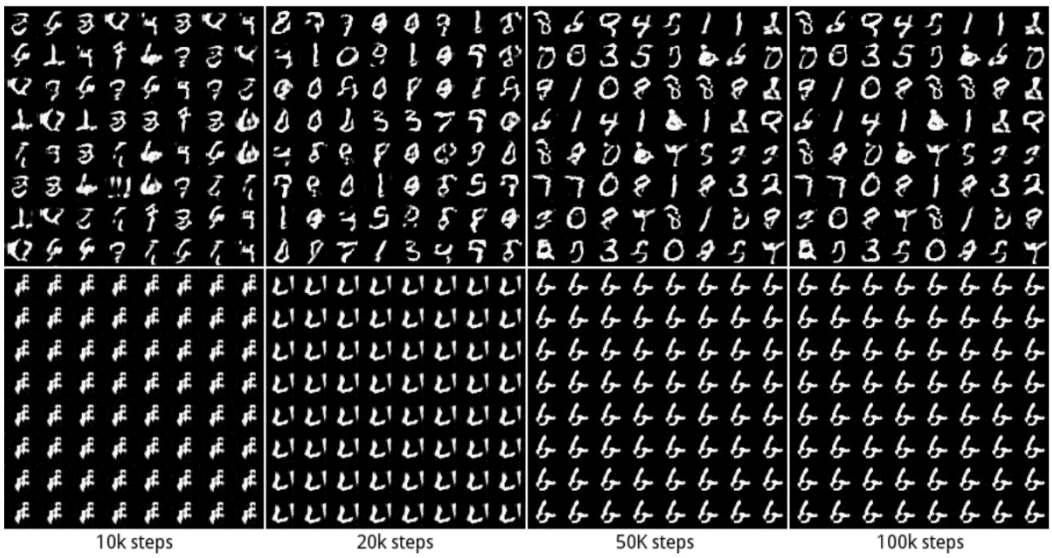


Figure 2.4. Example of Mode Collapse in MNIST dataset.

This would lead the generator to create very similar outputs, and in the worst-case scenario, it may learn only one output that seems plausible to the discriminator. For example, if we want to generate images of human faces, in case of mode collapse occurs, the GAN will always create the same face. The cause of mode collapse could be found in either the objective function or the structure of the GAN. There have been different attempts to overcome this problem with both structural and objective function approaches.

2.3.2 Failure to Converge

At the beginning of the training, the discriminator easily classifies if a sample is fake or real, but after some time it starts to have a hard time understanding which one is fake and which one is not. If the generator succeeds perfectly, then the discriminator has a 50% accuracy, in which case it would signify that the discriminator just tries to guess. This progression brings a problem to the whole GAN: the backpropagation from the discriminator starts to become less meaningful over time. In case the training of the GAN continues after reaching that point, the feedback from the discriminator would be totally random, leading the generator to train on aleatory values that might let the quality collapse.

2.3.3 Vanishing Gradients

The vanishing gradient problem is an issue that sometimes arises when training neural networks through gradient descent. The gradients control how much the network is going to learn during training, if the gradients get close to zero, then there is almost no improvement taking place. In the case of GANs, this happens when the discriminator is too good. The gradient along which the generator must train is diminished to the point that cannot usefully be used to learn. In the case of the discriminator being too good, when calculating the gradients there would be close to zero values updating the weights of the whole network. So, the feedback from the discriminator becomes too insignificant to let the generator make improvements. An attempt to overcome this problem is proposed in the GAN paper [1] with a modification to the minimax loss function. In 2017 another variant of the GAN has been designed to prevent this problem: the Wasserstein GAN [5].

2.4 WASSERTSTEIN GAN

Most of the problems of vanilla GAN are to research in the loss function. Wasserstein GAN introduces the Wasserstein distance, also called Earth Mover's distance (EM distance). The EM distance $W(q,p)$ is informally defined as the minimum cost of transporting mass in order to transform the distribution q into the distribution p (where the cost is mass times transport distance) (Ishaan Gulrajani, 2017)[6]. In the paper [5] is being argued that the divergences which GANs typically minimize are potentially not continuous with respect to the generator's parameters, leading to training difficulty. So, they instead propose to use the EM distance that under certain assumptions is continuous everywhere and differentiable almost everywhere. So, the WGAN value function based on the EM distance is:

In this formula, D is the set of 1-Lipschitz functions while the other values are interpreted as the GAN

$$\min_G \max_{D \in \mathcal{D}} \mathop{\mathbb{E}}_{\boldsymbol{x} \sim \mathbb{P}_r} \left[D(\boldsymbol{x}) \right] - \mathop{\mathbb{E}}_{\tilde{\boldsymbol{x}} \sim \mathbb{P}_g} \left[D(\tilde{\boldsymbol{x}}) \right]$$

formula, with the model distribution $\mathbb{P}_g$ defined by $\tilde{x} = G(z),\ z \sim p(z)$. Lipschitz functions are those one limited in how fast they can change, so that exists a real number such that, for every pair of points on the graph of those functions, the absolute value of the slope of the line connecting them is not greater than this real number [7]. In this case, the discriminator is called the 'critic' because it is not used anymore as a classifier but as a helper for estimating the Wasserstein metric between the two distributions of data (real and generated). Another change made by the WGAN is the addiction of "clip weights" used to enforce the Lipschitz constraint on the critic. After every gradient update on the critic function, the weights are clipped to a small range $[-c, c]$. Despite has been observed that the WGAN value function leads to better sample quality, the main critique of the method is the latter mentioned weight clipping.

2.5 WASSERSTEIN GAN WITH GRADIENT PENALTY

To overcome the defects of the WGAN, a different way of enforcing the Lipschitz continuity has been proposed by I. Gulrajani et al. 2017 [6]. The two main problems of weight clipping found in [6] were the capacity underuse and the exploding/vanishing gradients. With the intent of attaining the Lipschitz continuity via the weight clipping, it has been observed that the neural networks end up learning extremely simple functions that lead to the capacity underuse of the whole architecture. The problem of vanishing or exploding gradient occurs with the tuning of the clipping threshold c that if is badly set

would lead to difficulties in the interactions between cost function and weight constraint. The solution to these problems has been introduced with an alternative to weight clipping: gradient penalty.

A differentiable function, to be considered 1-Lipschitz must have its gradients with the norm at most 1 everywhere. To achieve it they constrained directly the gradient norm of the critic's output, so the new objective function proposed is

$$L = \underset{\tilde{\boldsymbol{x}}\sim\mathbb{P}_g}{\mathbb{E}}\left[D(\tilde{\boldsymbol{x}})\right] - \underset{\boldsymbol{x}\sim\mathbb{P}_r}{\mathbb{E}}\left[D(\boldsymbol{x})\right] + \lambda \underset{\hat{\boldsymbol{x}}\sim\mathbb{P}_{\hat{\boldsymbol{x}}}}{\mathbb{E}}\left[(\|\nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}})\|_2 - 1)^2\right].$$

The formula is composed of two parts, the first one being the original critic loss and the second one being the gradient penalty. This method has been proved to improve the training speed and sample quality compared to weight clipping.

