UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

DIPARTIMENTO DI INGEGNERIA MECCANICA E INDUSTRIALE

Corso di Laurea Triennale
in Ingegneria Meccanica e dei Materiali

Relazione Finale

TOWARD AI-ASSISTED DISCOVERIES IN SCIENCE

Relatore:
Chiar.mo Prof. Davide Pagano

Laureando:
Matteo Albini
Matricola n. 736135

Anno Accademico 2023/2024

Abstract

Artificial Intelligence (AI) has undergone a major evolution in the last decade. Thanks to the introduction of new algorithms and the development of new Artificial Neural Network (ANN) architectures in particular, it has proved to be an essential tool for scientific research. The main evolution lies in the solution proposed regarding the generative problem, and Generative Adversarial Networks (GANs) represent one of the best. Their use in science, coupled by the implementation of nature-inspired metaheuristic algorithms for optimization, is the first step towards the use of AI as a guide in exploring the space hypothesis. For this reason, this work focuses on the use of GANs, based on the generator-discriminator duality, and the Genetic Algorithm, Particle Swarm Optimization and, Whale Optimization Algorithm, whose optimization strategies are described and compared. Human limitation thus forces the use of such techniques to advance the frontier of knowledge. However, scientific development requires attempts to consolidate the acquisition of new notions. The one presented in this thesis concerns the learning by an ANN of the properties that characterize the fundamental operations and has as its starting point the use of GANs for the generation of meaningful equations. The completion of this task would allow a revolution in the field of research and thus the beginning of a new era.

Table of Contents

Introduction

Over the past decade, AI has taken an unprecedented turn in development and emerged as a key tool in many diversified fields, most importantly scientific research. At the backbone of such advancement are generative models, more precisely GANs, which may exhibit great potential in data creation and modeling complex systems. GANs introduced a new horizon in unsupervised learning with their exclusive form of generator-discriminator network and have entered newer dimensions of machine learning applications. Their success is further complemented with the use of optimization techniques like nature-inspired metaheuristics that offer robust solutions to the so-called challenging problem of optimizing neural network architecture.

The application of AI to scientific research goes well beyond conventional data analysis, and AI-driven models have now begun to actively investigate new hypotheses and uncover deeper patterns of complex systems. Such a capability for unveiling relations beyond the inspirations of human intuition holds huge promise for discovering new natural laws and, therefore, an improvement in theoretical knowledge in many scientific fields. The most important tasks in this process involve training neural networks, which in the main will boil down to an optimization problem-finding the best structure along with model parameters of the network so that meaningful results are obtained. In the context of this process, nature-inspired metaheuristic algorithms have generally been found among the most useful tools for network training and optimization.

This thesis will discuss, with special emphasis on how GANs is driving scientific research forward. The study also covers the challenges that arise due to optimization in the training phase of such networks, along with the metaheuristic algorithms that help overcome such situations. In particular, Chapter 4 gives a case study of how these concepts work when put into real-world applications, demonstrating that AI could discovery new laws governing natural phenomena. Nevertheless, every research has its own limitations and hindrances. The major one among them will be finding the right architec-

ture of the network such that new insights could be generated or a new law could be discovered. This means that further experimentation with strategies and approaches has to be done in order to ensure that the network functions properly.

The work is organized as follows. Chapter 1 provides the basic concepts of ML and ANNs . Initially, it introduces ML by giving its history and the types of learning that can be achieved through this approach: supervised, unsupervised, and reinforcement learning. Then it covers the architecture of ANNs from simple, the TLU, to a complex model of multilayer perceptron networks by emphasizing their capability for learning from experience and solving the classification problems.

Chapter 2 presents the recent development in machine learning known as Generative Adversarial Networks. This chapter illustrates the dual role played by generator and discriminator that GANs have adopted and subsequently how they generate realistic data by training the generator properly to fool the discriminator. Further, it presents several applications of GANs, from image synthesis to image-to-image translation.

Chapter 3 focuses on nature-inspired metaheuristic algorithms, important optimization methods of neural network training. The specific main algorithms treated in detail are Genetic Algorithm, PSO, and Whale Optimization Algorithm. It explains their biological inspiration, the mechanics behind each algorithm, and their respective strengths in solving optimization problems within large and complex spaces.

Chapter 4 describes a case study that illustrates concrete application via AI and GANs in scientific discovery. This covers the application of symbolic regression to explore hypothesis spaces and deliver meaningful equations. The case study also intends to show how the techniques reviewed above can be useful in the process to find new algebraic structures, and perhaps uncover new natural laws.

Chapter 1

Machine Learning and Neural Networks

In this chapter, Machine Learning (ML) is introduced, and its development through the centuries is described. Furthermore, Artificial Neural Networks (ANN) are presented, from the simplest to more complex structures, in comparison with biological neurons.

1.1 Machine learning development

Machine learning is a branch of Artificial Intelligence (AI) and represents all the techniques used to improve system performance by learning from experience. It has many different definitions due to its multifaceted range of types, but the one formulated by Tom M. Mitchell[1] is considered the most relevant:

> ”A computer program is said to learn from experience E with respect to some class of tasks T and the performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.”[1]

In order to explain, reference is made to the context of image classification, where E represents the past data with images having labels or assigned classes (for example, whether the image is of a class car or a class bike), T is the task of assigning class to new, unlabeled images, and P is the performance measure indicated by the percentage of images correctly classified.

[1] M. Mitchell, Professor of Machine Learning Department, School of Computer Science, Carnegie Mellon University,

In order to understand the theory behind ML and ANN, it is useful to take a historical overview of their development. The roots of Machine Learning go back to the 18th century, when Thomas Bayes article titled ”An Essay towards solving a Problem in the Doctrine of Chances” was published. It describes many important algorithms on which machine learning is based and establishes the basis of statistical learning [2]. Then, in the 19th century, other theorems and methods were formalized, including the least square error method, stated by Gauss and Legendre, and the Bayes theorem. In 1943, one of the most famous techniques of Machine Learning, called Artificial Neural Networks, was born. Warren McCulloch and Walter Pitts proposed the Threshold Logic Unit (TLU), which was an artificial neuron targeted as a computational model of the nerve net in the brain. The main drawback of this model was that its performance was based on fixed parameters; hence, it made it impossible to learn from experience. The effective foundation of machine learning is considered to have been in 1950, when Alan Turing, in his paper ”Computing Machinery and Intelligence”, wondered if machines could learn as humans and proposed the first idea of artificial intelligence [3]. The decisive step was taken in the late 1950s, when the first learning machines were developed. The first dates back to 1954 and was made by Marvin Minsky, while the second was created by Frank Rosenblatt in 1958, who designed the Perceptron model. The learning law that regulates its operations was demonstrated to converge for pattern classification problems only if the features are linearly separable. However, it was proven that multilayer perceptron structures could handle any classification task, but there was not any algorithm able to regulate the behavior of the net throughout experiences. Lack of suitable learning laws for multilayer perceptron networks put the brakes on Machine Learning and Artificial Neural Network development for a period called AI winter. The loss of attraction for this discipline and AI in general was due to a lack of resources powerful enough to deal with the complications that arose. It was needed to wait until the new century to reconsider how powerful Machine Learning was. From this point on, many algorithms have been developed, and the potential of machine learning became clear [1][4].

1.2 Types of Machine learning

As it became clear, the main principle of Machine Learning is learning from experience. Based on the teaching method, three categories of learning can be distinguished: *supervised*, *unsupervised*, and *reinforcement* [5].

1.2.1 Supervised Learning

In Machine Learning, it is called supervised learning, or predictive learning, the technique that aims to learn a good approximation $\hat{f}$ of the true mapping f from the input vector $\overrightarrow{x}$ to the outputs $\overrightarrow{y}$, using information contained in a dataset of examples, generated either through the direct observation of the phenomenon under analysis, either by performing experiments. The process is composed of two steps: ”training” and ”test”. Using labelled data, it is provided as input a part of the dataset, which allows to reconstruct $\hat{f}$ by iteratively minimizing a predefined cost (or loss) function, whereas the second one is used to assess the prediction accuracy of the model, on data that were not used during the training phase. Output variables may be either categorical, if the problem is a classification task, or continuous, if it is supposed to solve a regression task. For example, in an industrial context, it could be a regression problem to predict long term performance of a mechanical machine under certain conditions. Otherwise, it is possible to recognize surface defect type using image scanning of the product.

1.2.2 Unsupervised Learning

As regards unsupervised learning, it concerns unlabelled data, and therefore, the objective is to detect patterns in data, not make predictions. Indeed, unsupervised learning is often referred to as descriptive learning and is associated with pattern discovery techniques such as Hierarchical Clustering or K-Means. Usually, it could be divided into three sub-areas[2]: *clustering*, *density estimation*, and *dimensionality reduction*. The first one aims to section the data and bring together similar elements. An application of this method has led to the revision of taxonomies, previously based on appearances, finding genetic similarities and dissimilarities between different species. Density estimation is indeed composed of the techniques that allow for the reconstruction of the probability density function using a set of

[2]Murphy, K. (2012). Machine learning: A probabilistic perspective. Cambridge: The MIT Press

given data points. Finally, density reduction is useful in order to reduce the dimension of big-data without compromising its content. The most famous way to accomplish this task is using Principal Component Analysis (PCA), but neural network structures like autoencoders are also employed.

1.2.3 Reinforcement Learning

Reinforcement learning is a learning method based on interaction with the environment. It does not require a starting dataset, as it is free to interact autonomously in the different instances. Each situation allows to expand the repertoire of appropriate responses and to know what characteristics distinguish each space. It is possible to modify the performance by imposing policies and granting a reward, depending on the quality of the response provided. Thus, it is permitted to modify the choice parameters from instance to instance, making the behavior of the model as appropriate as possible for the current scenario. For example, reinforcement learning is applied in marketing, personalizing advertisements based on community interests.

1.3 Artificial Neural Networks

Artificial Neural Networks constitute one of the most commonly used Machine Learning technique and are an important resource in various scientific fields. Indeed, they are exploited in the statistical analysis area to solve classification problems or forecasting. ANN turns out to be an alternative to the classic algorithm of Machine Learning, thanks to its structure, defined *parallel distributed computing*. As regards neuroscience, Artificial Neural Networks constitute a computational model of biological brain development, given the close analogy between artificial and animal neurons. Moreover, they are widely applied to studying nonlinear dynamic systems, statistical mechanics, and automata theory [6].

As Artificial Neural Networks are a computational technique with high potential, the number of tasks they are capable of solving is constantly increasing. What this thesis focuses on is the use of ANN to discover new laws governing nature and solve conjectures that humanity has not yet been able to prove. Before deepening this argument, it is necessary to understand what they are and how they work.

1.3.1 Threshold Logic Unit (TLU)

As it was anticipated, an analogy between Artificial Neural Networks and the biological brain exists, and it is observable starting with the single element of the structure: the neuron.

Beginning with the natural ones, they are arranged in a framework known as *conncetionist*, which allows links to be generated. The communicative method is based on short-lived electrical signals, named *spikes*. The signal magnitude is mediated by *synapses*, which are electrochemical junctions located at the head of the connecting branches of neurons, called *dendrites*. Depending on the strength of the synapses, a signal may be excitatory or inhibitory, and consequently, it has a different impact on the extent of the total impulse annealed by the neuron. This is crucial since the generation of a response depends on a threshold value, which, if exceeded, leads to the generation of a response. In such a case, through a structure called *axon*, a signal would be transmitted to another neuron, allowing the transfer of information.

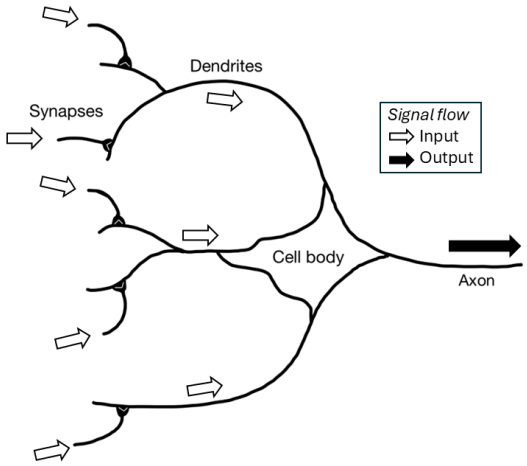

Figure 1.1. Essential components of a neuron shown in stylized form.

What has just been said can now be reflected in an artificial neuron, called *node* or *unit*. In order to focus attention on analogies, it is considered the most simple version of an artificial neuron, the Threshold Logic Unit (TLU). As the name suggests, even the TLU is based on a threshold value from which the answer depends. Due to the simplicity of this model, the output signal x_i is often boolean or binary. As far as the modulatory effect done by synapses, in this case it is done by weighing each input signal, that is, multiplying the

signal by a value called weight w_i, and combining them by simply adding them together. Thus, it is possible to calculate the activation value a

$$a = w_1x_1 + w_2x_2 + ... + w_nx_n. \tag{1.1}$$

The threshold value θ of the TLU can be represented graphically as a step, so that the relation between input and output is called *step function* or *hard-limiter*.

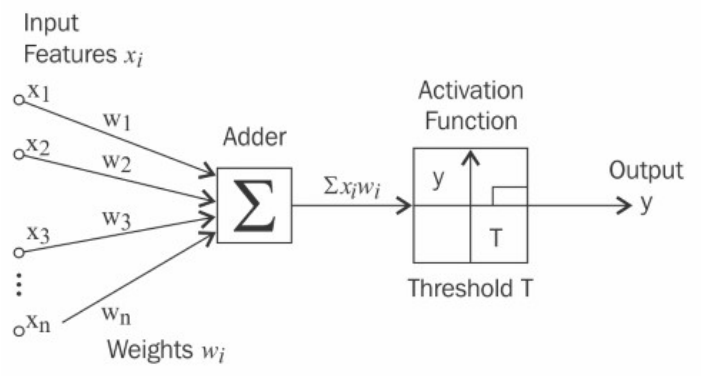

Figure 1.2. Activation-output threshold relation in graphical form.

Using this type of activation function, a significant problem is incurred: by linking several units together, there is no possibility of continuously graded signals occurring. To overcome this obstacle, another function can be used as an alternative to the hard-limiter. An example is the logic function sigmoid (or just sigmoid), which is characterized by a softening of the step. Graphically, it is represented by the curve in Figure1.3.

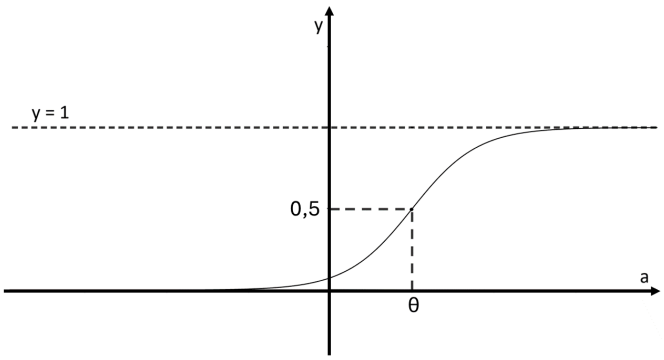

Figure 1.3. Example of squashing function—the sigmoid.

It can be seen that as a tends to largely positive values, the output tends more and more to 1 but never reaches it, and similarly, the smaller the activation value, the more the output tends to 0. In more detail, the function is described by the formula

$$y(a) = \frac{1}{1 + e^{-(a-\theta)/\rho}}, \tag{1.2}$$

where ρ determines for large values a flatter curve, while it makes the curve more steep for small values, and finally θ corresponds to the reinterpretation of threshold.

1.3.2 Linear separability

What TLU does is separate the inputs into two classes: those that give output 1 and those that give 0. Considering, for example, the case where each input pattern has two components x_1, x_2, it is therefore displayable in a two-dimensional space called pattern space, and $w_1 = 1$, $w_2 = 1$, $\theta = 1.5$.

x_1	x_2	Activation	Output
0	0	0	0
0	1	1	0
1	0	1	0
1	1	2	1

Table 1.1: Functionality of two-input TLU example.

Each pattern can assume a small range of values, as represented in Table 1.1, and by using its component values as space coordinates, they identify located points in the pattern space.

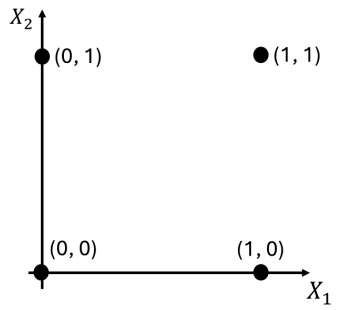

Figure 1.4. Two input patterns in pattern space.

Considering now the particular case where the activation equals the threshold, that for two input example gives

$$w_1 x_1 + w_2 x_2 = \theta. \tag{1.3}$$

Through simple operations it gives the formula of a straight line

$$x_2 = -\left(\frac{w_1}{w_2}\right) x_1 + \left(\frac{\theta}{w_2}\right), \tag{1.4}$$

where $\left(\frac{w_1}{w_2}\right)$ and $\left(\frac{\theta}{w_2}\right)$ are constants. By representing this decision line in the pattern space taken as an example, a separation of the cases into two classes takes place.

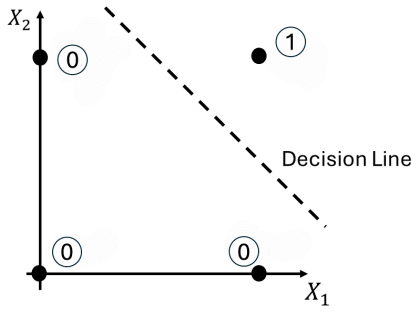

Figure 1.5. Decision line in two-input example.

Considering a more complex environment, the separation is made by other functions: the decision plane in a three-dimensional space, the hyperplane in a multidimensional context. The fundamental concept behind the above is that the TLU is a linear classifier and that the patterns are linearly separable. This also defines the limit of the TLU, namely that if patterns are not linearly separable, then they cannot be classified by the TLU.

1.3.3 Perceptron learning algorithm

In order to understand how the network is able to learn to give more and more accurate answers even if patterns are non-linearly separable, it is necessary to investigate how weights, thresholds, and input are interpreted in a multidimensional space. In this context, the activation a of an input is expressed as

$$a = w \cdot x, \tag{1.5}$$

where w and x are two dot multiplied vectors. Similarly, the threshold can be expressed in the same way. When calculating the projection of a generic input x onto w, an important property about the decision hyperplane is brought to light: it is always orthogonal to the weight vector. Addressing the core of the speech, it is referred to as the perceptron, i.e. the node related to TLU, and supervised learning is the technique used. An artificial neural network is able to solve a task when the decision surface classifies the data in the desired way. In order to modify the decision function, an iterative process of sifting through task-related data takes place. It allows the weights and thresholds to be modified so that the response is increasingly aligned with the desired one. The learning rule imposes on the network the method by which the parameter correction has to be carried out. The parameter changes are regulated by the formula

$$w' = w + \alpha(t - y)v, \tag{1.6}$$

where w and w' are the original and changed values of the weight, t is the desired response, and v is the input vector. α is a value between 0 and 1 and is called *learning rate* because it influences how much the network can learn from each iteration. As far as the threshold θ is concerned, it can be interpreted as a weight associated with a constant input of -1. This leads to the negative of the threshold being referred to as *bias*. Geometrically, it consists of rotating the weight relative to the input by a certain angle, regulated by the parameter α, to obtain the desired classification. The equation (1.6) defines the perceptron learning rule, and embedding it in an iteration scheme brings to the perceptron learning algorithm. Until now, it has been taken for granted that a hyperplane exists to perform the desired classification, but it is not always possible to fulfill every task. The perceptron convergence theorem explains this concept and states:

> "If two classes of vectors X, Y are linearly separable, then application of the perceptron training algorithm will eventually result in a weight vector w_0 such that w_0 defines a TLU whose decision hyperplane separates X and Y."

The convergence theorem was first proved by Rosenblatt (1962), while more recent versions may be found in Haykin (1994) and Minsky and Papert (1969). Another famous algorithm is called Widrow-Hoff or LMS algorithm and allows to obtain a solution that minimizes the Least Mean Square error function.

In the third chapter of this thesis, a particular family of algorithms, called ”meta-heuristic algorithms,” will be presented, with focus on: GA, WOA, and PSO.

1.3.4 Multi-layer perceptron networks

As it was said, the perceptron and TLU limits occur when inputs are not linearly separable. Unfortunately, it happens regularly. To address this issue, using more than one decision surface is the solution, since it allows to split the pattern space into the necessary number of parts. This is achieved by multi-layer perceptron networks [7].

The multi-layer model has basic characteristics that regard the activation function and links between nodes. Indeed, units are characterized by a non-linear and derivable activation function and are usually connected with all the neurons in the previous layer, hence the name of totally connected model.

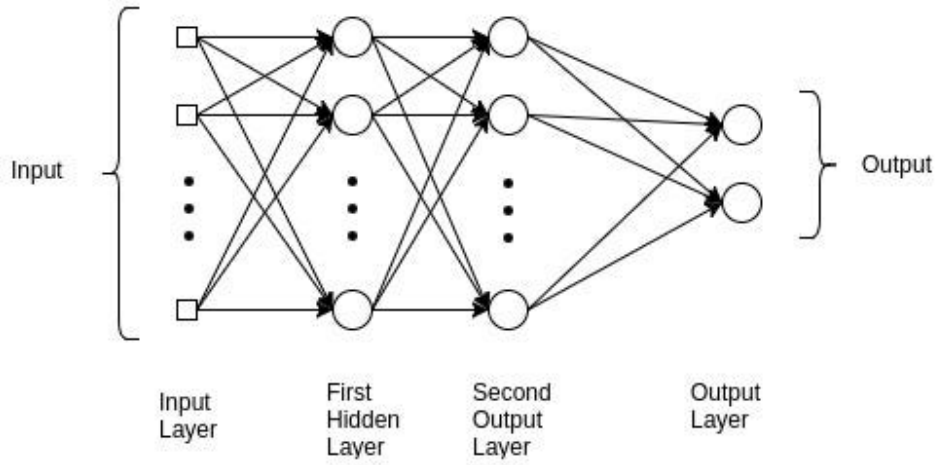

Figure 1.6. Multi-layer perceptron example.

Figure (1.6) shows that the structure of an ANN features one hidden layer, which is an intermediate layer of neurons between input and output. The number of neurons in each layer is independent of the number of nodes in the others, but there are some constraints. The input layer indeed has many nodes as the number of features of the input data and the output layer as the possible outcomes. As regards hidden layers, they impact network performance: an excessive number of nodes may result in overfitting as well as an increased computational expense.

Concluding the introduction of Artificial Neural Networks, Figure 1.7 presents the different types of decision regions that can be formed by single and multi-layer perceptrons with one and two layers of hidden units and two inputs. Shading denotes decision regions for class A, and as it is seen, the three-layer nets perform much better.

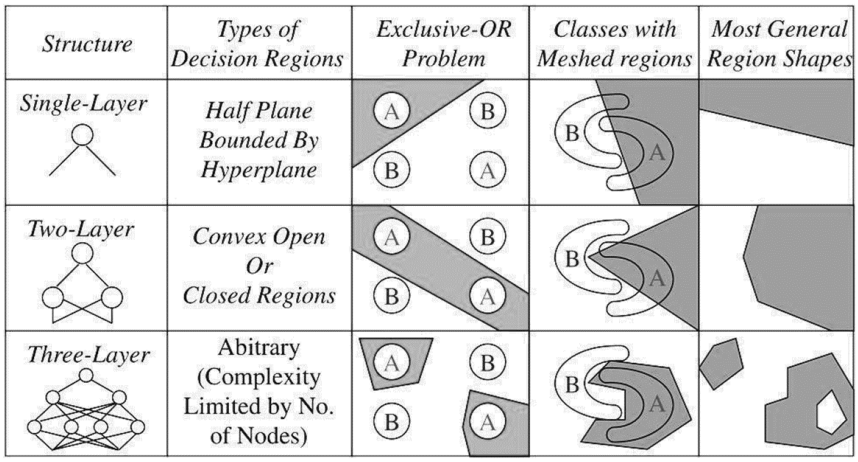

Figure 1.7. Layer effect on network performance [7].

Chapter 2

Generative Adversarial Networks

The second chapter of this thesis deals with Generative Adversarial Networks (GANs). The development of this model dates back a few years, so it is currently one of the main topics of Machine Learning research. In order to understand the fundamental concepts that characterise the model and put them into practice, three themes are presented in this chapter. In the first part, the generative model is introduced, then the architecture of GANs is presented using an example, and finally, some applications of the framework in solving certain tasks.

2.1 Generative Modeling

The context of Machine Learning in which GANs fall is the Unsupervised Learning, in particular the density estimation. In order to understand their role, it is useful to introduce the generative problem. The objective of generative modeling algorithm is to learn a $p_{model}(x)$ that approximates as closely as possible an unknown distribution $p_{data}(x)$, from which the training examples x are drawn from [8]. The most intuitive solution consists in writing a function $p_{model}(x, \theta)$ and finding θ^* value that makes p_{model} similar to p_{data}.

One of the most famous approaches to this task is maximum likelihood estimation, which consists of minimizing the Kullback-Leibler divergence (KLIC) between the two distributions. This method allows for minimizing the loss of information, that corresponds to the KLIC value, and to explicit density functions. However, many limits are imposed by the nature of this solution

to obtain useful results, mainly the need to deal with simple functions only. Since it is necessary to analyze more complex situations, this method does not meet the requirements. Generative Adversarial Networks, on the other hand, enable this obstacle to be overcome.

Now the generative problem is presented more in detail in order to know what GANs are based on. As it has already been hinted, Generative Neural Networks (GNN) have to learn despite the complexity of the probability distribution of data, which will be called $\mathcal{X}$. This role is entrusted to the so-called *generator*

$$g : \mathbb{R}^q \Rightarrow \mathbb{R}^n, \tag{2.1}$$

that maps samples from a tractable distribution $\mathcal{Z}$, supported in $\mathbb{R}^q$, to points in $\mathbb{R}^n$ that can be confused with the given data. In other word, the generator must be able to find points $z \sim \mathcal{Z}$ such that $g(z) \approx x$ where $x \sim \mathcal{X}$. Due to the generator operation, vector z results unknown and indeed it is called *latent variable* or *noise*, and consequently, its domain is named *latent space*. In addition, the generator allows to compute the probability $p_{\mathcal{Z}}(z)$, which is used to calculate the likelihood or evidence of a particular sample z.

2.2 Generator and discriminator duality

At this point, it is clear what the scope of the generator is, but it has not been explained how to achieve the objective. Hence, the training method to obtain the best results has to be discussed, also because it is a major problem. The key to the GNN training, indeed, is finding the right objective function that quantifies the discrepancy between the real and the generated samples, and it is not easy. However, Generative Adversarial Networks are an excellent option, and what they concern will now be featured.

GANs fall into a group of techniques called *implicit generative model*, that includes all the models that avoid the entire issue of designing a tractable density function and focus on generating samples to deal with [9]. More specifically, GANs involves the implementation of two neural network models, the generator $\mathcal{G}$ and the discriminator $\mathcal{D}$, which behave as if they were adversaries. The role of the first, as already explained, is to generate new instances similar to the real ones, while the discriminator has the function of recognising if the data is fake or not. Training can be said to be completed when the $\mathcal{D}$ is no longer able to make distinctions.

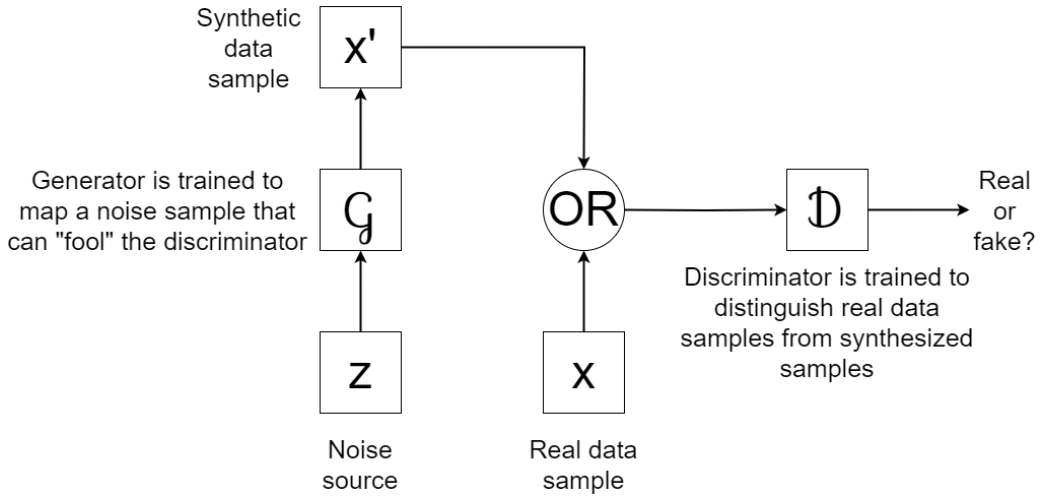

Figure 2.1. GAN common structure.

It is crucial that $\mathcal{G}$ learns without having access to the real data, but through interaction with $\mathcal{D}$. In particular, the error signal produced by the discriminator leads to the generation of better-quality forgeries. Being a sort of competition, the two opponents have different strategies, which means a respective function characterizes both

$$g_\theta(z;\theta) \quad \text{and} \quad d_\phi(x;\phi), \tag{2.2}$$

where θ and ϕ represent the learning parameters that define the behaviour of the two rivals, and the function $d(x)$ represents the probability that x comes from data. The training method consists of an attempt by both the generator and the discriminator to minimise their own cost function, with the result of encouraging the adversary to perform better and better .

An example that allows the concept just illustrated to be figured out is now presented. It is explained how the fitting to a one-dimensional Gaussian distribution takes place.

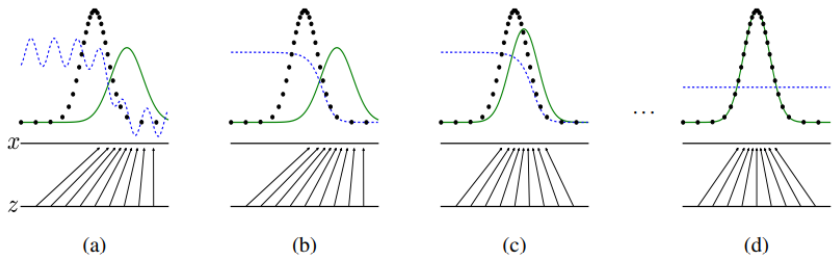

Figure 2.2. Convergence of GAN applied to one-dimension Gaussian distribution [10].

The GANs technique requires the discriminator and generator to be updated simultaneously. In fact, it can be seen from the sequence of graphs that the blue dashed line, which depicts the discriminating distribution, and the green solid line, which shows the trend of the generative distribution, change their shape at each step. As for the black dotted line, it is the representation of the data distribution. Below the curves, the domain of x and z is represented. The arrows indicate how the mapping of $g(z) = x$ varies: g contracts in regions of high density and expands in regions of low density of $\mathcal{X}$. In stage (a) it is consider an adversarial pair near convergence. Indeed, $g(\mathcal{Z})$ is similar to $\mathcal{X}$ and $\mathcal{D}$ is partially accurate. In (b) it is shown that the inner loop of the algorithm $\mathcal{D}$ is trained to discriminate samples from data, converging to $d^*(x) = \frac{x}{x+g(z)}$. In the third step (c), after an update to $\mathcal{G}$, it is evident that samples are more likely to be classified as data. Finally, in (d) the discriminator is not more able to discern generated values from real data, so that $d(x) = \frac{1}{2}$ and $g(\mathcal{Z}) = \mathcal{X}$ [10].