3. EVOLUTIONARY COMPUTING

Since the invention of the computer, the idea of evolution as a metaphor for problem-solving has been introduced. In the 1970s and 1980s, this idea has been developed into different algorithmic implementations [8]. Evolutionary Computing has been used in the scope of optimization problems. These stochastic optimization techniques are based on natural evolution and the fittest strategy found in biological organisms. They have been successfully applied to solve complex optimization problems across different areas. Their ability to be suitable for such a variety of different problems is by cause of the fact that they can be thought of as working on two levels. On a higher level, we have the phenotype that represents a collection of aspects (like eye color) and that is what we change in this algorithm. On a lower level, in order to work with this kind of abstract concept, we have the genotype, which is the representation of such abstract aspects in a form that can be manipulated.

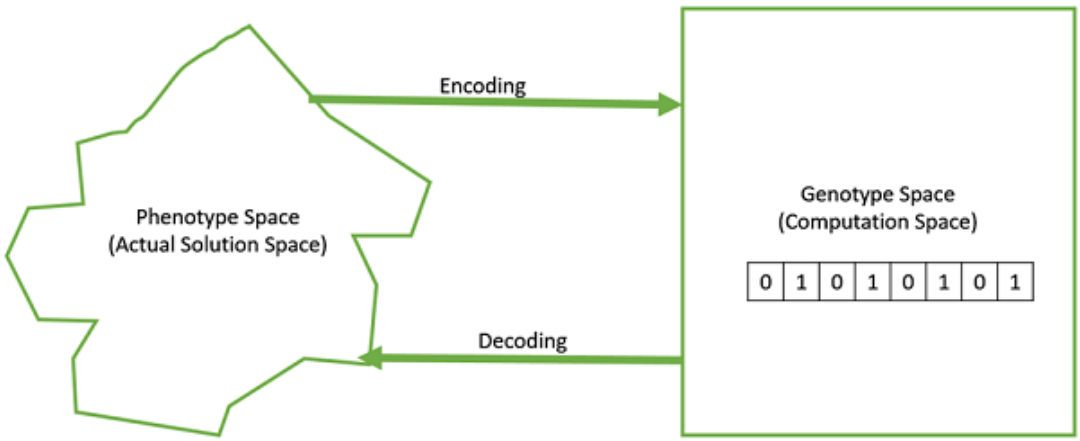


Figure 3.1. Phenotype-Genotype mapping

This two-level representation makes it possible to adapt a small group of possible genotypes to many kinds of phenotypes. From this field, a lot of algorithms have been arising, such as genetic algorithms, evolutionary strategy, genetic programming, and so on.

3.1 GENETIC ALGORITHM

Genetic algorithm (GA) has been firstly presented by J. Holland in 1975 [9]. GA is an algorithm to mimic the natural evolution of species, it is widely applied for optimization problems. The basic algorithm is often called a simple genetic algorithm (SGA) and from this base, a lot of improved algorithms have been introduced.

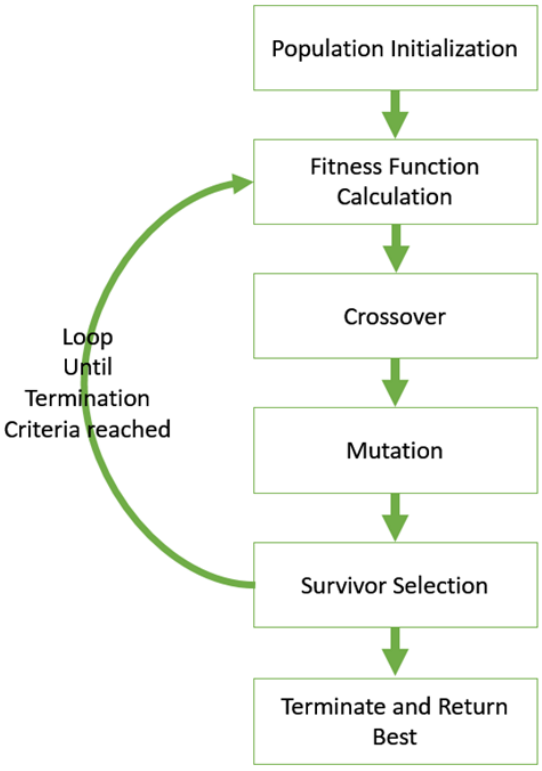


Figure 3.2. Flow chart of a Genetic Algorithm

In figure 3.2 is possible to see the classical process of the genetic algorithm. It starts with a population of individuals (chromosomes) randomly initialized. Then by applying different genetic operators it tries

to evolve these individuals to fitter ones, in order to make them better than the previous ones in terms of a certain measure called fitness function.

3.2 GENETIC OPERATORS

The genetic operators that are meant to improve the individuals are three: selection, crossover, and mutation.

3.2.1 Crossover operator

The crossover operator is the main process to generate new individuals, it consists of exchanging information between two different individuals. The crossover is inspired by reproduction and the biological crossover.

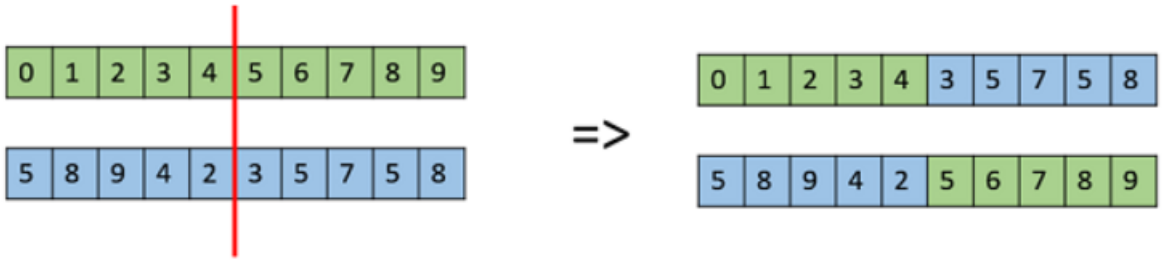


Figure 3.3. One-point Crossover

The easiest one is the one-point crossover, which consists of selecting a random point in the chromosomes and then swapping the tails of the two parents to produce offspring. There are multiple variations of this operator, such as multi-point crossover where you select more than one point or uniform crossover where each gene is randomly selected to be swapped. In order to decide which one is the most fitted for the problem we need to look if there are any constraints and how the method selected would affect the individuals. The crossover is used for the exploration of the search space (space of all possible solutions) because it makes more significant changes to the chromosomes.

3.2.2 Mutation operator

Inspired by biology, the mutation is an alteration of the genes in a chromosome. The main purpose of the mutation is to introduce diversity in the population, in order to avoid the convergence to all similar individuals. A classic example of mutation is the bit-flip mutation. In a binary encoded chromosome, one of the random genes is selected and they are flipped.

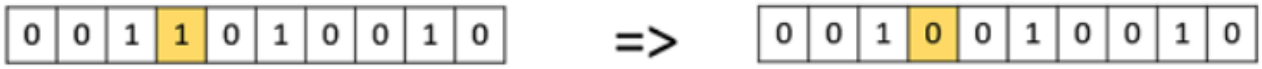


Figure 3.4. Bit Flip Mutation

As for crossover, not all the mutations are well-fitted for every problem. The mutation is considered essential to the convergence of the genetic algorithm for its ability to avoid local minima by preventing similar chromosomes. Contrary to crossover, the mutation is used for the exploitation of the chromosomes because it remains “closer” to the original parent.