Since it is clear the generator - discriminator duality, it is possible to deepen further the topic. As it has already been said, the discriminator is concerned with providing a probability and can therefore be represented symbolically by a function of the type

$$d_\phi : \mathbb{R}^n \Rightarrow [0, 1]. \tag{2.3}$$

Its classification method is thus definable as binary, since its responses are either ”real” or ”fake”. In fact, once ϕ has been chosen, its operations can only lead to two conclusions: if $d_\phi \approx 1$, then $x \sim \mathcal{X}$ and if $d_\phi \approx 0$, then $x \sim g_\theta(\mathcal{Z})$. Due to this, a function is particularly suited to being the loss of the problem

$$J_{GAN}(\theta, \phi) = \mathbb{E}_{x\sim\mathcal{X}}[log(d_\phi(x))] + \mathbb{E}_{z\sim\mathcal{Z}}[log(1 - d_\phi(g_\theta(z)))]. \tag{2.4}$$

It is called *cross-entropy loss function* and is treated differently from the discriminator and the generator: the first aims to maximise the value of the function, which corresponds to having made few misclassifications in the data, while the second aims to minimise it, trying to deceive the discriminator.

The relationship between the two networks is a recursive situation in game theory, so it is possible to find analogies and apply the constituent theorem to study the generative problem. For example, GANs training corresponds to look for the local Nash equilibrium (θ^*, ϕ^*) such that

$$\phi^* \in \arg\max_\phi J_{GAN}(\theta^*, \phi) \quad \text{and} \quad \theta^* \in \arg\min_\theta J_{GAN}(\theta, \phi^*). \tag{2.5}$$

It is, in fact, the local minimum of each player's cost with respect to that player's parameters. The main obstacle is that with local moves, no player can reduce its cost further, assuming the other player's parameters do not change [8] [11].

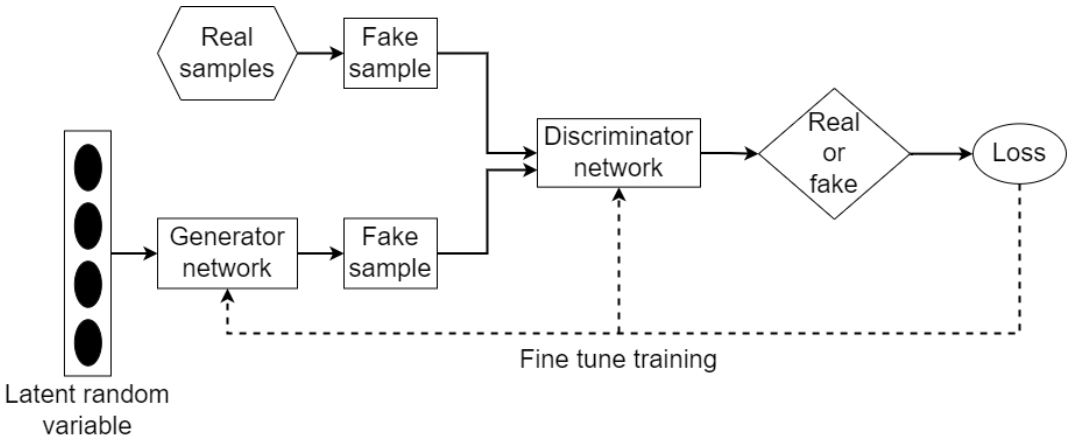

Figure 2.3: The main loop of GAN training.

Two sub-problems are therefore addressed in the GANs, one concerning the generator and one the discriminator, and balancing their resolution is a critical hurdle. Indeed, a stalemate is frequently reached, as in the cases that will be now presented. It is supposed to train the model with an iterative method, which consists of alternatively improving the performance of the discriminator and the generator. Now the model has reached the optimal situation for the generator is assumed, i.e. $g_{\theta^*}(\mathcal{Z}) = \mathcal{X}$ and the discriminator is completely confused, which means $d_{\phi^*} = \frac{1}{2}$. The above is a condition of unstable equilibrium since it is opposite from the discriminator objective. Consider another case in which the GAN is trained with a gradient-descent algorithm and the discriminator is brought to optimality, i.e. to the point of the cost function that $\nabla_\theta J_{GAN}(\theta, \phi^*)$ is close to 0. The generator would than be unable to improve since the cost function has already been driven to a point on non-derivability. On the other hand, not training the discriminator sufficiently would render the improvement of the generator ineffective. Another problem inherent in the nature of GANs is called mode collapse and occurs in the case where the function $g(\mathcal{Z})$ converges to a single point $x^{(1)} \sim \mathcal{X}$, that is

$$g_\theta(z) = x^{(1)} \quad \text{for almost all} \quad z \sim \mathcal{Z}. \tag{2.6}$$

The discriminator would come to the conclusion that $d_{\phi^*}(x^{(1)}) = \frac{1}{2}$ and $d_{\phi^*}(x^{(j)}) = 1$ for all $j > 1$. Finally, the opposite could also occur where the generator is unable to map any point of $\mathcal{Z}$ to $x \sim \mathcal{X}$ [8][9]. Thus, the bilateral influence that generator and discriminator cause on the parameterised function makes it impossible to predict the effectiveness of the GANs a priori and often leads to settle for a theoretically imperfect result.

What has just been explained makes the modus operandi of Generative Adversarial networks clear, and it is therefore possible to understand the potential of this model and its versatility. In fact, there are numerous facets that allow it to be adopted for solving various tasks, some of which will now be explored in more detail. Furthermore, the last chapter of this thesis will deal with a case of particular interest, which sees GANs applied in a completely new way.

2.3 GAN applications

As Generative Adversarial Networks are a new and high-performance Machine Learning technique, a great deal of research is being carried out in this area. Some of this research has focused on solving various tasks through the use of this model. Therefore, some of the areas in which these networks have proven to be effective are now presented [12].

2.3.1 Classification

This task consists of identifying classes and being able to tag each piece of data with the correct class. A Deep Convolutional Generative Adversarial Network (DCGAN) is used to perform this task. Indeed, by reading the output of the convolutional layers of the discriminator, the feature map can be appreciated. In order to understand how features can be extrapolated, it is necessary to understand what characterises Convolutional Neural Networks (CNNs).

2.3.1.1 Convolutional Neural Networks

Convolutional Neural Networks [13] are feed-forward networks, i.e., in which the flow of information is unidirectional, from input to output. Their main characteristic is the organisation of the layers, which can be either *convolutional*

or *pooling*, in modules. This type of architecture makes CNNs capable of performing well in the field of image classification. To understand what makes them so efficient, it is necessary to describe the different types of layers. Convolutional layers are used for feature extraction, which is enabled by their feature map. Neurons are in fact organised in groups, and each in turn has a receptive field that is connected to the previous layer. The weights that characterise these connections are often referred to as filter banks and are the same for all neurons within a feature map. Mathematically, it can be expressed as

$$\Upsilon_k = f(W_k \cdot x), \tag{2.7}$$

where the input is denoted by x, the convolutional filter related to the kth feature map is denoted by W_k, and $f(\cdot)$ represents the nonlinear activation function. Pooling layers are used to reduce the spatial resolution of feature maps. There are two methods to perform this operation: the first involves the use of *average pooling aggregation layers* to propagate the signal to the subsequent layers, while the second, developed more recently, consists of propagating the maximum value of the receptive field and is in fact referred to as *max pooling aggregation layers*.

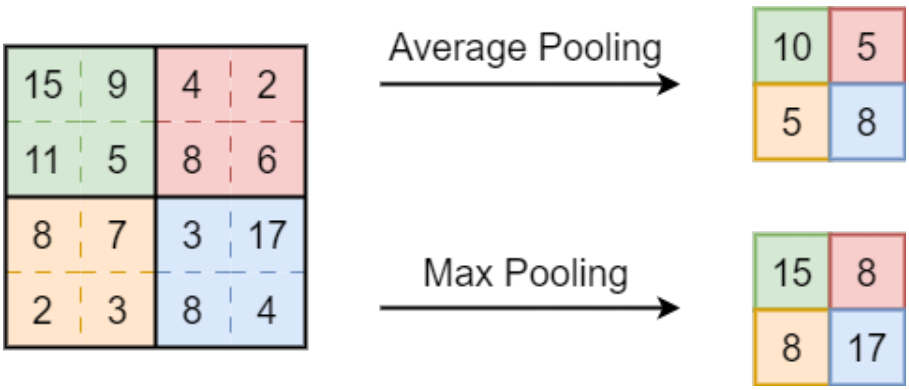

Figure 2.4. The two types of pooling: average versus max.

To conclude this description, the fully connected layers allow the interpretation of feature representation and perform the high-level reasoning function. To improve feature extraction, it is common to stake several convolutional and pooling layers.

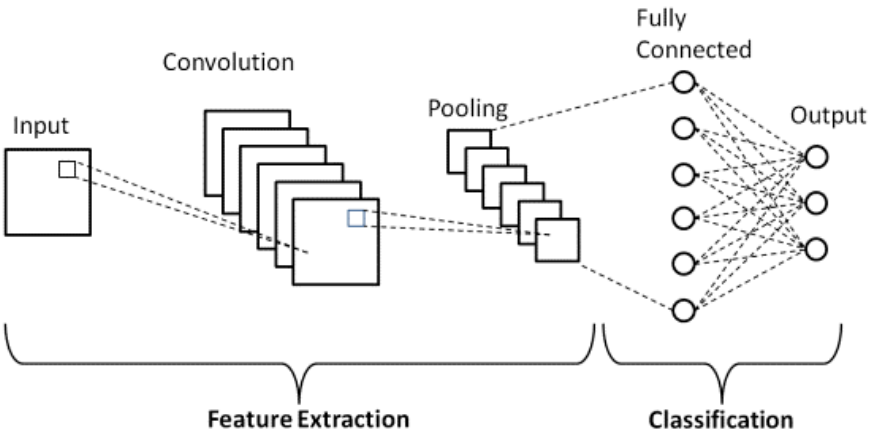

Figure 2.5. CNN layers organization [13].

2.3.1.2 CNNs implemented in GANs

Now that the structure of convolutional neural networks is clear, the method for implementing them in the GAN model is presented. First, both the generative and discriminative networks are of the convolutional type, but with some modifications of the classical structure presented above.

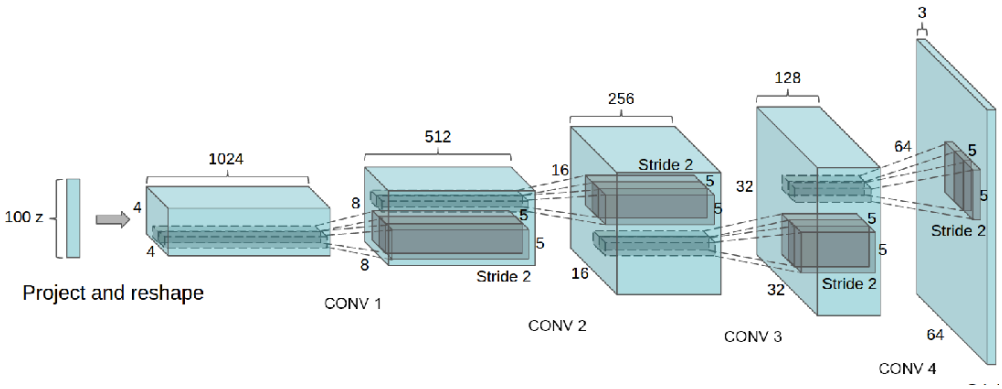

Figure 2.6. An example of DCGANs: a 100 dimensional uniform distribution $\mathcal{Z}$ is projected to a small spatial extent as a convolutional representation with many feature maps. A series of four fractionally-strided convolutions then converts this high-level representation into a 64 × 64 pixel image. Notably, no fully connected or pooling layers are used [14].

As far as the determinstc spatial pooling function is concerned, it is replaced by strided convolutions, allowing both the generator and discriminator to learn their own spatial downsampling. Continuing on, the fully connected layers on top of convolutional features are eliminated. Another modification

introduced is called batch normalization, the effect of which is to stabilise learning by using a zero mean and unit variance input. In particular, it has the effect of aiding gradient flow in deeper models, thus preventing collapse. Applying it to all layers results in sample oscillation and model instability, while leaving the generator output and discriminator input layers untouched avoids any repercussions. Finally, the ReLU activation function applied to the generator, with the exception of the output layer, which uses the Tanh function, allowed the model to learn more quickly [14].

2.3.2 Image synthesis

Another area to which GANs are widely applied is image synthesis. One of the most appropriate architectures for this task is called LAPGAN. In order to explain how image synthesis takes place, it is necessary to digress into the Laplacian pyramid.

2.3.2.1 Laplacian pyramid

To understand what the Laplacian pyramid [15] consists of, it is necessary to introduce some elements. Suppose to have a $j \times j$ image and apply downsampling $d(\cdot)$ to it. The result is an image of size $\frac{j}{2} \times \frac{j}{2}$. Conversely, let $u(\cdot)$ be an upsampling, which results in a smoother image expanded to twice the original size. Before constructing the Laplacian pyramid it is first necessary to construct a Gaussian pyramid using downsampling $\mathcal{G}(I) = [I_0, I_1, \cdots, I_K]$ such that $I_0 = I$ and I_k is k repeated application of $d(\cdot)$ to I, i.e. $I_2 = d(d(I))$. K is the number of levels in the pyramid. To determinate the parameter h_k of the Laplacian pyramid $\mathcal{L}(I)$ it is used the formula

$$h_k = \mathcal{L}_k(I) = \mathcal{G}_k(I) - u(\mathcal{G}_{k+1}(I)) = I_k - u(I_{k+1}). \tag{2.8}$$

Using backward recurrence, the image is reconstructed to the full resolution

$$I_k = u(I_{k+1}) + h_k. \tag{2.9}$$

2.3.2.2 LAPGAN

To realise this architecture, a set of convolutional GANs $\mathcal{G}_0, \mathcal{G}_1, \cdots, \mathcal{G}_K$ is used, each of which captures the distribution of coefficient h_k for natural images at a different level of the Laplacian pyramid. By applying the Formula

2.9, the image is sampled, with the difference that parameters h are generated by the generative models

$$\tilde{I}_k = u(\tilde{I}_{k+1}) + \tilde{h}_k = u(\tilde{I}_{k+1}) + \mathcal{G}_k(z_k, u(\tilde{I}_{k+1})). \tag{2.10}$$

To perform the generation, the condition that $I_{K+1} = 0$ is set, so using the final model level G_K the residual image is generated. As the Figure 2.7 shows, $\mathcal{G}_0, \mathcal{G}_1, \cdots, \mathcal{G}_{K-1}$ are so-called conditional GANs, i.e. they takes as input two elements: one is an upsampled version of the current image $\tilde{I}_{k+1}$ as a conditioning variable, while the other is the noise vector z_k.

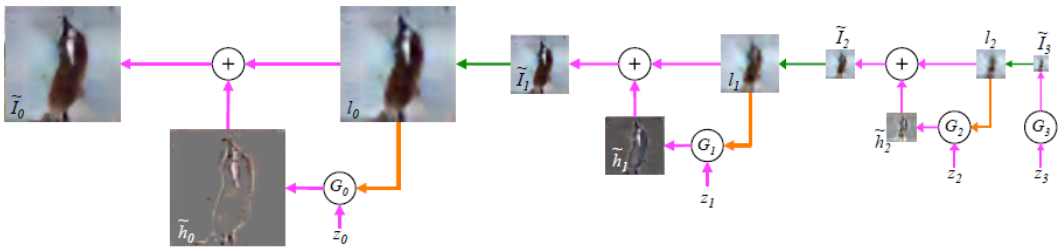

Figure 2.7. It is showed the procedure for a pyramid with $K = 3$, using 4 generative models. It is started with a noise sample z_3 and a generative model $\mathcal{G}_3$ to depict $\tilde{I}_3$. This is upsampled and then used as the conditioning variable at the next level. This process repeats to yield a final full resolution sample I_0 [15].