3.2.3 Selection operator

The selection is the stage of the GA where the chromosomes to be passed onto the next generation are chosen. This is one crucial step to drive the generations into better solutions. A good balance between population diversity and good chromosomes must be found in order to avoid premature convergence. In case of premature convergence, the population, converging too early, would lead the algorithm to get stuck into local optima resulting in the inefficiency of the search for the best individual. One of the most popular ways of selecting is basing the choice on the fitness of the individuals. The fitness of an individual is determined by a function that takes as input the candidate solution and produces as output

the score to determine how good or fit is the solution. Usually, the fitness function matches the objective function of the problem. One common fitness proportionate selection is the Roulette Wheel.

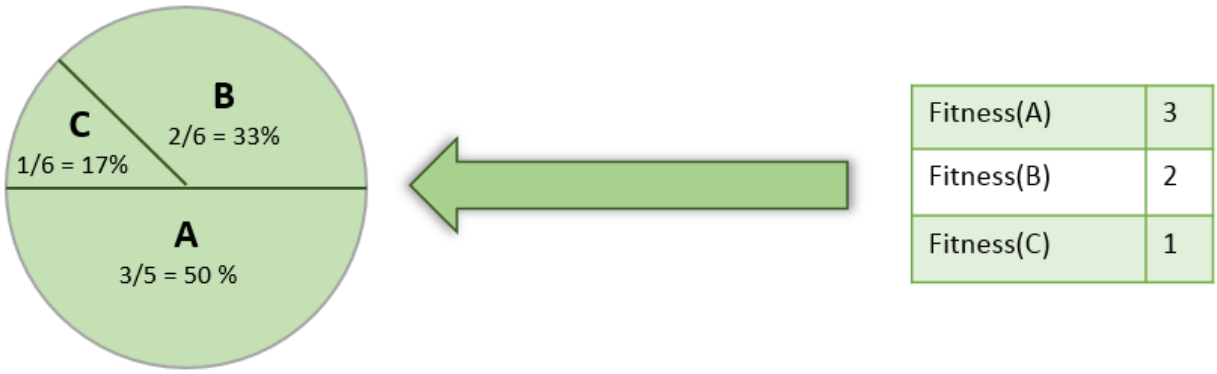


Figure 3.5. Roulette Wheel Selection

The main idea behind the roulette wheel is that better individuals get a higher chance of being chosen, and these chances are proportional to fitness. Once determined the percentages of the individuals (portions of the wheel) we spin the wheel n times to select n individuals. The selected individuals will be part of the next generation of individuals. As per the other operations that are different ways to implement the selection of the individuals.

3.3 MAIN ADVANTAGES

The major advantages of the Evolutionary Algorithms compared with traditional optimization techniques are [10]:

- EAs tend to escape more easily from the local optimum because of the randomness introduced by the operators.
- EAs do not require prior knowledge on the specific domain even though, if available, it could be exploited.
- EAs are conceptually simple and relatively easy to implement.
- EAs do not require objective function to be continuous and can be used in algebraic form.

There are also some problems regarding EA, such as poor performance in handling constraints, long computation time, and high computational complexity when the solution space is hard to explore.

4. RELATED WORK

In the latest years, given the difficulty of finding the best parameters for neural networks, multiple experiments using genetic algorithms have been conducted. The approaches to the usage of genetic algorithms have been various. Three main approaches can be found: *evolution of connection weights* that is a global approach to the initial weights and biases to obtain a fast convergence; *network architecture evolution* that implies an adaptation of the network topology to establish its learning and generalization ability; *learning parameters and rule evolution* that is the adaptation of the hyperparameters and learning rules [11].

4.1 APPROACHES RESEARCHED

4.1.1 Evolving connection weights

The initial weights of a neural network can highly influence the speed of convergence in backpropagation, since, depending on the point of the search space from which it has started, better or worse solutions can be obtained. Weight evolution involves the setting initial values and the way the EA changes them. The simplest method of all is based on making a random initialization, other methods require statistical analysis of the training data, which makes the process more complicated but at the same time more powerful. Chang et al. [12] researched the optimization of the initial weights applied to hip bone fracture prediction. Their strategy was to define for each artificial network the training data and let the GA find the optimum initial weights. The encoding of the weight and biases consisted of a representation of one digit. Given the fact that the initial population's range would affect the search efficiency, they proposed to narrow the range between -0.5 and 0.5. For the evaluation of the chromosomes, the mean square error has been used, representing how the solution was fit for the problem. In order to avoid choosing bad solutions they put a threshold on the validation set's error; in case the error was higher than what was expected the chromosome would not be chosen.

The stopping criteria consisted in stopping the algorithms after 100 epochs. The study found out that, despite all the difficulties and limitations found in the design of the algorithm, using a simple GA has been proved to be effective for improving the accuracy of neural networks.

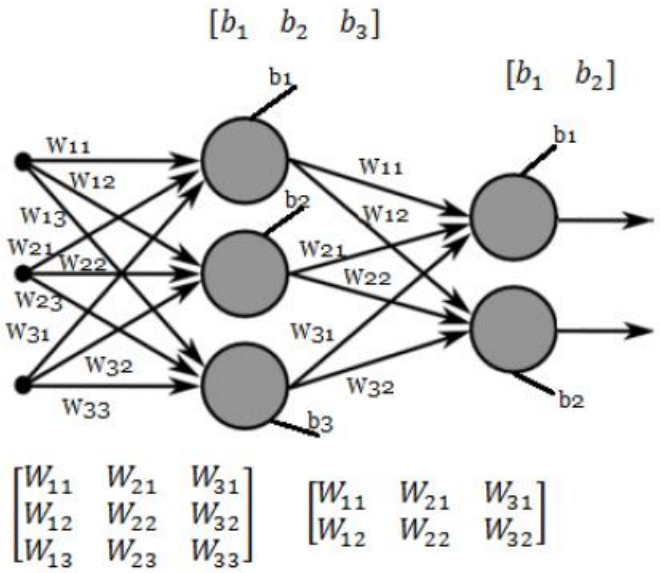


Figure 4.1. Matrix encoding of a solution.

The same findings were obtained by Orive et al. [13]. Their study was also conducted on the initialization of the weights. Different from the latter was the codification of a solution, matrices were used instead of simple digits. (Figure 4.1). In the paper different types of crossover have been used and has been proven that there are significant differences between the results using different crossover operators. Other than these little differences the results were the same: tuning the initial weights via genetic algorithm has been statistically demonstrated that led to significantly better performances.

4.1.2 Network Architectures

Until now, the architectures have been designed manually by experts with enough experience. The network structure is very important, a poor-designed architecture could lead to overfit or lead to difficulties in learning. The design of a optimal network architecture can be formulated as a search problem in the architecture space, where each point represents a possible network architecture [11].

One of the most important algorithms for the evolution of the topography of the networks has been the NEAT. Introduced by Stanley & Mikkulainen, [14] it showed the advantages of evolving the architecture simultaneously with the weights. The NEAT algorithm uses a direct encoding: every node and every connection is stored in the DNA. From this base, architecture researchers have built methods to develop evolutionary algorithms for architecture design.