With these notions in mind, it is possible to go further and introduce training. To explain how it is done, we refer to Figure 2.8. It is started with an image taken from the training set $I_0 = I$, from which is obtained a low-pass version l_0. At this point, with the same probability l_0, is used to produce either a real image or a generated one. The first option involves the computation of a high-pass $h_0 = I_0 - l_0$, while the second involves the generation of an image $\tilde{h}_0$ through the use of l_0 and a noise vector z_0. The last step consists of passing the image l_0 and either h_0 or $\tilde{h}_0$ to the discriminator. The same procedure is repeated in scale 1 and scale 2, independently from each other, using I_1 and I_2. Note that at the last level of the scale, the generator operates as a standard GAN, that means $\tilde{h}_K = g_K(z_K)$, and the discriminator D_K takes as input only h_K or $\tilde{h}_K$.

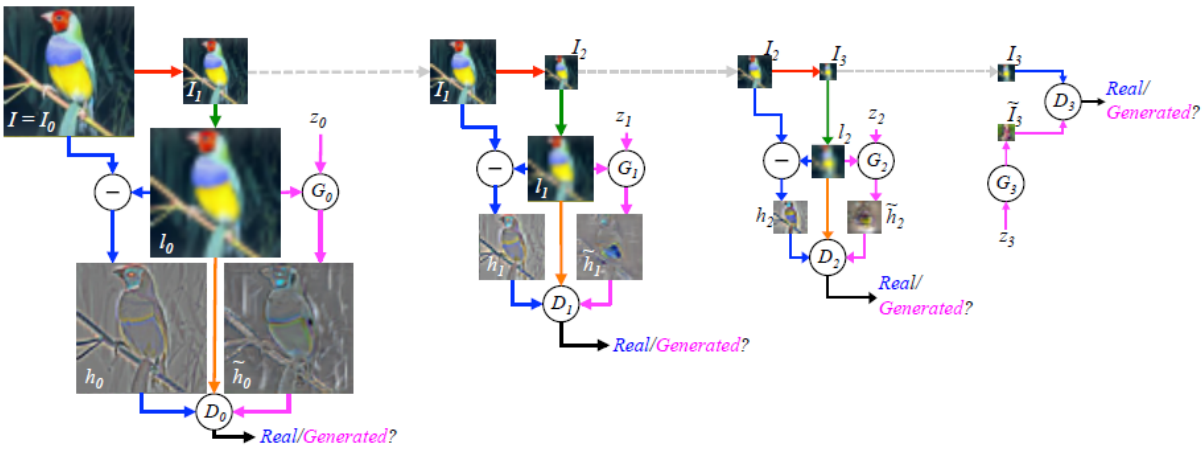

Figure 2.8. The training procedure of the LAPGAN model [15].

The feature of conditioning can be extended to incorporate natural language to synthesize images from text descriptions. It is called *reverse captioning* and is operated by a specific type of GAN, which are the What-Where GAN (GAWWN). Thanks to an interactive interface, complicated images can be built up with textual descriptions of parts and user-supplied bounding boxes.

2.3.3 Image-to-image translation

The last field of application presented in this thesis is called *image-to-image translation* and consists of training GAN to translate an image from a source domain X to a target domain Y in the absence of paired images [16]. Some examples of translation are grayscale to colour images, image to semantic labels, and edgemap to photograph. This task can also be solved using supervised learning settings, but obtaining paired data is difficult, expensive, and often requires artistic authoring. On the other hand, the model capable of translating from one domain to another, called *cycleGAN*, does not require paired input-output images.

Since its training method presents a clear difference compared to the classic GANs procedure (Section 2.2), it will now be presented.

The model contains two mapping functions: $G : X \rightarrow Y$ and $F : Y \rightarrow X$ and the associated discriminators D_X and D_Y. D_Y aims to distinguish the images produced by G from those in the Y domain, vice versa for D_X and F (Figure 2.9 (a)).

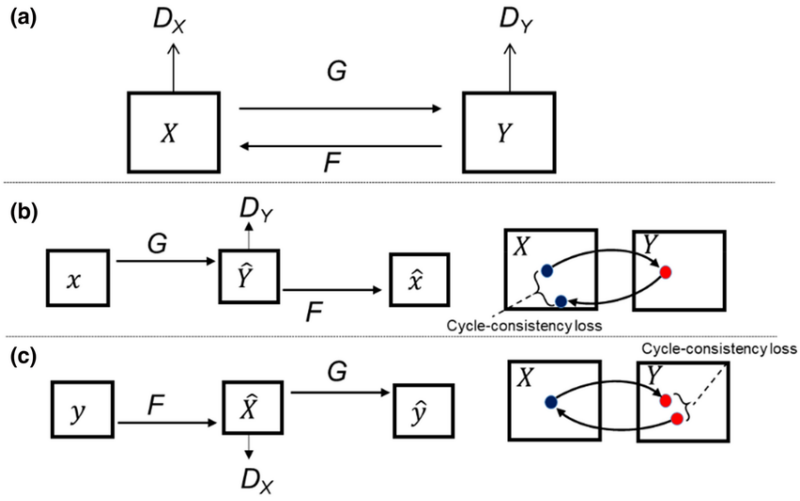

Figure 2.9. Representation of the model and explanation of the cycle consistency properties[16].

Now taking the Formula 2.4 and adapting it for the function G and its discriminator D_Y gives

(2.11)

$$J_{GAN}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p_{data(y)}}[log(D_Y(y))] + \mathbb{E}_{x \sim p_{data(x)}}[log(1 - D_Y(G(x)))],$$

where $x_i \in X$, $y_i \in Y$, and p_{data} denotes the respective distributions. A loss function is likewise defined for the function F and the discriminator D_X. Following the operating principle of GANs, G aims to minimize the function against the adversary D, which seeks to maximize it. Training a model with this function, however, does not allow to complete the task since a biunivocal correspondence between an input x_i and a desired output y_i is not guaranteed. Indeed, the model needs another feature to allow the reconstruction of the images, as shown in Figure 2.10. What leads to the desired results is the *cycle consistency loss function*

(2.12) $$J_{cyc}(G, F) = \mathbb{E}_{y \sim p_{data(y)}}[\|F(G(x)) - x\|] + \mathbb{E}_{x \sim p_{data(x)}}[\|G(F(y)) - y\|].$$

It gives two fundamental properties: the first, called *forward cycle consistency*, is defined as $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$ (Figure 2.9 (b)). Similarly, *backward cycle consistency* is the property that allows $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$ (Figure 2.9 (c)). Therefore, to obtain effective training, the function to be used turns out to be

(2.13)

$$J(F, G, D_X, D_Y) = J_{GAN}(G, D_Y, X, Y) + J_{GAN}(F, D_X, X, Y) + \lambda J_{cyc}(G, F),$$

where λ controls the relative impact of the two targets.

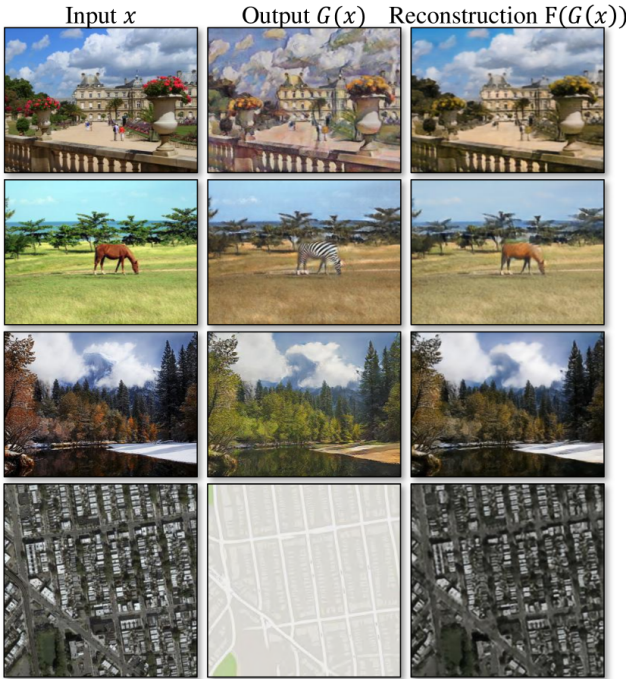

Figure 2.10. The input images x, output images $G(x)$ and the reconstructed images $F(G(x))$. From top to bottom: photo $\leftrightarrow$ Cezanne, horses $\leftrightarrow$ zebras, winter $\leftrightarrow$ summer Yosemite, aerial photos $\leftrightarrow$ Google maps [16].

In conclusion, in order to highlight its great efficiency of this method, Figure 2.11 shows the results of applying it to different cases.

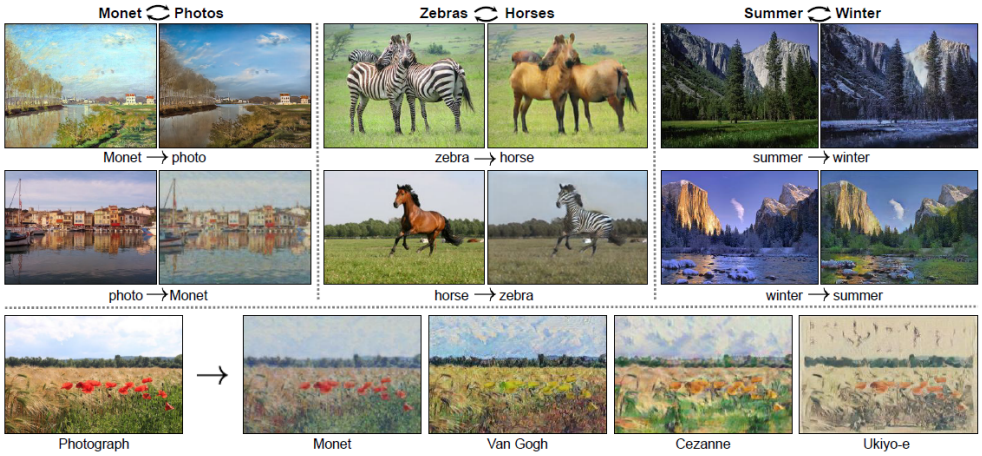

Figure 2.11. The model applied to several translation problems: *(left)* Monet paintings and landscape photos from Flickr; *(center)* zebras and horses from ImageNet; *(right)* summer and winter Yosemite photos from Flickr; *(bottom)*: using a collection of paintings of famous artists, the method learns to render natural photographs into the respective styles [16].

Chapter 3

Nature-inspired Metaheuristic Algorithms

Neural Networks are able to solve certain tasks thanks to the objective function, which describes the rules to be followed and possible penalties. The method by which the network learns them is described by the algorithm used. This chapter presents three algorithms derived from the study of recursive patterns in the natural environment: Genetic Algorithm (GA), Particle Swarm Optimization (PSO), and Whale Optimization Algorithm (WOA). In order to understand how they are implemented, an overview regarding optimization is given. Finally, the code of a practical application is linked.

3.1 Optimization Algorithms

Heuristics refers to the strategy of solving a problem by trial-and-error to produce acceptable solutions to a complex problem, such as in a child's learning experience. The aim is to find good and feasible results in an acceptable timescale. Metaheuristics, on the other hand, includes high-level procedures, which, as the prefix *meta-* suggests, allows for more accurate results. They are particularly useful in cases where exact methods are not convenient, for example because of too long calculation times. The algorithms discussed in this chapter are of the second type and are drawn from careful observation of biological macro- and microsystems. In addition, all metaheuristics algorithms use certain tradeoff of randomization and local search. The first to be developed is GA, proposed in the 1960s by John Holland and his collaborators at the University of Michigan. In particular, they

were the first to introduce random recombination techniques for solving systems. Since then, GA has proved to be an excellent strategy for solving optimization problems, in fact, numerous books and articles have been written about it. One of these, written by L.J. Fogel, together with A.J. Owen and M.J. Walsh, published in 1966, dealt with the development of evolutionary programming techniques, based on representing solutions as finite-state machines subjected to random mutations. The development of this article led to the birth of a much broader discipline called *evolutionary algorithms*, or *evolutionary computation*. During the 1980s and 1990s, increased research on this subject led to the proliferation of metaheuristic algorithms, including Simulated Annealing (SA), an optimization technique inspired by the annealing process of metals, Ant Colony Optimization (ACO), which translates the pheromone-based communication of social ants, and Particle Swarm Optimization (PSO). In 1997, the publication of ”No free-lunch theorems for optimization” introduced a major breakthrough: if algorithm A outperforms algorithm B in solving a problem, it is not necessarily better in every other application. So the search was no longer for the absolute best method, but focused on solving each problem separately. Over time, more and more metaheuristic algorithms have been developed to try to meet all needs, such as Whale Optimization Algorithm (WOA) [17][18].

Since the types of optimization problems can be very different, the algorithms in turn also have characteristics that differentiate them.

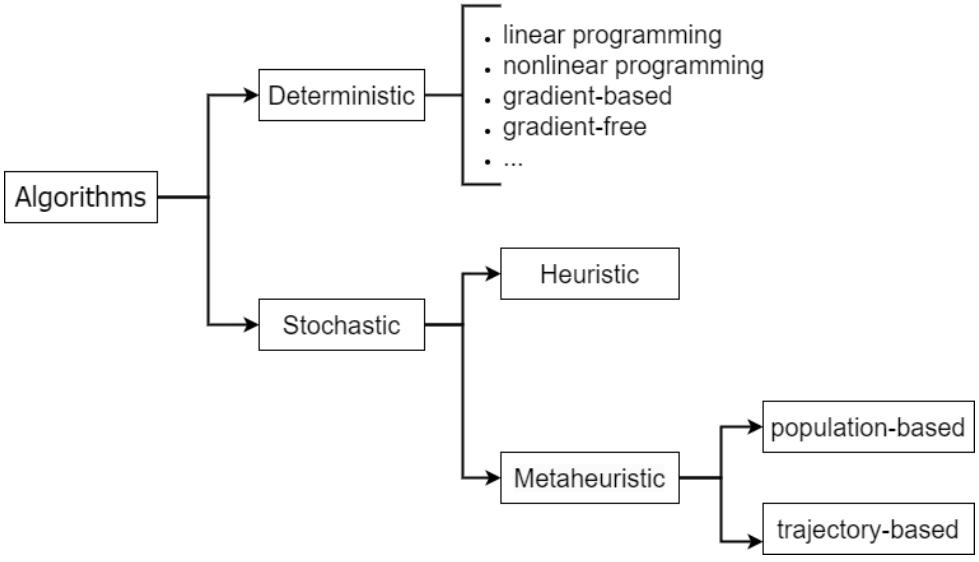

Figure 3.1. Classification of algorithms.

Algorithms fall into two broad categories determined by the approach to the problem. Deterministic algorithms, such as hill-climbing techniques, define a precise and retractable path as it is defined by a strict procedure. Most are gradient-based, i.e. they use the value of the derivative to determine the path to follow. This characteristic makes them useless in cases where the objective function is discontinuous. As is often the case in neural networks, stochastic algorithms are the best alternative. Indeed, they use a different criterion, based on information sharing, called *swarm intelligence*, and a random component that allows larger spaces to be explored rapidly. The latter are divided into *heuristic*, which, as can be guessed from the definition of heuristics, allow a good result to be found, but are hardly the best, and *metaheuristic*, which by contrast, perform much better. Of this type exists two categories: trajectory-based and population-based [17].

3.2 Natural Optimization Methods

Over millions of years, all biological environments have undergone major changes, recognized by the term evolution, which have enabled adaptation to the framework through the improvement of certain pre-existing features. Through abstract reasoning, certain features of nature can be visualized in a modern context, thus exploiting the problem-solving capacity brought about by evolution. An example of this are metaheuristic algorithms, some of which are precisely bio-inspired. Two characteristics distinguish this class: diversification and intensification. The first consists of the generation of different solutions to explore the available space, while intensification means focusing the search in a region where a good solution has been found. Through the subsequent selection of the best result, these algorithms are able to find global optimality, while diversification via randomization avoids the trap of local optima. Three will now be presented, along with the Python code submission: Genetic Algorithm, Particle Swarm Optimization, and Whale Optimization Algorithm.