Esteban et al. [15] proposed an evolutionary process to discover the architectures automatically. In their method, each individual consists in a trained architecture and during each evolutionary step, two individuals at random are chosen and compared by their fitnesses. Given the high computational expense, they developed a kind of grid computing solution, making the competitions in parallel, the computers in the architecture are called workers. This kind of scheme uses repeated pairwise competitions, which makes it an example of a tournament selection operator, also it prevents workers from idling when they finish early. The encoding used is slightly different than others: each neural network architecture is encoded as a graph. The vertex in the graph represents a rank-3 tensor (because of images, they always have three dimensions, the first two for the spatial coordinates and the third one for the number of channels) or activations. The mutations they chose were due to the similarity of the actions that a human designer may take when improving an architecture. Examples of mutations:

- Alter Learning Rate
- Identity (Keep training)
- Insert Convolutional Layer (Insert at random location)
- Alter stride (alter the stride of the convolutional layers)
- Filter size (horizontal or vertical at random)

To get a sense of variability they repeated the experiment 5 times and they concluded that their neuro-evolutional approach is capable of constructing large and accurate networks.

In these types of approaches, the population size is a crucial decision, given the fact that the search space is huge, bigger populations let the algorithm explore the space more thoroughly.

4.1.3 Hyperparameters Evolution

Hyperparameters are essential to the learning success of the networks. Although the research has proposed some standard values, they are often depending on the type of data and architecture you are working with. For example, some studies have shown that a set of hyper-parameters worked well in

simple networks but did not have the same effect in more complex architectures [16]. Evolutionary Algorithms have been proved to be effective for the search of these parameters.

Publication [17] has been the most inspiring for this thesis. In their work, they present a framework for optimizing the hyper-parameters of a deep convolutional neural network. In the evolutionary algorithm, each hyper-parameter is encoded in a single gene for each individual. To avoid obtaining undesired values for the genes they applied a range and a resolution. The parameters used in the experiment were the following:

Parameter	Value
N_{GENE}	6
$N_{GENERATIONS}$	35
$N_{INDIVIDUALS}$	500
p_c	0.6
p_m	0.05

Figure 4.2. EA parameters of the experiment.

Where Pc and Pm are respectively the probability of crossover and probability of mutation. The hyper-parameters they searched for, consisted of the kernel sizes of the convolutional layers and the number of filters. The fitness function chosen is simply the error on the test set for the dataset used computed after 4000 iterations. Although the framework is not too complex it has been proved to still be a more powerful tool than random search.

4.1.4 GAN-Specific Approaches

Most of the cited approaches could have been applied to different types of neural networks, in more recent studies some papers focused especially on GANs. In March 2018, Wang et al. proposed the first evolutionary generative adversarial network approach (E-GAN). The main purpose of their work was to stabilize GAN training and improve generative performance. In the original GAN, the Jansen-Shannon divergence is used as the metric, but more metrics have been introduced. They devised an evolutionary algorithm that evolves a population of generators $\{G\}$. To take advantage of different metrics E-GAN uses pre-defined objective functions that alternatively train the generator's weights. The evolutionary

step consists of 3 stages: variation, evaluation, and selection. One of the main contributions of E-GAN is the variation step where asexual reproduction along with mutations is used for the next generation's individuals.

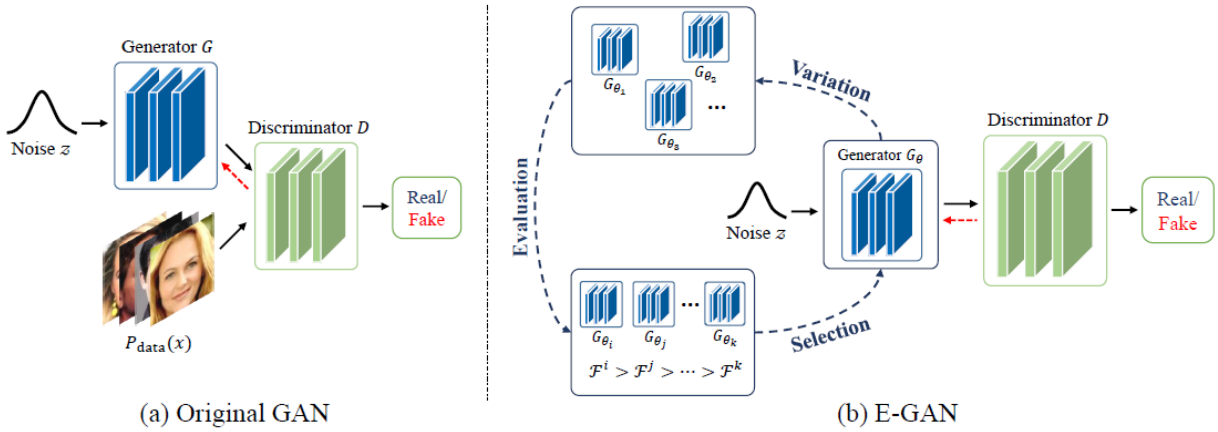


Figure 4.3. Original GAN compared with E-GAN approach.

In E-GAN the generator quality was compared with conventional GANs using FID score and demonstrated that E-GAN achieves convincing generative performance and minimizes training problems in conventional GANs.

Thereafter, later in 2018, Abdullah et al. proposed a spatial coevolutionary approach called Towards Distributed Coevolutionary GANs (Lipizzaner) [19]. In which, the researchers investigate the usage of coevolutionary algorithms with conventional GAN training. They aimed to bridge the gap between works of deep learning and evolutionary computing communities towards a better understanding of gradient-based and gradient-free GAN dynamics. In the spatial coevolution, GAN individuals are distributed on a grid where the local interaction of individuals governs the fitness evaluation, selection, and mutation.

Another GAN-specific interesting approach was studied by Roziere et al. [20]. They focused on the noise used for the generator. Instead of randomly generating a latent vector z, they used an evolutionary algorithm to select the best one without changing anything in the training of the network. They showed that the data produced by the latent noise generated through their algorithm, is of better quality while preserving the diversity of the original GAN.

5. METHODOLOGY

After looking at different possibilities on how to implement genetic algorithms with deep neural networks, in this chapter we are going to elaborate on the work done. First, we will describe the proposed approach, we will then elaborate on the dataset chosen and conclude with the instruments and techniques used, describing in detail both the fitness function selected and all the steps that compose the algorithm.

5.1 PROPOSED APPROACH

Our approach is focused on the improvement of the hyperparameters of a WGAN (previously explored in chapter 2.5). The method is similar to the one implemented by [17] but the hyper-parameters that have been taken into consideration are different. This difference led to other types of problems for the definition of the constraints for the genetic operators. The approach is based on a "standard" genetic algorithm that uses simple operators to achieve a better set of parameters. As not so many experiments have been done for WGANs with tabular data, we preferred using basic operators to see how much of an impact they would have had.

5.1.1 Hyper-Parameters Encoding

In our evolutionary algorithm, the encoding of the individuals consists of 6 genes that represent the actual values of the hyper-parameters we want to tune. The encoding is simple but effective.

```
[489, 0.00402, 0.84, 0.785, 4, 8]
```

Figure 5.1. Example of an Individual

The parameters that are being taken into consideration are respectively:

- Batch Size
- Learning Rate (The learning rate of the Optimizers)
- Beta 1 (The betas of the Optimizers)
- Beta 2

- Number of Critic Training (How many times more the Critic is being trained than the Generator)
- Weight of Gradient Penalty (The weight variable in the loss function)

This direct encoding seemed to us to be the most effective. Other kinds of approaches have been thought, but the complexity that they would have brought to the implementation of the algorithm was exceeding past the possible benefits.