3.2.1 Genetic Algorithm

Genetic Algorithm is a population-based algorithm founded on the Darwinian theory of natural selection and the principles of genetics [19]. It is one of the most widely used algorithms due to its ability to perform even in very complex situations. In fact, it is suitable for optimizing very different objective

functions: continuous and discontinuous, stationary and non-stationary, linear and non-linear, or even with random noise. Versions of GA generally fall into two categories, depending on whether the variables are continuous or binary. The difference lies only in certain encoding and decoding steps of the data, and therefore only the continuous version will be dealt with, as well as the best-performing one.

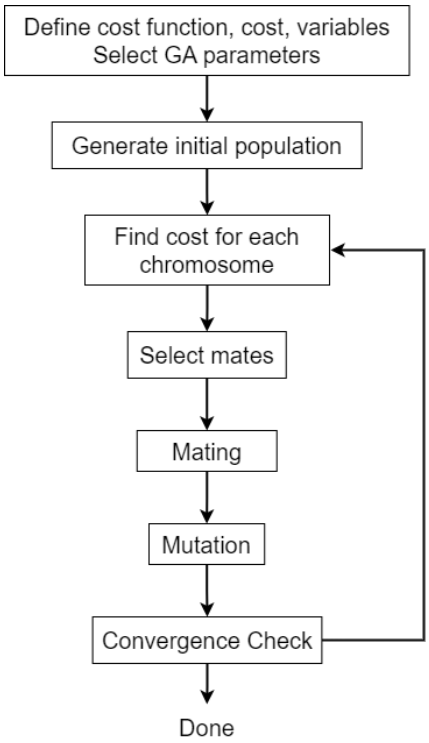

Figure 3.2. Flowchart of a continuous GA.

Solving a problem using GA consists of finding an optimal solution in terms of the problem variables. After defining the objective function and the GA parameters, the role of which will be discussed below, the population can be initialized. Each individual is defined as *chromosome* and consists of an array of variables of size $1 \times N_{var}$, where N_{var} is the number of parameters of the problem, so that

$$chromosome = [p_1, p_2, \cdots, p_{N_{var}}]. \tag{3.1}$$

Each chromosome is associated with a value corresponding to that of the cost function according to the variables of the respective. The next step is to select only those chromosomes most deserving of producing offspring in the next generation, that is, to apply the theory of natural selection. The decision criterion is based on the value previously associated to each element

in the population, determining the most promising. This process allows the population to continually evolve into better and better individuals, bringing it ever closer to the desired outcome. One of the GA parameters, called *selection rate*, defines the fraction of N_{pop} that survives to the next step. This parameter influences the next generations as a very tight selection lead to a low variance in the offspring, while a too large residual population slows down the optimization process.

```
def natural_selection(self, population, costs):
    # transform the x_rate into a numerical value indicating how far
        up the population vector index it has to go
    n = int(self.x_rate * self.population_size)
    costs = costs[:len(population)]
    indices = np.argsort(costs)
    # sort the population according to the value of the cost
        function to identify which individuals are the best performers
    sorted_population = population[indices]
    selected_population = sorted_population[:n]
    return selected_population
```

In order to proceed, two parent chromosomes, referred to as $parent_1$ and $parent_2$, are selected to create the new generation, similar to what happens in the crossing over. There are many methods for choosing individuals for mating, and one of them is called *roulette wheel weighting*. It is about a random extraction method in which the probability of choosing a chromosome depends on its fitness. In fact, it is inversely proportional to the value of the cost function associated with each individual, so the better performers have a higher probability of being chosen.

```
def roulette_wheel_weighting(self, population, costs):
    probability = []
    costs = np.sort(costs)
    ordinated_costs = costs[:len(population)]
    cost_n = costs[-1]
    # normalize the values of the cost function so that the sum is
        one
    for i in range(len(ordinated_costs)):
        probability.append((ordinated_costs[i] - cost_n) /
            (sum(ordinated_costs) - (cost_n * len(ordinated_costs))))
    rand = np.random.uniform(probability[-1], 1)
```

```
    # find the chromosome that corresponds to the interval in which
        rand falls
    for q in range(len(probability)):
        if rand > probability[q]:
            chosen_chromosome = population[q]
            return chosen_chromosome
```

The parents are then mated to form the new generations. The laws describing the process are

$$offspinrg_1 = parent_1 - [\beta_1(p_{m1} - p_{d1}), \beta_2(p_{m2} - p_{d2}), \beta_3(p_{m3} - p_{d3}), \cdots, \beta_n(p_{mn} - p_{dn})],$$

$$offspinrg_2 = parent_2 - [\beta_1(p_{m1} - p_{d1}), \beta_2(p_{m2} - p_{d2}), \beta_3(p_{m3} - p_{d3}), \cdots, \beta_n(p_{mn} - p_{dn})],$$

where β_n represents a parameter in the range $[0, 1]$, p_m and p_d instead are the variables contained in the genome of the parent. By using a random number, it is possible to introduce a casual component into the algorithm that allows exploration of the domain without logical constraints. The process just described is called *blending crossover* and is one of the most widely used techniques to mix two chromosomes.

```
def mating(self, population, costs):
    for i in range(self.number_of_crossover):
        father = self.roulette_wheel_weighting(population, costs)
        mother = self.roulette_wheel_weighting(population, costs)
        beta = np.random.uniform(low = 0, high = 10, size =
            (self.num_parameters))
        #crossover: blending method
        offspring_1 = father - np.multiply(beta, mother - father)
        offspring_2 = mother + np.multiply(beta, mother - father)
        population = np.vstack((population, offspring_1))
        population = np.vstack((population, offspring_2))
    return population
```

The last part of the single iterative cycle of GA is called *mutation*, and as the name suggests, consists of the random mutation of a few traits on a chromosome, increasing the genetic heritage of the population. It depends on a parameter named *mutation rate*, which represents the fraction of genes involved compared to the total genes in the population.

```
def mutation(self, population, best_index):
```

```
mutation_number = int(self.mutation_rate * self.num_parameters *
    len(population))
for t in range(mutation_number):
    chromosome_choice = np.random.randint(0, len(population)-1)
    if t == chromosome_choice and best_index != t:
        for p in range(len(population)):
            gene_choice = np.random.randint(0,
                self.num_parameters-1)
            if p == gene_choice:
                population[t][gene_choice] = np.random.uniform(low
                    = -self.max_x, high = self.max_x)
return population
```

One of the fundamental principles that characterize many metaheuristic algorithms is *elitism*, which consists of the invariability of individuals to which corresponds an excellent degree of optimization. In fact, as can be seen in the lines of code, one of the parameters for choosing a chromosome is the difference from the most efficient.

To conclude the description of GA, it is necessary to discuss how the algorithm ends the optimization, i.e., stopping criteria. As long as the algorithm proceeds, the chromosomes tend to become more and more similar until the entire population has the same genetic heritage. However, this type of conclusion is not computationally advantageous and also restricts the result to a single optimal solution. It is therefore very useful to introduce stopping criteria in order to improve the performance of the algorithm. Two measures are imposing a maximum number of iterations and keeping track of a parameter, such as population mean cost or minimum cost, to know whether the evolution is proceeding in a profitable manner.

3.2.2 Particle Swarm Optimization

Particle Swarm is a trajectory-based algorithm inspired by the collective motion of flocks of birds and schools of fish [20][21]. In fact, group dynamics is the result of an unconscious optimization process of their situation, which in the case of birds and fish concerns, for example, water temperature or food supply. In particular, the measure that animal species seek to optimize is reciprocal distance, as it allows the transmission of information between individuals. From the abstraction of these concepts comes PSO, whose features will now be presented.

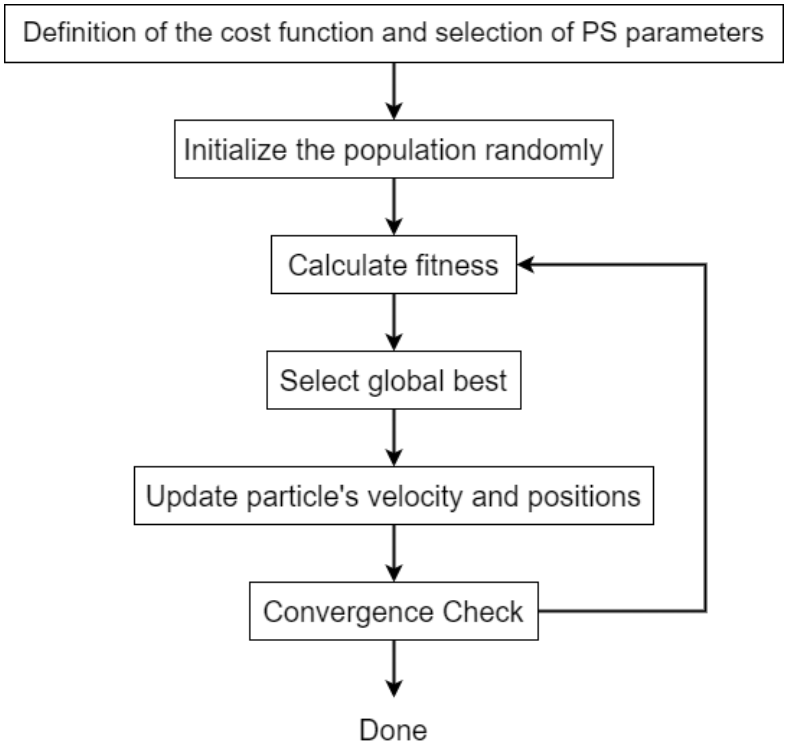

Figure 3.3. Flowchart of PSO.

The population is defined by particles p, characterized by their own velocity and position. The velocity quantifies the shift operated by the particle at each iteration, while the position indicates the n coordinates of the particle on the $n - dimensional$ cost function. Just as the displacement of the mentioned animals is varied, the population also does not have a uniform motion, but it is conditioned by a cognitive, social, and inertia component. To understand their influence, the algorithm flowchart is followed (Figure 3.3).

After defining the problem and initializing the population, fitness is calculated and associated with each particle. A comparison then takes place to determine which of these is the most efficient, that is, which is in the best position. The next step is to update each individual's position through the formula

$$p_{m,n}^{new} = p_{m,n}^{old} + v_{m,n}^{new}, \tag{3.2}$$

where $p_{m,n}$ identifies the nth position coordinate of the mth particle, which velocity in each dimension is described by the vector $v_{m,n}$. As for velocity, the three components are related according to

$$v_{m,n}^{new} = w\ v_{m,n}^{old} + \Gamma_1\ r_1(p_{m,n}^{pbest} - p_{m,n}) + \Gamma_2\ r_2(p_{m,n}^{gbest} - p_{m,n}). \tag{3.3}$$

The first component is inertia, in particular the parameter w whose values usually range from 0,9 and 0,4 as the iterations are completed.

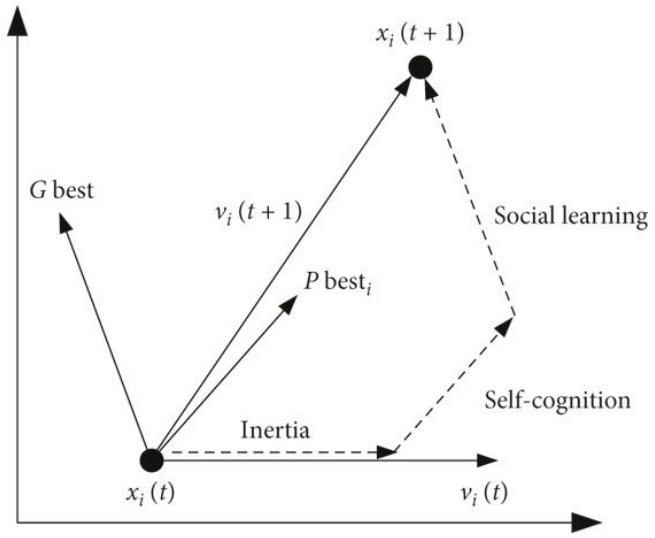

Figure 3.4: The concept of a particle's iteration [22].

In this way, an almost free exploration of space is possible at first until the study is concentrated in the narrow space towards which the particles tend to converge. In the second one, $p_{m,n}^{pbest}$ is the best position of the running iteration. This factor is the implementation of cognitive behavior, that is the tendency to follow the current best. The last is instead the social component, which allows the narrowing down to the best solution $p_{m,n}^{gbest}$ among those found from the beginning of the process. Γ_1 and Γ_2, usually equal to 2, are called *acceleration constants* and represents the learning factors. To implement the velocity update in Python, the code turns out to be as follows.

```
def update_velocity(self, position, velocity, best_position,
    global_best_position):
    inertia = self.w * velocity
    cognitive_component = self.c1 * 2 * np.random.rand(1,
        len(position)) * (best_position - position)
    social_component = self.c2 * 2 * np.random.rand(1,
        len(position)) * (global_best_position - position)
    new_velocity = inertia + cognitive_component + social_component
    return new_velocity
```

Through the analysis of PSO evolution, several tricks can facilitate the convergence of the system to the point of maximum optimization. One ex-

ample is to initialize the population uniformly in the search space, allowing it to converge faster. As regards velocities, it is difficult to establish in advance which values are better since there is a risk of the so-called *velocity explosion*. The latter consists of the violation of the search space by particles initially located along the edge of the search space. It is therefore a good alternative to set the initial velocities at relatively low speeds in order to reduce the probability of running into this problem. Another expedient is to impose boundaries on the velocity so as not to incur the issue described. Consequently, however, the choice of this range is not simple, as each problem requires its own configuration of parameters. A narrow range indeed limits the size of the step, which in some cases can lead to a sinking in the local minima. Finally, unlike GA, PSO needs a convergence criterion as it may exhibit zigzag or oscillatory behavior in the proximity of the global optima. As regards the stop criteria, they can be of different types but are generally based on the maximum number of iterations or a target value of the fitness.

3.2.3 Whale Optimization Algorithm

Whale Optimization Algorithm is the last algorithm presented in this thesis, and like PSO is of the trajectory-based type. It takes its inspiration from the hunting method of humpback whales, whose strategy consists of emitting air bubbles to encircle prey and tightening the radius of the trap produced while swimming around the prey within a shrinking circle and along a spiral-shaped path simultaneously [23].

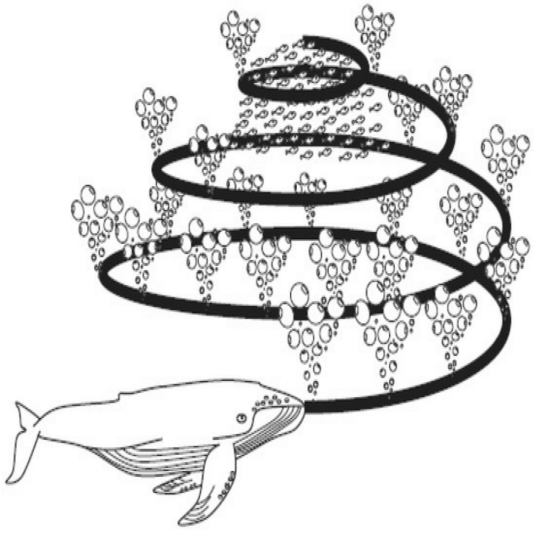

Figure 3.5. Bubble-net feeding behavior of humpback whales [23].

In keeping with the context in which it is inspired, each member of the population of WOA is referred to as whale. Furthermore, as with PSO, each one is associated with values corresponding to the coordinates at the respective point. In the case of real whales, the target is known and lies in an area where they can detect as much prey as possible, whereas in optimization problems, this type of information is not available. This algorithm, therefore, after initializing the problem, uses the position of the element with the lowest values of the objective function as the target. Through the sharing of this information, the optimization process takes place. The entire cycle is summarized in the flowchart in Figure 3.6.

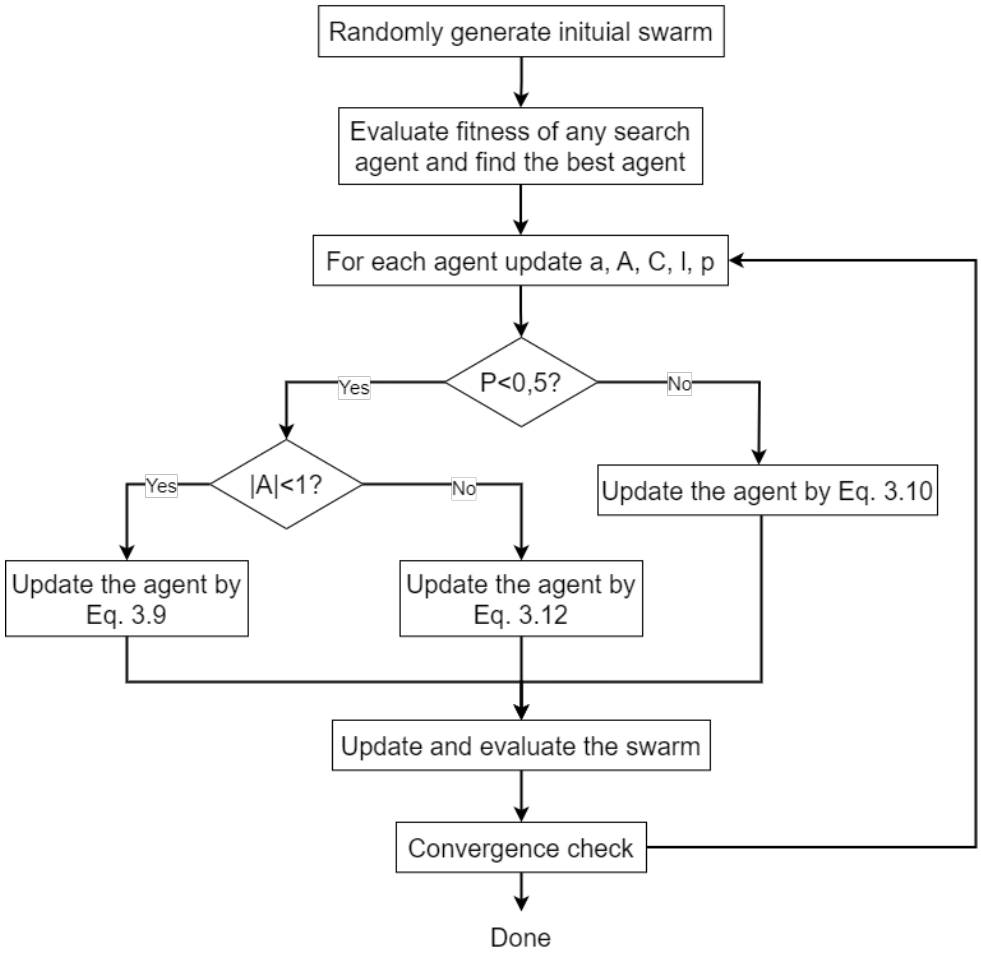

Figure 3.6. Flowchart of WOA.

The next step involves updating the whale positions, the displacement of which is determined by the choice of a random parameter $p \in [0, 1]$ ex-

tracted at each iteration. This allows the rotation with the same probability of the mechanisms called *shrinking encircling* and *spiral updating position*. Mathematically, the two methods are described by two different functions, in which several common parameters appear.

Since displacement is also defined within multidimensional spaces, all acting factors are vectors. Therefore, the parameters are defined as

$$\overrightarrow{A} = 2\overrightarrow{a} \cdot \overrightarrow{r} - \overrightarrow{a}, \tag{3.4}$$

$$\overrightarrow{C} = 2 \cdot \overrightarrow{r}, \tag{3.5}$$

$$\overrightarrow{D} = |\overrightarrow{C}\overrightarrow{X}^*(t) - \overrightarrow{X}(t)|, \tag{3.6}$$

$$\overrightarrow{D}' = |\overrightarrow{X}^*(t) - \overrightarrow{X}(t)|, \tag{3.7}$$

$$\overrightarrow{D}_{rand} = |\overrightarrow{C} \cdot \overrightarrow{X}rand - \overrightarrow{X}(t)|, \tag{3.8}$$

where $\overrightarrow{a}$ is linearly decreased from 2 to 0 over the iterations and $\overrightarrow{r}$ is a random vector in $[0, 1]$. As far as $\overrightarrow{X}(t)$, it is the position vector at the t-th iteration, while $\overrightarrow{X}^*(t)$ is the best position at the moment. Thus, shrinking encircling mechanism is achieved by the reduction of the parameter $\overrightarrow{a}$ which consequently also affect $\overrightarrow{A}$. The new position, hence is located in the interval contained between the initial position (X, Y) and (X^*, Y^*), that is

$$\overrightarrow{X}(t+1) = \overrightarrow{X}^*(t) - \overrightarrow{A} \cdot \overrightarrow{D}. \tag{3.9}$$

Regarding the spiral updating method, it allows the position to be changed according to a logarithmic law that converges towards the best one (X^*, Y^*) and mimics the helix-shaped movement of humpback whales as follows

$$\overrightarrow{X}(t+1) = \overrightarrow{D}' \cdot e^{bl} \cdot cos(2\pi l) + \overrightarrow{X}^*(t), \tag{3.10}$$

where b is a constant for defining the shape of the logarithmic spiral and l is a random number between $[-1, 1]$.

The combination of the two mechanism constitutes the *bubble-net attacking method*, described by the system

$$\overrightarrow{X}(t+1) = \begin{cases} \overrightarrow{X}^*(t) - \overrightarrow{A} \cdot \overrightarrow{D} & if \;\; p < 0,5 \\ \overrightarrow{D}' \cdot e^{bl} \cdot cos(2\pi l) + \overrightarrow{X}^*(t) & if \;\; p \geq 0,5 \end{cases}. \tag{3.11}$$

To increase the search component in the entire space at the initial optimization phases, the used formula is

$$\overrightarrow{X}(t+1) = \overrightarrow{X}_{rand} - \overrightarrow{A} \cdot \overrightarrow{D}_{rand}. \tag{3.12}$$

In particular, it is applied when $|A| \geq 1$, i.e. the best solution has not yet been determined with enough confidence.

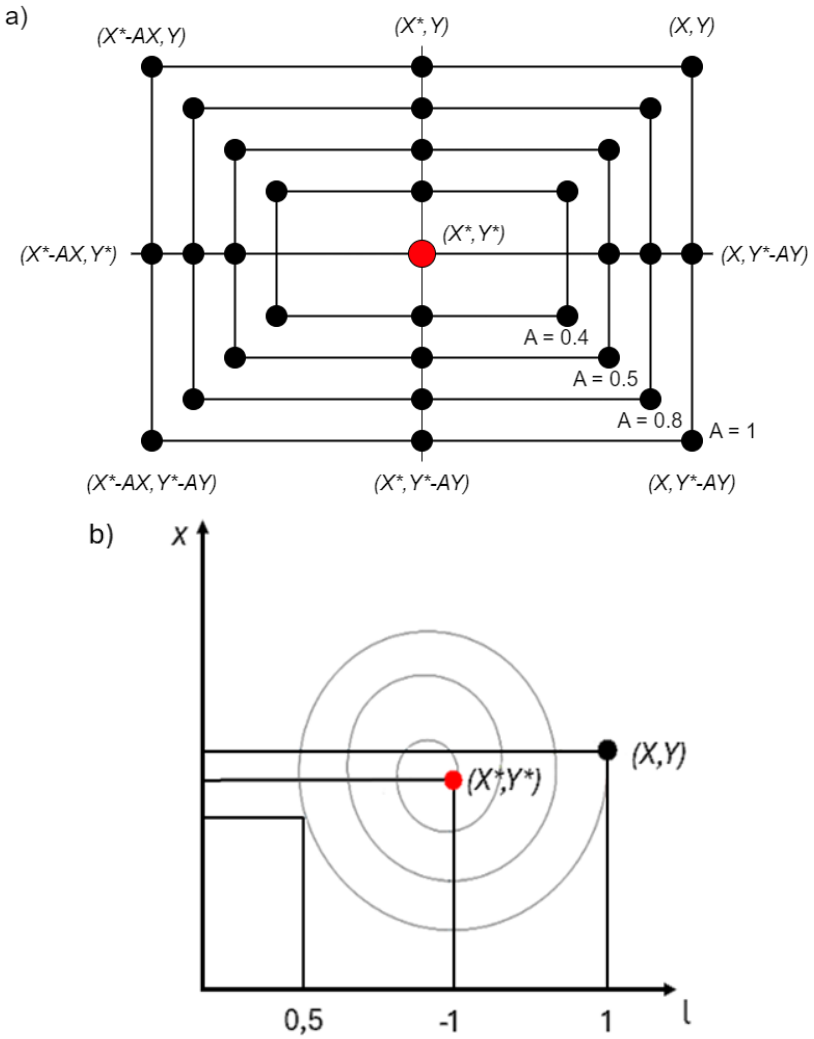

Figure 3.7: Bubble-net search mechanism implemented in WOA (X^* is the best solution obtained so far): (a) shrinking encircling mechanism and (b) spiral updating position.

The Python function that allows the position of each whale to be updated according to the rules just set out is:

```
def update_position(self, A, global_best_position, whales, C, l, p,
    i ):
    if p < 0.5:
        if abs(A) < 1:
            D = abs(C * global_best_position - whales[i])
            new_position = global_best_position - A * D #Eq. 3.9
```

```
        else:
            X_rand = np.random.uniform(-self.max_x, self.max_x)
            D = abs(C * X_rand - whales[i])
            new_position = C * X_rand - whales[i]      #Eq. 3.12
    else:
        D = abs(global_best_position - whales[i])
        new_position = global_best_position + D * math.exp(self.b *
            l) * math.cos(2 * math.pi * l) #Eq. 3.10
    return new_position
```

As far as convergence is concerned, throughout the entire iterative cycle, the population changes targets very quickly in the initial stage, where the domain is explored, while as the analysis proceeds, and thus $|A| < 1$, the convergence phase begins.

3.3 Algorithm comparison

The last section of this chapter is dedicated to the implementation of GA, WOA, and PSO for an optimization problem, with observation on the influence of algorithm parameters on compilation time and Means Square Error (MSE) on test data. The aim is to teach the network using supervised learning the curve of the sphere function by reducing the MSE between the predicted values and the real ones.

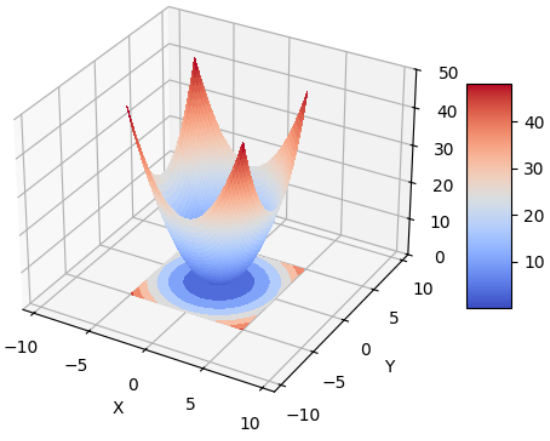

Figure 3.8: Three-dimensional plot of sphere function.

The sphere function, depicted in Figure 3.8, is a continuous, convex function that, in d dimensions, lends d local minima except for the global one, and

whose formula is

$$f(x) = \sum_{i=0}^{N} x_i^2. \tag{3.13}$$

For each algorithm, three different analyses were performed and 100 repetitions were made for each. The Table 3.1 shows the parameters that were changed in each simulation and the values of those held constant for all three algorithms.

Simulation	Iterations	Population	Domain
1	6 / 17	50	[-1.5, 1.5]
2	10	5 / 50	[-1.5, 1.5]
3	15	40	[-1, 1] / [-10, 10]

Table 3.1: Parameter values for each simulation.

In the first, the number of iterations to train the network varied casually between 6 and 17, in the second, the population took on a size between 5 and 50 individuals, while in the third, the initialization space of the parameters characterizing each individual varied randomly between [-1, 1] and [-10, 10].

Iteration	**GA**	**PSO**	**WOA**
Means	11,49	12,22	10,93
Population	**GA**	**PSO**	**WOA**
Means	25,68	27,10	25,09
Domain	**GA**	**PSO**	**WOA**
Means	4,68	4,88	5,45

Table 3.2: Comparison of averages in each simulation and for each algorithm.

As can be seen from Table 3.2, it is possible to make comparisons between the algorithms as the averages of the variable parameters associated with each simulation made on the three algorithms are comparable.

This is a sufficiently simple problem, so the network used also has an uncomplicated structure. It is a multilayer Artificial Neural Network with two nodes in the input layer, five nodes in the hidden layer, and one in the output layer. The activation function is linear, i.e., the activation is proportional to the input.

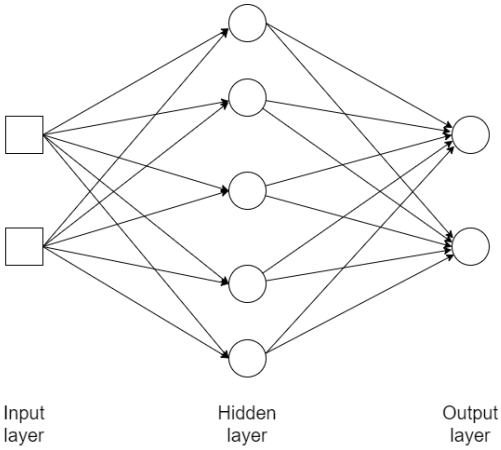

Figure 3.9: Structure of the ANN.

Now that the tools used have been described, it is possible to proceed by making the necessary observations. In all three algorithms, an increase in compilation time can be seen as the number of iterations and population size rise (Graphics a, c). This is because the two parameters have an effect that directly affects the executive process of network training. In other words, the number of data to be analyzed in the simulations increases as the two parameters grow. As can be seen from the graphs reporting the variation of population and iterations over time, the curve depicted can be approximated to a linear function, the equations of which are given in the Table 3.3.

	ITERATION	**POPULATION**
PSO	$y = 7.6325x - 2.0838$	$y = 1.564x + 0.4287$
GA	$y = 8.9716x + 5.9151$	$y = 2.1893x + 4.5769$
WOA	$y = 7.4217x - 0.0802$	$y = 1.3468x - 0.0681$

Table 3.3: Comparison between the linear function that describes the variation of iteration and population over time for each algorithm.

A clear distinction can be made between the variation of training iterations and population parameters. It can be seen from the angular coefficient exhibited by all three algorithms: the variation in the number of iterations has considerably more influence than the variation in the number of individuals. This is due to the fact that more repetitions imply the execution of more calculations than the introduction of a few individuals. Hence the considerable difference in influence.

As regards the influence of the two parameter variation on the MSE in the test data, the three algorithms exhibit again the same behavioral pattern

(Graphics b, d). Increasing iterations has no significant effect in this context, while the population size shows that, with a few individuals, there are probabilities of not achieving the best performance, which is instead guaranteed with a large population. Indicatively, the threshold is 20 individuals.

As far as the variations domain is concerned, it does not seem to have an influence on the compilation time but considerably on the quality of the results (Graphs e, f). With all the algorithms, in fact, the size of the space to be analyzed is a determining factor in the training of the network, which is less efficient as the domain increases. Intuitively, in fact, as the scanning space expands, changes must also be made to other parameters of the networks in order to help achieve convergence. Iterating the algorithms several times and increasing the number of individuals is an excellent solution.