5.1.2 Genetic Operators

The genetic operators that we have implemented are not too different from the ones we have talked about in chapter 3. Although we could have implemented some more complex ones, we decided to keep it as simple as possible knowing that good results could have been achieved without bringing unnecessary complexity to the algorithm. The main difficulty for our genetic operators has been the constraints that this type of encoding brought.

Genetic Operator	Method Implemented
Selection	Roulette Wheel
Crossover	Single Point
Mutation	Real Number mutation

Figure 5.2. Chosen Genetic Operators

Selection and Crossover are essentially the same. For the mutation operator, we had to implement a custom one. Bearing in mind that in the same individual there are different types of numbers (float and int) we needed to sum or subtract different values regarding the single gene. For each one, we decided on a different range of numbers for the single-point mutation. Hereunder are the different numbers used.

Parameter	Random Value Between
batch size	± (25, 50)
learning rate	± (0.00001, 0.005)
beta 1	± (0.05, 0.15)
beta 2	± (0.050, 0.150)
number critic	± 1
weight gp	± 1

Figure 5.3. Values used for Mutation

The decision on the range of values has been influenced by the common state-of-the-art parameters. We experimented with values outside certain ranges and understood that having numbers too far from the standard ones would not bring any benefit. In order to maintain the parameters inside a range, we implemented some constraints on the possible values that they could assume. Furthermore, the process of constructing the new population was characterized by Elitism. With this method, the best organism can carry over the next generation unaltered. This is done to guarantee that the solution quality obtained by the GA does not decrease with generations.

5.1.2 Fitness Function

The choice of the fitness function has been not trivial. As there is not a standard way to check the overall quality of samples, we have researched a metric that could give us the best results. One fitness function that was interesting and that we tried to use was the Frechet inception distance. Unfortunately, the latter one was a good fit for images but could have not been successfully applied to the tabular data we were dealing with. After having taken into consideration different loss functions, we have decided to choose the one that is most used in the research: the Kullback-Leibler (KL) divergence. This divergence, often called relative entropy is the measure of how one probability distribution is different from a second one. In our case, the KL divergence is calculated between the distribution of the real sample and the synthesized one. One thing to note is that the measure is asymmetric so switching the place of the distributions would lead to different results. Hereunder we can look at the formula:

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$$

In our case Q is the distribution of the synthesized sample while P represent the distribution of the original data. (The formula is for discrete distribution, in our case, as we are dealing with continuous data the relative entropy is defined to be an integral). What we are trying to see is how much the synthesized distribution is close to the original one and this explains the order of the distributions in the formula. After some trials, we decided to calculate this divergence after 300 iterations of the training of the GAN, which seemed the best choice to have a good trade-off between speed and quality of the measure.

5.1.3 Algorithm steps

The algorithm is divided into more steps. You first initialize the population and then you start to modify and evaluate individuals for more generations. As we have seen already, we have shown all the building blocks for the algorithm. We will now analyze how they are put one after another.

```
Genetic Algorithm
1. Initialize the first  population

2. For n generations (#1 loop)

    2.1. Repeat until the generation has reached the maximum population (#2 loop )
        1. Select the individuals via Roulette Wheel
        2. Apply Crossover with probability 0.5
        3. Apply Mutation with probability 0.1

    2.2. Replacement with Elitism

3. Return the best solution
```

Figure 5.4. Scheme of the algorithm steps

The first step in the algorithm is to initialize the initial population: you create some random individuals (set of hyper-parameters) and you then evaluate the quality of those based on the fitness function. Once you have a starting population you begin to iterate for the number of generations that have been priorly defined. In each generation, we perform a set of operators that will bring us to the next population. The first thing we do is the selection of a couple of individuals for the crossover and mutation step. This selection is done via the technique of the roulette wheel. Once we have selected the individuals the crossover step is applied with some probability. The same thing is for the mutation step that is applied after. Depending on these probabilities the crossover and mutation can be more or less frequent, the percentage of probability is one of the parameters that we can tune in the algorithm. Once we have these new individuals the calculation of the fitness score is performed again. At the end of the generation, we replace the original population with the new individuals applying the elitism replacement.

5.2 DATA

Most of the data that is used in the academy for GANs is image data. For the purpose of this thesis, we opted to use tabular data. Although it seems easier, the distributions of tabular data are usually way more complex than the ones from images, making the training of the GAN even more challenging.

The dataset we have taken into consideration is from the Kaggle challenge 'Give Me Some Credit' [21]. The dataset has been chosen because it contains only numerical numbers. As we are using a plain WGAN, the categorical variables would have been poorly learned.

	column_1	*column_2*	*column_3*	*column_4*	*column_5*	*column_6*	*column_7*	*column_8*	*column_9*	*column_10*	*column_11*
count	150000	150000	150000	150000	150000	120269	150000	150000	150000	150000	146076
mean	0.067	6.048	52.295	0.421	353.005	6670.221	8.453	0.266	1.018	0.24	0.757
std	0.25	249.755	14.772	4.193	2037.819	14384.674	5.146	4.169	1.13	4.155	1.115
min	0	0	0	0	0	0	0	0	0	0	0
25%	0	0.03	41	0	0.175	3400	5	0	0	0	0
50%	0	0.154	52	0	0.367	5400	8	0	1	0	0
75%	0	0.559	63	0	0.868	8249	11	0	2	0	1
max	1	50708	109	98	329664	3008750	58	98	54	98	20

Figure 5.5. Basic Statistics of the Data (the column names were replaced because of length).

Is it possible to see that the data is all numerical, although some columns are discrete and others are continuous. Also, for the purpose of the experiment, we did not apply too much pre-processing. A couple

of steps were although needed. The data contained around a certain amount of Nan values, in order to work with it and train the algorithm we decided to drop them. The step right after was the standardization of the data given the fact that all the variables were in different scales. Furthermore, there has been an attempt to improve the training identifying outliers, as we can see in *column_5* and *column_6* there are probably some values that are way out of the standard deviation that could be considered outliers. Based on this assumption, the GANs were trained two times. Once without removing the outlier and once removing them. After these records were removed the quality of the generated data has instantly improved.

5.2.1 Data distribution

The distribution of the data varies a lot depending on the variable we take into consideration. Some of them are more gaussian-like whilst others show very skewed distributions. As we did not try to standardize the distributions (i.e., transforming them as gaussian as possible) it is expected that the distributions of the columns that the WGAN needed to learn were different from each other.

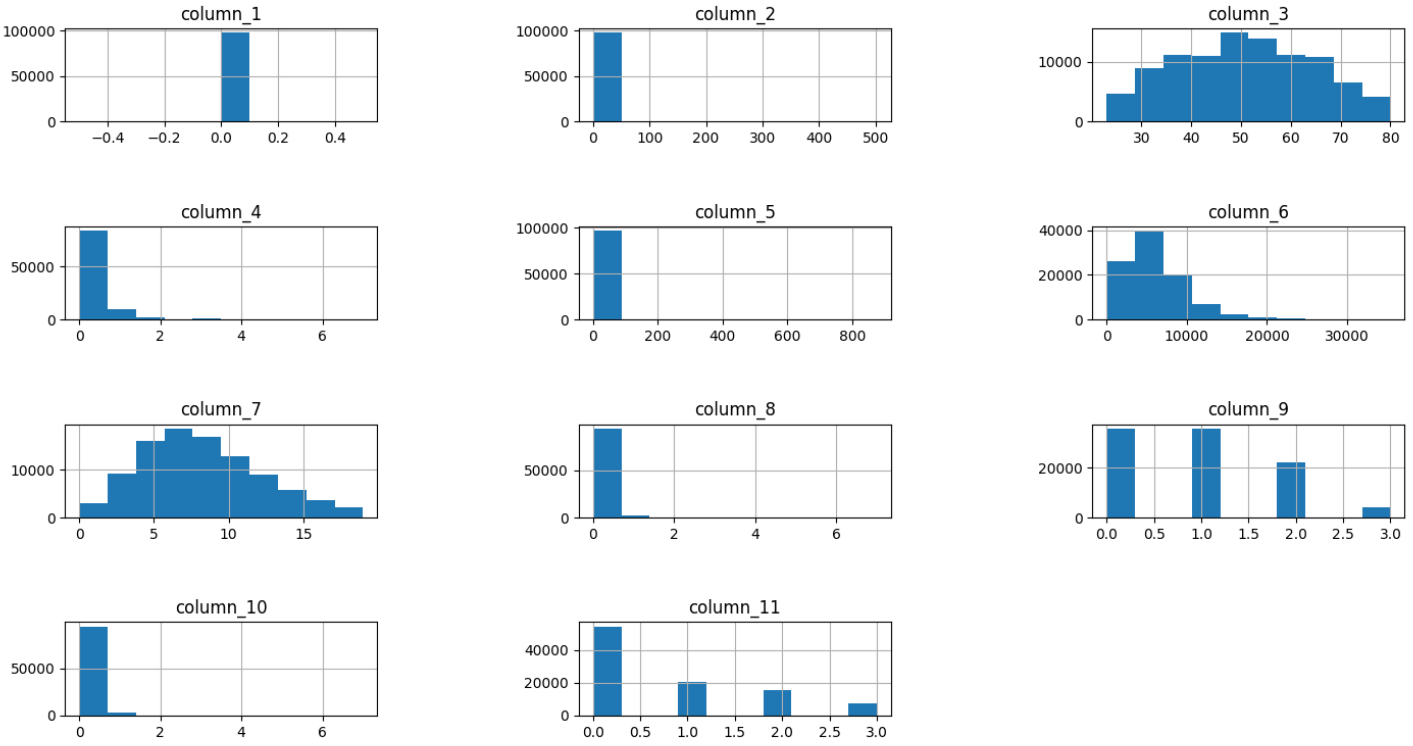


Figure 5.6. Distributions of the variables.

The differences in the distributions can be observed in figure 5.6. Other than the distributions another aspect of the data was highlighted by the graphs. Is possible to observe that some of the columns have few discrete values (*column_9* and *column_11).* As the WGAN cannot directly work with discrete data we need to tackle this problem creating some constraints. The WGAN in its vanilla form can only generate data that have decimals, so for these variables to be discrete, we will round up to the whole number. This kind of post-processing would have not been needed if some other GAN structures would have been used. More advanced architectures are implemented with the possibility of choosing, before the training, constraints for the generation of the data. Also, some other architectures were built exactly thinking of the categorical variables like the CTGAN [23].

5.3 EXPERIMENTAL SETUP

5.3.1 Equipment

The computer used for this experiment had an Nvidia GeForce RTX 2070i and the run of the algorithm took about 12 hours. We have also tried some cloud alternatives like Google Colab, but unfortunately, due to the long process, it was not possible to take advantage of this tool.

5.3.2 Neural Networks Architectures

Given the fact that we have used a simple WGAN, the structure was composed of only two Neural Networks. (In more complex structures it is possible that the number of neural networks increases. There are often neural networks for the encoding and decoding of the data giving better performances in terms of training.) In both generator and critic, given the data was tabular we decided on using dense layers. This architecture was chosen given that the data taken into consideration was tabular. We chose not to use too many layers to keep the structure simple and to avoid overfitting. In case we had more complex data to deal with, the decision would have been different. The structures are not composed of many layers for the same reasons. Hereunder is possible to see the two networks utilized.

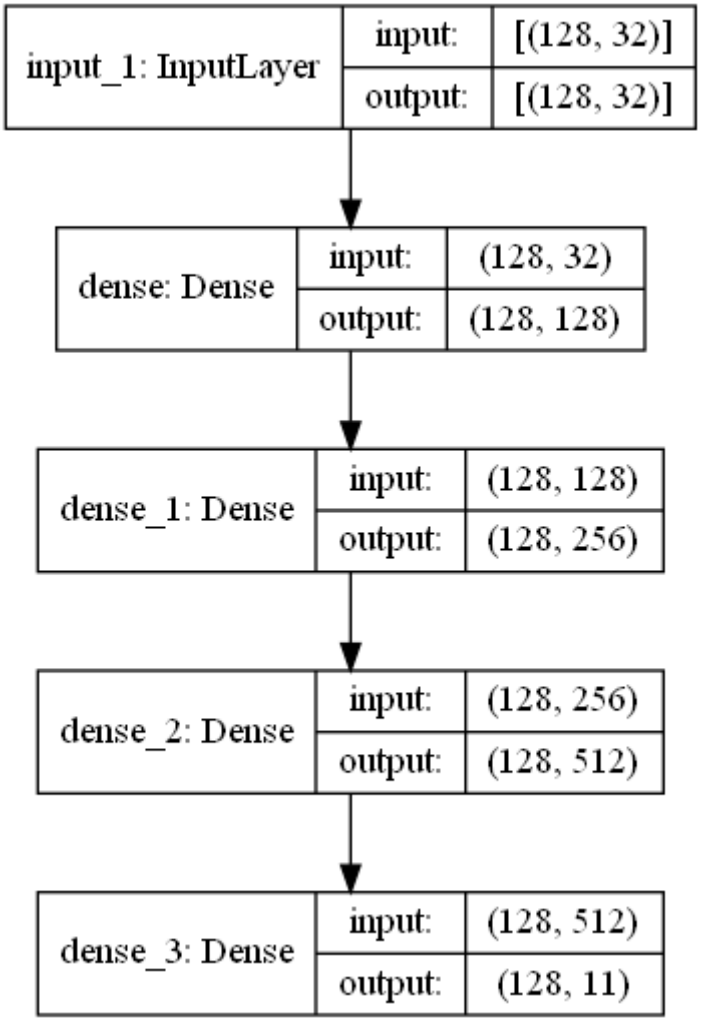
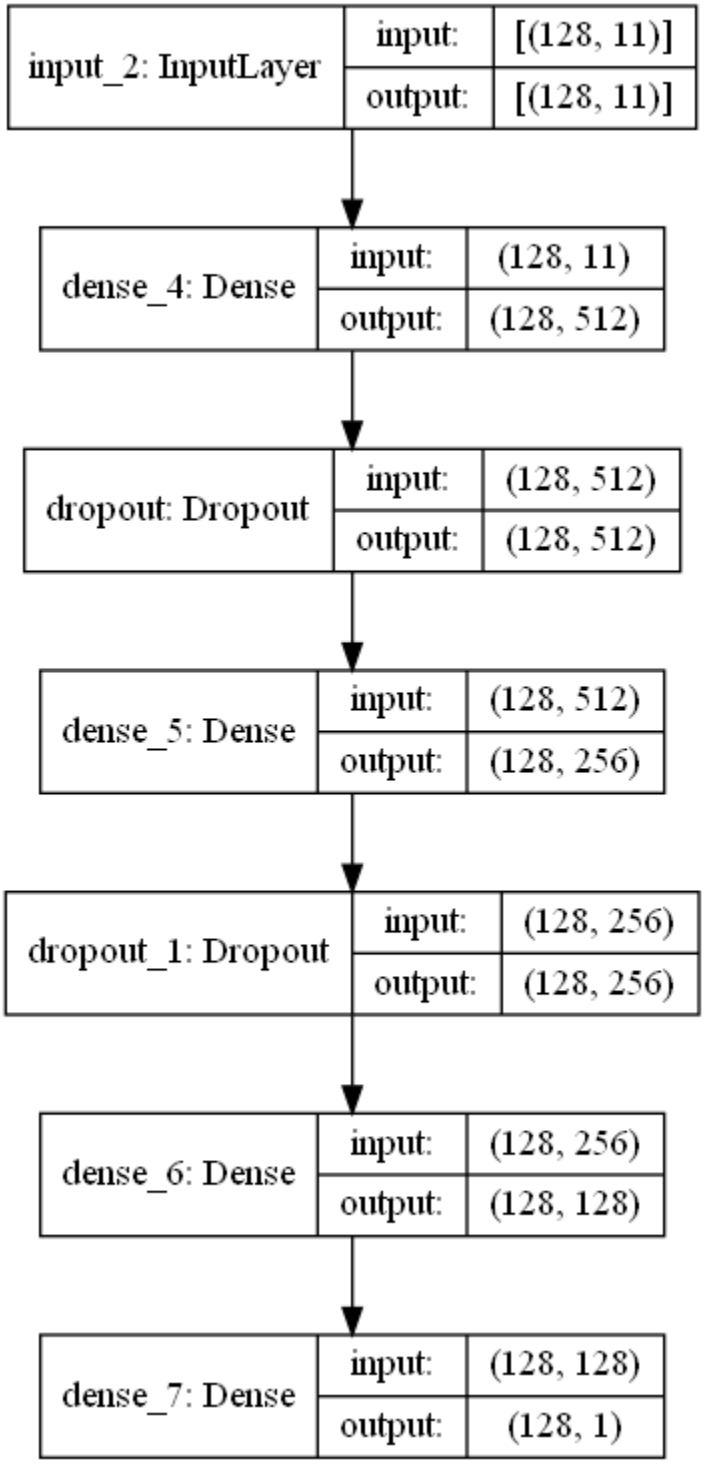