Although the three algorithms exhibit the same reactions to parameter variations, the results they provide are not homogeneous. Therefore, their performance will now be observed in more detail. Table 3.4 shows the significant average values of time taken and MSE obtained from each of the three simulations.

SIMULATION 1			
	GA	PSO	WOA
TIME	109,0101	90,85578	81,03931
SIMULATION 2			
	GA	PSO	WOA
TIME	60,81938	43,14038	33,72336
SIMULATION 3			
	GA	PSO	WOA
MSE	31,11409	17,47451	6,272245

Table 3.4: Comparison of GA, PSO, and WOA across simulation 1, simulation 2, and simulation 3.

As can be seen, the most efficient algorithm in each simulation was WOA. As this algorithm naturally has a circular trajectory analysis strategy, it was easier to learn the spherical function. GA, on the other hand, is intuitive in that, although it is able to achieve convergence in the first two tests and is able to obtain excellent results in the third with some parameter modifications, it is not the most efficient algorithm for this optimization problem, given the long compilation times and the high average MSE. For PSO, its performance is acceptable but not the best. The fact of not having a movement feature that facilitates optimization makes it less performant than WOA.

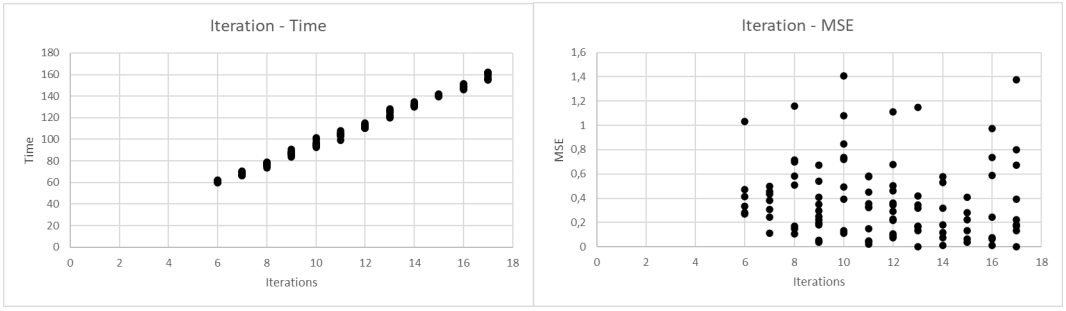

Figure 3.10: Algorithm: GA. Parameter: Iteration (6-17). a) Iteration - Time, b) Iteration - MSE.

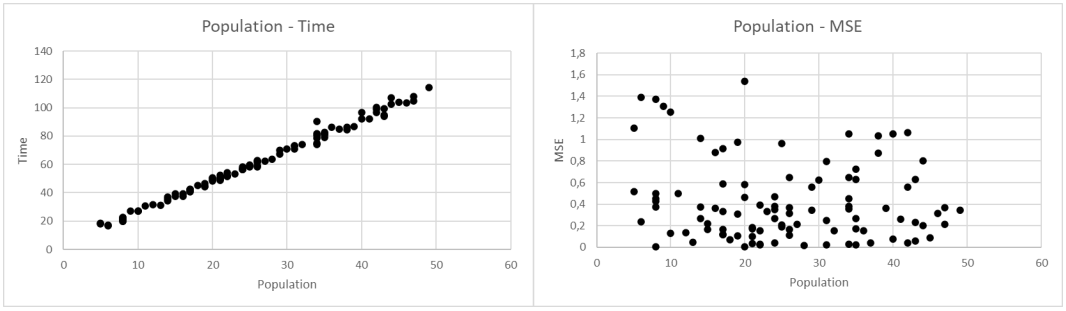

Figure 3.11: Algorithm: GA. Parameter: Population (5-50). c) Population - Time, d) Population- MSE.

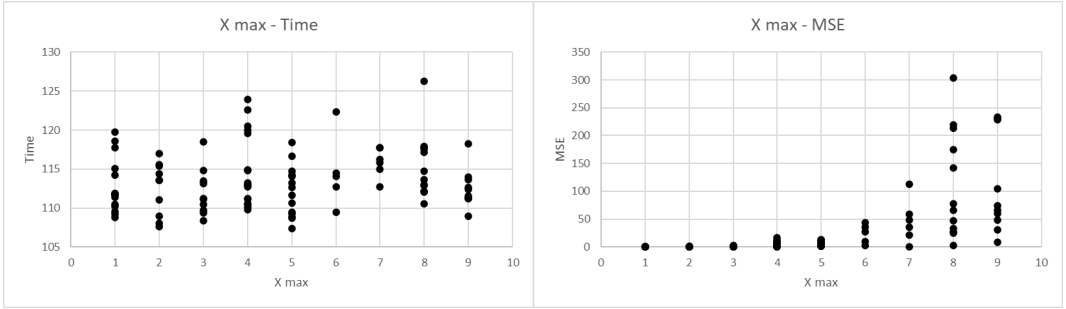

Figure 3.12: Algorithm: GA. Parameter: Domain (1-9). e) Domain - Time, f) Domain - MSE.

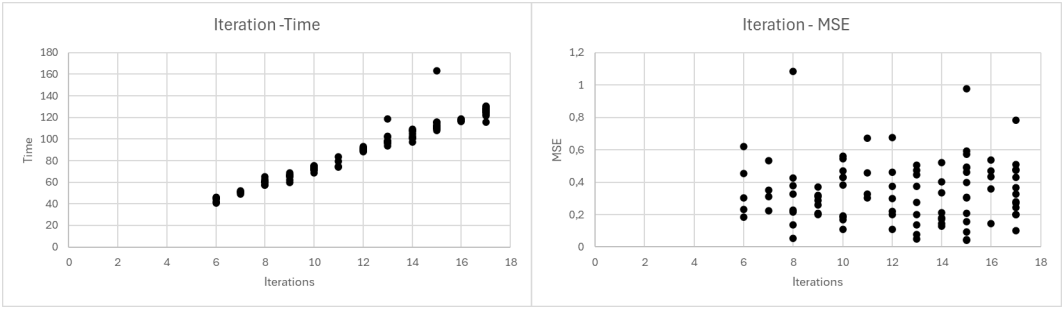

Figure 3.13: Algorithm: PSO. Parameter: Iteration (6-17). a) Iteration - Time, b) Iteration - MSE.

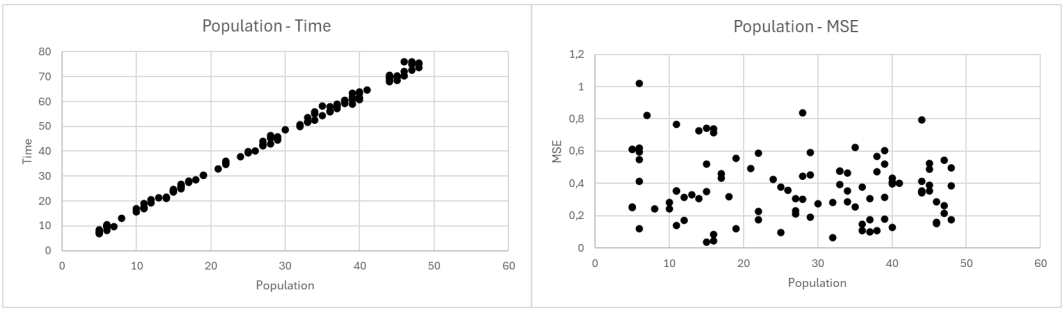

Figure 3.14: Algorithm: PSO. Parameter: Population (5-50). c) Population - Time, d) Population- MSE.

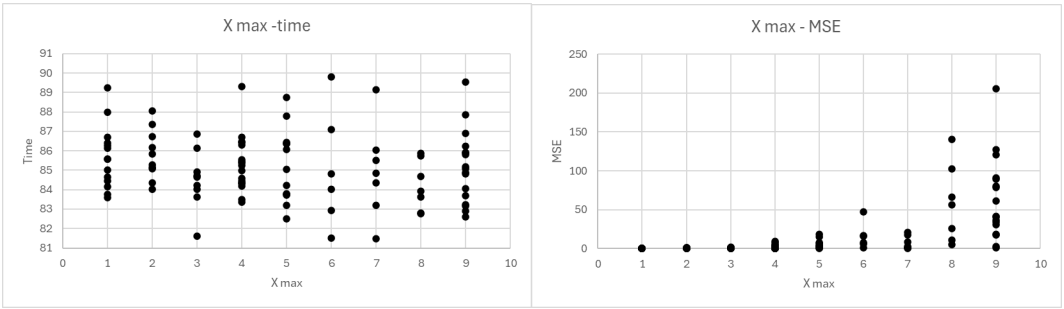

Figure 3.15: Algorithm: PSO. Parameter: Domain (1-9). e) Domain - Time, f) Domain - MSE.

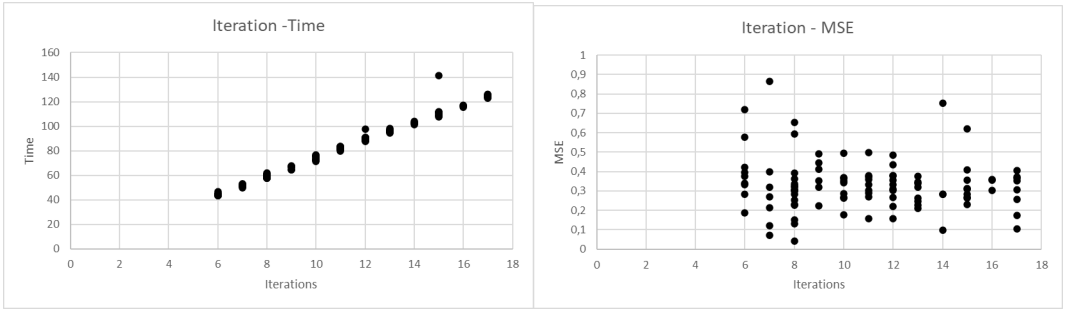

Figure 3.16: Algorithm: WOA. Parameter: Iteration (6-17). a) Iteration - Time, b) Iteration - MSE.

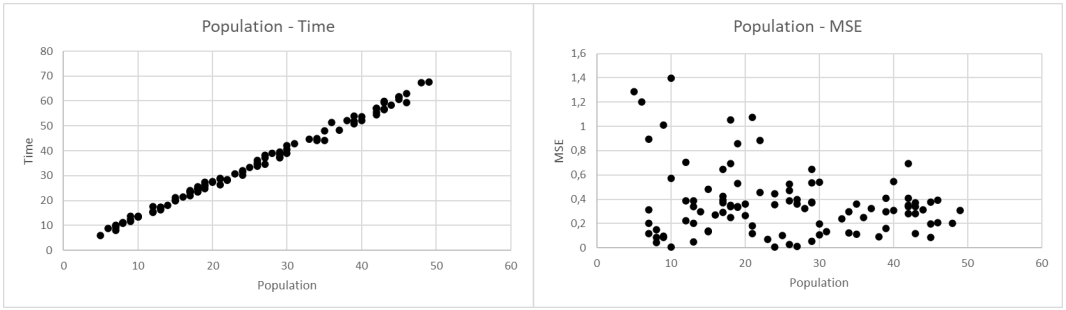

Figure 3.17: Algorithm: WOA. Parameter: Population (5-50). c) Population - Time, d) Population- MSE.

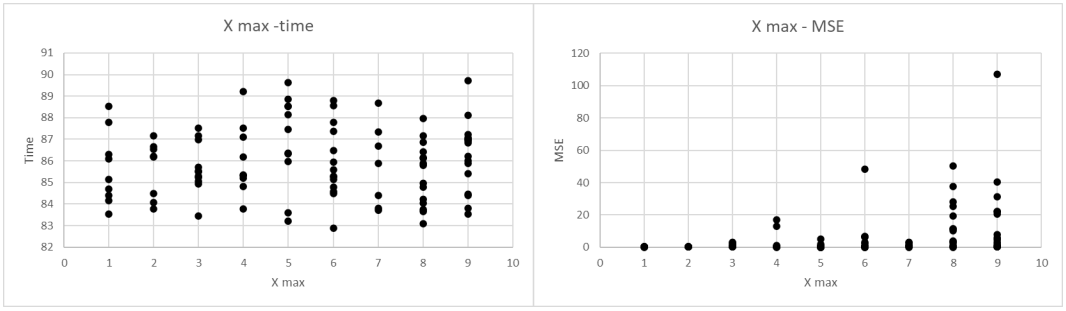

Figure 3.18: Algorithm: WOA. Parameter: Domain (1-9). e) Domain - Time, f) Domain - MSE.

Future studies could be done using so-called hybrid versions of algorithms, which, by their nature, have optimization strategies that mix the strengths of different algorithms, allowing for a fairer balance of exploration and exploitation.

The source code to see how the network, algorithms and data have been implemented is available at the link:

`https://github.com/Matteo7100/THESIS_PROJECT`

Chapter 4

AI for scientific advances

In this chapter, the main topic of the thesis is presented. The focus is on the use of AI as a guide tool for scientific discoveries. This is still a field of study in its infancy, which therefore allows for the development of numerous future works. In particular, the topic of using AI for the generation of scientific hypothesis is addressed through the explanation of the symbolic regression problem and the presentation of its application to the algebraic structures. In conclusion, related and future works about this scope are described.

4.1 Hypothesis space and Symbolic regression

Until a few years ago, the relationship between humans and AI was of pure convenience. Its fields of application were in fact optimizing parameters and functions and automating procedures to collect, visualize, and process data. Although it has become an indispensable tool in research, the performance of these tasks has not gone beyond the sphere of human skills and capabilities. In other words, the inferences that Artificial Intelligence has so far been able to make are, albeit with difficulty, within the reach of human capabilities. The new frontier of research is to revolutionise this relationship by harnessing the power of AI to enable humans to overcome their limitations.

More specifically, human limitations lie in navigating the space of hypothesis that characterizes every scientific problem, the vastness of which makes systematic exploration unfeasible. To give an example, in biochemistry it is estimated that there are 10^{60} drug-like molecules to be explored

[24]. In the 2010s, the availability of large datasets, aided by fast and massively parallel computing and storage hardware (graphics processing units and supercomputers) and coupled with new algorithms, has increased the power of AI methods. Among the most relevant methods is deep representation learning, in particular multilayer neural networks capable of identifying essential features that can simultaneously solve many tasks underlying scientific problems.

One of the techniques used to move within the space hypothesis is *symbolic regression*, which consists of finding a mathematical model capable of describing the observed data. To understand, suppose having access to a dataset containing the astronomical coordinates of the planet Mars and wanting to find the geometric curve that best describes its orbit. By making several attempts with various equations, it is concluded that its trajectory has an elliptical shape. This procedure was carried out analytically by Jhoannes Kepler in 1601, who took four years and about forty attempts to reach this result. It can therefore be guessed that in order to apply this technique to more complex systems and find new models in human times, AI support is necessary. More specifically, to explain symbolic regression, suppose a table of numbers is given, whose rows are of the form $(x_1, x_2, \cdots, x_n, y)$ where $y = f(x_1, x_2, \cdots, x_n)$. The task is to discover the correct symbolic expression for the unknown mystery function f, optionally including the complication of noise. In cases where f is a linear combination of known functions of monomials in $(x_1, x_2, \cdots, x_n)$ the problem reduces to a linear regression, the resolution of which is often handily solved, for example, ranging from Fourier expansions to wavelet transforms. Let now consider a much more complex case, in which the equations of the linearly independent system are strings of symbols. As the length increases, the number of combinations grows exponentially, so handling such a system is extremely difficult. It is a well-known combinatorial challenge, typical of many NP-hard problems. For reasons of an unknown nature, mystery functions f in many scientific contexts are often endowed with properties that allow the system to be simplified and thus arrive at a result.