Figure 5.7. Generator used

As we can see from figure 5.7 the Generator is composed of 4 dense layers with an input of a vector of 32 (noise) and an output of a vector of size 11 that corresponds to the number of columns of the dataset. As activation function, all of them are using a 'relu'. The sizes of the various layers can be changed and adapted to the data you are working with.



.

Figure 5.8. Discriminator used

Although it could seem deeper, the discriminator has the same number of layers.

The only difference with the generator is that we have used Dropouts to slow down the learning process of the discriminator bearing in mind that it is usually trained more times than the generator.

5.3.3 Parameters of Genetic Algorithm

Multiple parameters have been tried out, but the final choice for the best trade-off between time and good quality of the experiment were these ones:

- Population size: 250
- Probability of crossover: 0.60
- Probability of mutation: 0.10
- Number of Generations: 20

Also, the decision has been influenced by the academy that often advises using a little percentage for mutation and a bigger one for the crossover. In the beginning, we have tried different probabilities for the operators. In the case of larger values, the individuals result in having a larger diversity, although it is beneficial to explore the full search space, you often end up getting further away from the local optima. One way to implement the benefit of the diversity given by the high probabilities of the operators would possibly be to implement a decay rate. In case a decay rate is set, the percentages would get lower and lower the more the generations pass by. This would lead the algorithm to explore better the search space at the beginning and once it reaches different local optima it would iterate close to them. Another constraint that we have was the population size and the number of generations, as it gets computationally more expensive, we could have not incremented these values that might have had helped the search with high operators' probabilities.

6. RESULTS

In this chapter, we are going to illustrate the results obtained with the best parameters found from the genetic algorithm. At first, we are going to illustrate how we decided to measure the quality of the generated data and the reasons why we have chosen those metrics. We will then proceed on showing the performance obtained by comparing the synthetic data with the original one. The performances of the best-parameters dataset are compared with the state-of-art parameters dataset.

6.1 METRICS USED

Understanding how good the quality of the generated data is not trivial and until now, although a lot of metrics have been developed, there are no grounded ways to assess the data quantitatively and

qualitatively [22]. Unfortunately, as for the fitness function, there is not a one-fit-all solution. This means that is impossible (to date) to measure the quality of a dataset with just one metric. So, to determine the quality of our data we decided to use 3 metrics that overall cover the different aspects of the data. Of course, more metrics could have used, but that depends a lot on what it is the end goal of the synthesisation. In our case, we are just looking forward to comparing how the different generated datasets are, in terms of similarity to the original data.

6.1.1 Basic Statistics

The first thing we thought of was that the generated data should have had similar behaviour to the real data, so we expected all the basic statistics to be close to each other:

- Mean
- Standard Deviation
- Percentiles
- Maximum
- Minimum

The main reason why you want to see how close the basic statistics are, is to see if the generated values are kept in the expected ranges. Sometimes, during the generation of synthetic data, you could end up with values that do not reflect reality. For example, a lot of time your data cannot have values under zero or values that exceed a certain threshold. From looking at the maximum and the minimum you can already have a grasp of the range of your generated data. To avoid this, it is often implemented the use of constraints on the range of the possible values that your variable can assume (could be implemented either on the post-processing as we did or on the algorithms themselves). On the other hand, the mean, standard deviation, and percentiles give you a further understanding of how the values are distributed along with the dataset, thus if you end up with some kind of mode collapse. To calculate how similar these basic statistics were to the original data we decided on comparing them calculating the pairwise Euclidean distance between the statistics. In order to do so, two data frames were created: one containing all the aforementioned statistics of the real data, and one of the generated data. These datasets were rapidly created utilizing the function ‘describe’ of pandas. One drawback of this metric is that the resulting value is not scaled. This implies that there is not a universal way of interpreting the results as it could be for some metric in a specified range (i.e., 0 to 1). In our case, as we are using it just

to compare different datasets gives us just the general idea of how good one sample is in respect to another. As it is a distance, the smaller the value, the closer the generated data to the original one.

6.1.2 Distributions

Another aspect that was worth calculating was the distribution of each variable. Although looking at the statistics gives us a general idea of how the data is behaving, some tests give us a deeper understanding of what concerns the distributions. To see the similarity between the distributions, we decided to use the following two statistical tests: Chi-Squared and Kolgomorov-Smirnov.

The Chi-Squared test is used to compare distributions of discrete or categorical variables. The test returns the resulting p-value so that a small value indicates that we can reject the null hypothesis and suggests that the distributions are different.

The Kolmogorov–Smirnov test is used to compare the distributions of continuous variables using the empirical Cumulative Distribution Function (CDF). It returns 1 minus the KS Test D statistic, which indicates the maximum distance between the expected CDF and the observed CDF values.

Both the tests output a value between 0 and 1: if the distributions are identical, you get the maximum value (1), in case they are completely different you would get the minimum value (0). As both of them have the same range, we decided to combine them by calculating their mean. As opposed to the latter metric, this would be useful to give you information on the dataset without having to compare it to others.

6.1.3 Correlation of Covariances

Looking at how the data is distributed and if it has the right values does not mean that the data generated is of high quality. As you are often generating more than one column you expect your generated data to keep the same relations between the variables of the original data. This is needed because the generated data is often used to solve some machine learning tasks, and in order to do so you need to keep all the possible information. To understand the relationship that exists between the variables we have decided to look at the covariance matrix. As the covariance is the joint variability of two random variables, the matrix would give us all the info we need. To calculate the similarity between these matrixes, once again, the pairwise Euclidean distance has been used.

6.2 FINAL RESULT

The best parameters found from our algorithm for the final individual were the following:

Batch Size	Learning Rate	Beta 1	Beta 2	Number Critic	Weight GP
375	0.00217	0.9	0.76	3	9

Figure 6.1. Best Parameters Found

This individual reached a fitness function of 0.7854 meaning that the distributions of the real and synthetic samples were very close to each other. From the parameters found we can see that the ones that are the farthest from the state-of-art ones are learning rate and beta 2. This means that are possibly the ones that influence more the result of the WGANs. The other parameters, although not equal, are close to the standard ones. One of the reasons why they are so close could be the fact that we put some constraints on the possible values they could have taken, but at the same time, I highly doubt that going too far from the standard ones would have led to optimal results.

From the final comparison of the real data with the synthetic one generated by the best individual, we got these results.

	distance_distribution	distance_statistics	distance_correlation
genetic_parameters	0.580	20151.221	4.569
standard_parameters	0.162	866108.729	4.656

Figure 6.2. Results of the experiment

As we can see from the table, the parameters found by the genetic algorithm resulted in having better data. The distributions are more similar and the distances between statistics and covariances are smaller. The results are not outstanding in terms of difference, for example, we can see that the distances of the correlation are close to each other. As we already discussed, it is not strange to have values that are not so distant, as the parameters are not too different. Another reason why it might have not been an excellent result is that Neural network models are stochastic. This means that, given the same model configuration and the same training dataset, a different internal set of weights will result in each time the model is trained that will in turn have a different performance. The data has been generated using the best parameters but not the exact model used during the research of those.

7. CONCLUSIONS AND FUTURE WORK

In this thesis, we have shown that although the quality of the best parameter's data did not surpass by far the quality of standard parameters, the proposed genetic approach is better than the normal random search. We have to take into account that there has not been done too much work on the pre-processing of the data, maybe approximating the data to a gaussian-like distribution would stabilize the training of the network and let focus the networks on the hyper-parameters themselves. Furthermore, we have proposed a comparison method that provides an overall view of the quality of the generated samples.

Future improvements can be achieved in a couple of different areas of the project. First, we did not have enough computational power to run big populations, and as we know that the search space is immense, having more individuals in the population would already bring better results. The genetic operators that have been used were basic, so as we saw from the literature review, more complex operators would bring better individuals. Another improvement that can be done is on the fitness function; as there is not a universal metric for the quality of the produced data, using different fitness functions could bring better results, as it has been done on [18]. Other than working on the genetic algorithm itself some more work can also be done on the GAN architecture having some more stable networks could bring benefits to the calculation of the fitness function.

REFERENCES

[1] Ian J. Goodfellow, J. P.-A. F. (2014), Generative Adversarial Networks,

https://arxiv.org/abs/1406.2661

[2] A. Aggarwala, M. Mittal, G. Battineni, (2021), 'Generative adversarial network: An

overview of theory and applications'

[3] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. Tygar, 'Adversarial machine learning', ACM Workshop on Security and Artificial Intelligence, pp. 43–58, 2011.

[4] J. Gui, Z. Sun, Y. Wen, D. Tao, J. Ye, (2020), 'A review on Generative Adversarial Networks: Algorithms, Theory, and Applications'

[5] M. Arjovsky, S. Chintala, L. Bottou, (2017), 'Wasserstein GAN'

[6] I. Gulrajani, F. Ahmed, M. Arjovsky, V Dumoulin, A. Courville, (2017), 'Improved Training of Wasserstein GANs'

[7] Wikipedia, 'Lipschitz continuity', en.wikipedia.org/wiki/Lipschitz_continuity

[8] A. E. Eiben, J. Smith, (2015), 'From evolutionary computation to the evolution of things', Nature

[9] J. H. Holland. (1975), 'Adaptation in Natural and Artificial Systems', University of Michigan

[10] K. Tang, X. Yuan, P. Liu, J. Yang, (2011), 'Linear Evolutionary Algorithm', IntechOpen

[11] P.A. Castillo, M.G. Arenas, J.J. Castillo-Valdivieso, J.J. Merelo, A. Prieto, G. Romero, (2003), 'Artificial Neural Networks Design using Evolutionary Algorithms'

[12] Y. Chang, J. Lin, J. Shieh, M. F. Abbod, (2012), 'Optimization the Initial Weights of Artificial Neural Networks via Genetic Algorithm Applied to Hip Bone Fracture Prediction', Hindawi Publishing Corporation

[13] D. Orive, G. Sorrosal, C. E. Borges, C. Martin, A. Alonso-Vicario, (2014), 'Evolutionary algorithms for hyperparameter tuning on neural networks models'

[14] K. O. Stanley, R. Miikkulainen, (2002), 'Evolving Neural Networks through Augmenting Topologies', Massachusetts Institute of Technology

[15] E. Real, S. Moore, A. Selle, S.Saxena, Y. L. Suematsu, Jie Tan, Quoc V. Le, A. Kurakin, (2017), 'Large-Scale Evolution of Image Classifiers', International Conference on Machine Learning, Sydney

[16] T.M. Breuel, (2015), 'The effects of hyperparameters on SGD training of neural networks'

[17] S. R. Young, D. C. Rose, T. P. Karnowski, S.H. Lim, R. M. Patton, (2015), 'Optimizing Deep Learning Hyper-Parameters Through an Evolutionary Algorithm'

[18] C.Wangy, C. Xuz, X. Yao, D. Taoz, (2018), 'Evolutionary Generative Adversarial Networks'

[19] T. Schmiedlechner, I. Ng Zhi Yong, A. Al-Dujaili, E. Hemberg, U. O'Reilly, (2018), 'Lipizzaner: A System That Scales Robust Generative Adversarial Network Training'

[20] B. Roziere, F. Teytaud, V. Hosu, H. Lin, J. Rapin, M. Zameshina, and O. Teytaud, (2020), 'EvolGAN: Evolutionary Generative Adversarial Networks', arXiv:2009.13311v1

[21] https://www.kaggle.com/c/GiveMeSomeCredit/

[22] A. Borji, (2018), 'Pros and Cons of GAN Evaluation Measure'

[23] L. Xu, M. Skolaridou, A. Cuesta-Infante, K. Veeramachaneni, (2019) 'Modeling Tabular Data using Conditional GAN', arxiv:1907.00503