In fact, it happens that f and its variables have known units, which allows confirmation and simplification through dimensional analysis. Furthermore, since f is in many cases of a low-degree polynomial nature, it can be determined by solving the system of linear equations from which the coefficients are derived. The third property that sometimes characterizes f is composi-

tionality, i.e., f is a composition of a small set of elementary functions, each typically taking no more than two arguments. Since it is also characterized by smoothness and symmetry, a feed-forward neural network is able to do the interpretation by using a smooth activation function and simplifying variables. Finally, the separability property that features f allows the primary system to be segmented into simpler subsystem.

4.2 GAN for algebraic structure

The case study of this thesis represents the combinatorial challenge per excellence. It involves analysing strings of operations of increasing length with an Artificial Neural Network capable of identifying the mathematical properties that characterize the algebraic structure. The project consists of two main phases: the writing of the equations and the recognition of the properties that characterize the operations. As this is an extremely innovative project, it requires considerable development before a fully functioning architecture can be achieved. Consequently, in starting this work, the focus was placed on the first task, which will be described in its critical aspect. In order to write the equations, it is intended to identify the two sequences of variables and operations whose values are identical. To accomplish this task, brute force can be used, i.e., manually writing down all possible combinations of variables and operations, selecting the verified ones, and thus constructing the dataset to complete the second task. However, this method is extremely inefficient, since as the number of operations that make up the algebraic structure increases, the number of possible combinations grows exponentially. As mentioned earlier, this brings the problem back to the NP-hard type. The alternative is based on the use of Generative Neural Networks. Two approaches have been attempted so far to find the best strategy. The first try consisted of using a GAN with a discriminator in the form of a cost function, to generate only the most interesting sequences to be used in the formulation of the equations.The first objective was to write an algorithm capable of learning the properties underlying the algebraic structure of the ring. The ring, therefore, consists of a set R over which two binary operations are defined, called sum and product, denoted by $+$ and $\cdot$ respectively, which satisfies the following three sets of axioms, called the ring axioms:

1. R is an abelian group under addition, meaning that:

- $(a+b)+c = a+(b+c)$ for all a, b, c in R (that is, $+$ is associative).
- $a+b = b+a$ for all a, b in R (that is, $+$ is commutative).
- There is an element 0 in R such that $a+0 = a$ for all a in R (that is, 0 is the additive identity).
- For each a in R there exists $-a$ in R such that $a+(-a) = 0$ (that is, $-a$ is the additive inverse of a).

2. R is monoid under multiplication, meaning that:

   - $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ for all a, b, c in R (that is, $\cdot$ is associative).
   - There is an element 1 in R such that $a \cdot 1 = a$ and $1 \cdot a = a$ for all a in R (that is, 1 is the multiplicative identity).

3. Multiplication is distributive with respect to addition, meaning that:

   - $a \cdot (b+c) = (a \cdot b) + (a \cdot c)$ for all a, b, c in R (left distributivity).
   - $(b+c) \cdot a = (b \cdot a) + (c \cdot a)$ for all a, b, c in R (right distributivity).

The strategy used to make GAN represent sequences of calculations consists in using an output layer with 5 nodes, which thus provides an output vector of size 5. Each index is associated with a pointer encoding the assigned operation or variable, as represented in the Table 4.1.

1	2	1	2	3	0
+	·	a	b	c	0

Table 4.1: Conversion of the network supplied values to sequence values.

The first two values of the vector encode the operation, while from the third onwards the corresponding variable is encoded. In addition, the sequences also have the element 0, to increase the complexity of the problem. The use of the first two indices as representatives of the operations is not accidental. The method of writing the sequence is in fact called *prefix notation*, or *polish notation*, which consists precisely of writing the operators before the operands. With this assumption, it is therefore possible to generate the sequences necessary to continue the formulation of the equations, which would then have to be evaluated on the basis of their veracity. However, several critical issues emerged when writing the cost function, which is the most delicate part of the process. During its writing, penalties were imposed to introduce the network to generate values within the acceptable ranges for

operations and operands, $[1, 2]$, and $[0, 3]$, respectively. This achieved the desired result, but a major problem typical of GANs arose. As explained in relation to Eq. 2.6, the network tends to converge to a small group of sequences, which, although the latter are valid, prevents a suitable dataset from being obtained for the next steps. It was thus thought to modify the cost function to stimulate the network to generate as many combinations of sequences as possible with the same frequency, thus discouraging convergence to a small group. This enhanced the network's outputs, allowing more sequences to be generated with each execution. Although there are difficulties in introducing the network to provide the desired output, the strategy outlined works and is therefore an excellent starting point towards the goal. Taking advantage of the introduction of a memory buffer to penalise according to the sequences omitted from iteration to iteration could be further improved. However, given the complexity of the problem with only two operations, which would certainly increase with the lengthening of the strings, it was decided to change the approach. Thanks to a program capable of generating all possible combinations of any length and coupling them to construct and evaluate equations, the strategy and the objectives have changed. Consider, for example, having all combinations of sequences with a number of operations strictly less than n. It is easily verified that the majority of the equations constructed are not correct. This evidence gives rise to the need for a neural network capable of generating true equations directly and uniquely using the newly developed code. The attempts made so far have made it possible to set up an initial strategy that has probed promising but needs numerous further elaborations.

The two cases of study just described constitute the first attempts to develop a design that can be extended far beyond the threshold of known algebraic structures, and the method for doing so is intuitive. In fact, thinking of using complex elements, such as matrices or vectors, as variables in the equations, of introducing more mathematical operations and in a larger number. By submitting such a dataset to the network, the latter would be able to formulate new as yet undiscovered relations and prove problems such as the best-known mathematical conjectures.

4.3 Related and future works

A solution to a symbolic regression task has been proposed by Silviu-Marian Udrescu and Max Tegmark [25]. Their model, called AI-Feynman, was able to identify the totality of 100 lows contained in "*Feynman Lecture on Physics*," demonstrating the viability of the project. Indeed, the two authors' interpretation of the properties described above (Paragraph 4.1) made it possible to obtain all the formulas sought from the dataset. Another area of application is Automatic Chemical Design [26]. Using an Artificial Neural Network called Variational Autoencoder (VAE), it was possible to create new molecules. In more detail, the molecules used to train the network are converted into latent space in the form of a continuous vector by the encoder, The predictor estimates the chemical properties that characterize them, while the decoder has the task of generating new ones by performing such things as decoding random vectors, perturbing known chemical structures, or interpolating between molecules. The continuous representation of molecules also allows the use of gradient-based optimization techniques to efficiently guide the search for optimized functional compounds. The latter method is already widely used and has been a major breakthrough for molecule design, as it is a complex and tedious practice to perform manually. However, its development still admits of numerous improvements to increase its efficiency, such as the introduction of a better verification criterion for stability of the generated structure. As far as the physical and mathematical sphere is concerned, the next step involves the implementation of increasingly complex mathematical operators, besides addition, product, subtraction, and division, and new algorithms that allow their simplification. In a less specific context, Artificial Intelligence constitutes an immense resource for scientific research, and therefore great attention is being paid to it. Indeed, since the available potential has been realized, great and new goals have been set. Achieving them, however, entails the evolution not only of computing techniques but also of other areas related to AI, such as research into new computer technologies.

Chapter 5

Conclusion

The present work explores how Artificial Intelligence, and more importantly Generative Adversarial Networks, nature-inspired metaheuristic algorithms, and symbolic regression, provide new ways to further scientific research. GANs, with their specific framework of generator-discriminator, have proven to be a powerful tool in data generation and modeling. Ability to model even very complex distributions like the ones without labeled data places them as a crucial component in unsupervised learning, especially on image synthesis and hypothesis exploration.

Indeed, a number of nature-inspired metaheuristic algorithms have been found to work well in the optimization of those neural networks including the Genetic Algorithm, Particle Swarm Optimization, and Whale Optimization Algorithm. Nature-inspired algorithms simulate natural processes as a means to explore high-dimensional complex optimization spaces. Comparing these algorithms, as done in this thesis, shows that while all have their merits, generally speaking, GA tends to provide better exploration capability, PSO converges faster, and WOA turns out to be a balanced approach in its spiral updating method. However, the choice of the algorithm completely depends upon the nature and requirement of the problem at hand and whether it requires more exploration or exploitation in the search space.

This research can be furthered by proposing a number of methods. First, hybrid optimization strategies that combine the strengths of various algorithms can be experimented on to enhance both exploration and convergence. Second, refine GAN architectures so that stability in training would

be further enhanced, preventing the mode collapse problems, for more reliable results. Third, more sophisticated symbolic regression techniques may be added so that the network is allowed to discover even more complex relations in scientific data.

Looking ahead, the integration of AI into high-level scientific research promises to revolutionize the field. AI's ability to explore vast hypothesis spaces and uncover hidden patterns offers the potential to discover new laws of nature that might otherwise remain inaccessible. The currently evolving AI will eventually accelerate the time of research in the implementation of scientific discovery and further yield the revolutionary insight continuum into physics, biology, and beyond. This is a new chapter where AI is a crucial partner in human intellectual advance.

Bibliography

[1] Das Amit Kumar, Dutt Saikat, and Chandramouli S.
Machine learning.
Pearson India Education Services Pvt. Ltd, 2019.

[2] Thomas Bayes.
An essay towards solving a problem in the doctrine of chances.
Royal Society, 1763.

[3] A. M. TURING.
I.—computing machinery and intelligence.
Mind, LIX(236):433–460, October 1950.
ISSN 0026-4423.
doi: 10.1093/mind/lix.236.433.
URL http://dx.doi.org/10.1093/mind/LIX.236.433.

[4] B. Yegnanarayana.
Artificial Neural Networks.
Prentice-Hall of India Private Limited, 2006.

[5] Massimo Bertolini, Davide Mezzogori, Mattia Neroni, and Francesco Zammori.
Machine learning for industrial applications: A comprehensive literature review.
Expert Systems with Applications, 175:114820, August 2021.
ISSN 0957-4174.
doi: 10.1016/j.eswa.2021.114820.
URL http://dx.doi.org/10.1016/j.eswa.2021.114820.

[6] Kevin Gurney.
An introduction to neural networks.
UCL Press Limited, 2004.

[7] Richard P. Lippmann.

An introduction to computing with neural nets.
ACM SIGARCH Computer Architecture News, 16(1):7–25, March 1988.
ISSN 0163-5964.
doi: 10.1145/44571.44572.
URL http://dx.doi.org/10.1145/44571.44572.

[8] Lars Ruthotto and Eldad Haber.
An introduction to deep generative modeling.
GAMM-Mitteilungen, 44(2), May 2021.
ISSN 1522-2608.
doi: 10.1002/gamm.202100008.
URL http://dx.doi.org/10.1002/gamm.202100008.

[9] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio.
Generative adversarial networks.
Communications of the ACM, 63(11):139–144, October 2020.
ISSN 1557-7317.
doi: 10.1145/3422622.
URL http://dx.doi.org/10.1145/3422622.

[10] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio.
Generative adversarial nets.
In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.
URL https://proceedings.neurips.cc/paper_files/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf.

[11] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, and Xi Chen.
Improved techniques for training gans.
In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.
URL https://proceedings.neurips.cc/paper_files/paper/2016/file/8a3363abe792db2d8761d6403605aeb7-Paper.pdf.

[12] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A. Bharath.
Generative adversarial networks: An overview.
IEEE Signal Processing Magazine, 35(1):53–65, January 2018.
ISSN 1558-0792.
doi: 10.1109/msp.2017.2765202.
URL http://dx.doi.org/10.1109/MSP.2017.2765202.

[13] Waseem Rawat and Zenghui Wang.
Deep convolutional neural networks for image classification: A comprehensive review.
Neural Computation, 29(9):2352–2449, September 2017.
ISSN 1530-888X.
doi: 10.1162/neco_a_00990.
URL http://dx.doi.org/10.1162/NECO_a_00990.

[14] Alec Radford, Luke Metz, and Soumith Chintala.
Unsupervised representation learning with deep convolutional generative adversarial networks, 2015.
URL https://arxiv.org/abs/1511.06434.

[15] Emily Denton, Soumith Chintala, Arthur Szlam, and Rob Fergus.
Deep generative image models using a laplacian pyramid of adversarial networks, 2015.
URL https://arxiv.org/abs/1506.05751.

[16] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros.
Unpaired image-to-image translation using cycle-consistent adversarial networks, 2017.
URL https://arxiv.org/abs/1703.10593.

[17] Xin□She Yang.
Engineering Optimization: An Introduction with Metaheuristic Applications.
Wiley, June 2010.
ISBN 9780470640425.
doi: 10.1002/9780470640425.
URL http://dx.doi.org/10.1002/9780470640425.

[18] Xin□She Yang.

Nature-Inspired Optimization Algorithms.
Elsevier, 2014.
ISBN 9780124167438.
doi: 10.1016/c2013-0-01368-0.
URL http://dx.doi.org/10.1016/C2013-0-01368-0.

[19] Randy L. Haupt and Sue Ellen Haupt.
Practical Genetic Algorithms.
Wiley, May 2003.
ISBN 9780471671749.
doi: 10.1002/0471671746.
URL http://dx.doi.org/10.1002/0471671746.

[20] J. Kennedy and R. Eberhart.
Particle swarm optimization.
In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4 of *ICNN-95*, page 1942–1948. IEEE.
doi: 10.1109/icnn.1995.488968.
URL http://dx.doi.org/10.1109/ICNN.1995.488968.

[21] Federico Marini and Beata Walczak.
Particle swarm optimization (pso). a tutorial.
Chemometrics and Intelligent Laboratory Systems, 149:153–165, December 2015.
ISSN 0169-7439.
doi: 10.1016/j.chemolab.2015.08.020.
URL http://dx.doi.org/10.1016/j.chemolab.2015.08.020.

[22] Jian Zhang, Jianan Sheng, Jiawei Lu, and Ling Shen.
Ucpso: A uniform initialized particle swarm optimization algorithm with cosine inertia weight.
Computational Intelligence and Neuroscience, 2021(1), January 2021.
ISSN 1687-5273.
doi: 10.1155/2021/8819333.
URL http://dx.doi.org/10.1155/2021/8819333.

[23] Seyedali Mirjalili and Andrew Lewis.
The whale optimization algorithm.
Advances in Engineering Software, 95:51–67, May 2016.
ISSN 0965-9978.

doi: 10.1016/j.advengsoft.2016.01.008.
URL http://dx.doi.org/10.1016/j.advengsoft.2016.01.008.

[24] Hanchen Wang, Tianfan Fu, Yuanqi Du, Wenhao Gao, Kexin Huang, Ziming Liu, Payal Chandak, Shengchao Liu, Peter Van Katwyk, Andreea Deac, Anima Anandkumar, Karianne Bergen, Carla P. Gomes, Shirley Ho, Pushmeet Kohli, Joan Lasenby, Jure Leskovec, Tie-Yan Liu, Arjun Manrai, Debora Marks, Bharath Ramsundar, Le Song, Jimeng Sun, Jian Tang, Petar Veličković, Max Welling, Linfeng Zhang, Connor W. Coley, Yoshua Bengio, and Marinka Zitnik.
Scientific discovery in the age of artificial intelligence.
Nature, 620(7972):47–60, August 2023.
ISSN 1476-4687.
doi: 10.1038/s41586-023-06221-2.
URL http://dx.doi.org/10.1038/s41586-023-06221-2.

[25] Silviu-Marian Udrescu and Max Tegmark.
Ai feynman: A physics-inspired method for symbolic regression.
Science Advances, 6(16), April 2020.
ISSN 2375-2548.
doi: 10.1126/sciadv.aay2631.
URL http://dx.doi.org/10.1126/sciadv.aay2631.

[26] Rafael Gómez-Bombarelli, Jennifer N. Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D. Hirzel, Ryan P. Adams, and Alán Aspuru-Guzik.
Automatic chemical design using a data-driven continuous representation of molecules.
ACS Central Science, 4(2):268–276, January 2018.
ISSN 2374-7951.
doi: 10.1021/acscentsci.7b00572.
URL http://dx.doi.org/10.1021/acscentsci.7b00572.