Neuroevolution of Generative Adversarial Networks

Chapter · May 2020

DOI: 10.1007/978-981-15-3685-4_11

4 authors:

Victor Franco Costa
University of Coimbra
16 PUBLICATIONS **44** CITATIONS
SEE PROFILE

Nuno Lourenço
University of Coimbra
50 PUBLICATIONS **207** CITATIONS
SEE PROFILE

João Nuno Correia
University of Coimbra
40 PUBLICATIONS **184** CITATIONS
SEE PROFILE

Penousal Machado
University of Coimbra
269 PUBLICATIONS **1,836** CITATIONS
SEE PROFILE

Neuroevolution of Generative Adversarial Networks

Victor Costa, Nuno Lourenço, João Correia, and Penousal Machado

Abstract Generative Adversarial Networks (GAN) is an adversarial model that became relevant in the last years, displaying impressive results in generative tasks. A GAN combines two neural networks, a discriminator and a generator, trained in an adversarial way. The discriminator learns to distinguish between real samples of an input dataset and fake samples. The generator creates fake samples aiming to fool the discriminator. The training progresses iteratively, leading to the production of realistic samples that can mislead the discriminator. Despite the impressive results, GANs are hard to train, and a trial-and-error approach is generally used to obtain consistent results. Since the original GAN proposal, research has been conducted not only to improve the quality of the generated results but also to overcome the training issues and provide a robust training process. However, even with the advances in the GAN model, stability issues are still present in the training of GANs. Neuroevolution, the application of evolutionary algorithms in neural networks, was recently proposed as a strategy to train and evolve GANs. These proposals use the evolutionary pressure to guide the training of GANs to build robust models, leveraging the quality of results, and providing a more stable training. Furthermore, these proposals can automatically provide useful architectural definitions, avoiding the manual discovery of suitable models for GANs. We show the current advances in the use of evolutionary algorithms and GANs, presenting state-of-the-art proposals related to this context. Finally, we discuss perspectives and possible directions for further advances in the use of evolutionary algorithms and GANs.

Victor Costa
CISUC, Department of Informatics Engineering, University of Coimbra, e-mail: vfc@dei.uc.pt

Nuno Lourenço
CISUC, Department of Informatics Engineering, University of Coimbra, e-mail: naml@dei.uc.pt

João Correia
CISUC, Department of Informatics Engineering, University of Coimbra, e-mail: jncor@dei.uc.pt

Penousal Machado
CISUC, Department of Informatics Engineering, University of Coimbra, e-mail: machado@dei.uc.pt

1 Introduction

Generative Adversarial Networks (GAN) [16] is an adversarial model that makes use of neural networks to produce samples based on an input distribution. GAN can be applied in several contexts, for example, in the generation of image, video, sound, and text, being able to produce impressive results concerning the quality of the created samples. This model gained a lot of relevance in recent years, leveraging the interest of the community on improving the original proposal.

The original GAN model proposes the use of two neural networks: a discriminator and a generator. These two components are trained in an adversarial manner, resulting in a zero-sum game between them. Each component of a GAN is trained following its specific process. To train the discriminator, a real data distribution, usually in the form of a dataset, is used as input. The generator is trained using a second distribution, such as a normal or a uniform distribution. This distribution forms the latent space that is used to feed the generator and produce samples with the objective of mimic the characteristics of the real data distribution. At the end of the training process, the generator learns to capture the real data distribution indirectly, i.e., without looking into the input dataset. Therefore, the generator will be able to create fake samples based on the learned distribution. On the other hand, the discriminator learns to distinguish between these fake samples and samples drawn from the real data distribution.

Despite the fact that GANs can be used as a generative component to produce samples in a variety of areas, applications in the image domain are more frequently reported by the production of realistic samples, representing significant advances when compared to other methods [3, 21, 51]. Therefore, the focus of this chapter is on the applications of GANs to the image domain. Nevertheless, the techniques presented here can be extended and adapted to other contexts.

Although GANs have attained incredible results, their training is challenging, and the presence of problems such as the vanishing gradient and the mode collapse is common [7, 13]. The balance between the discriminator and the generator is frequently the cause of these problems. In the case of the vanishing gradient, the discriminator becomes so powerful that it can distinguish almost perfectly between samples created by the generator and real samples. After this, because of the training approach used in GANs, the process stagnates. Regarding the mode collapse, the problem occurs when the generator fails to capture the entire representation of the distribution used as input to the discriminator. This is an undesired behavior, as we want not only to reproduce realistic samples but also to reproduce the diversity of the input distribution. Although there is a diversity of strategies and techniques to minimize the effect of these problems, they are still affecting the GAN training [17, 38]. Most of the proposed solutions appeal to mathematical models to deal with these problems, such as the use of more robust loss functions and stable neural network layers [3, 5, 26, 51]. Other proposals also worked on the architecture of the neural networks in order to avoid these issues [30, 34].

In spite of these issues, research was also conducted to improve the original GAN model with respect to the quality of the results, leveraging it to impressive levels

[3, 21, 26]. Other researches also proposed changes on the model to introduce a conditional input [20, 28, 31, 36]. Thus, a relevant effort is being made to improve GANs, not only to overcome the difficulties on the original model but also to extend the initial concept to different objectives[1].

In GANs, the adversarial characteristics and the necessity of an equilibrium between the generator and the discriminator make the design of the network crucial for the quality of the results. Therefore, the topology and hyperparameters that compose the neural networks of the generator and the discriminator are important to keep the balance between them in the training process. If one component becomes more powerful than the other, the GAN training will probably become unstable and may not produce the desired outcome. In this case, the design of the neural network is paramount to achieve convergence on training.

The design of a neural network is usually defined by hand in an empirical process, based on expert knowledge, which requires spending human time in repetitive tasks, such as experimentation and fine-tuning [7]. Experiments are used to validate and fine-tune the model, aiming to find efficient architectures to produce a neural network for a specific problem. However, some approaches can be used to automatize this process. In the field of evolutionary computation, neuroevolution can be used to design and optimize neural networks [27, 41, 50]. An evolutionary algorithm (EA) is based on the evolutionary mechanism found in nature, using it to evolve a population of potential solutions, producing better outcomes for a given problem [40]. In neuroevolution, this concept is adapted to the context of neural networks. In this case, the population is composed of individuals encoded through a genotype that represents, in some level of abstraction, neural networks. The genotype-phenotype mapping is the process that uses the genotype to produce the concrete representation of the neural network, which represents the phenotype of the individual. As in a regular EA, these individuals are evaluated and selected for reproduction to form the next generations of potentially better solutions.

Neuroevolution can be applied to evolve both the network architecture (e.g., topology, hyperparameters and optimization method) and the internal parameters (e.g., weights) [50]. NeuroEvolution of Augmented Topologies (NEAT) [41] is a well-known neuroevolution method that evolves the weights and topologies of neural networks. A further proposal originated DeepNEAT [27], a modification of the model that expands NEAT to larger search spaces, such as in deep neural networks.

Although neuroevolution is usually applied to standalone neural networks, the concepts can also be applied in the context of GANs. Furthermore, in the mechanics of the GAN model, the generator and discriminator are competing in a zero-sum game in the task of creating and discriminating fake and real samples. Therefore, a competitive model can be suitable to represent populations of individuals in GANs. In EAs, coevolution is the simultaneous evolution of at least two distinct species [19, 35, 42]. In competitive coevolution, individuals of these species are competing together, and their fitness function directly represents this competition. Thus, the

[1] A list of proposals related to GANs can be found at https://github.com/hindupuravinash/the-gan-zoo.

applicability of a competitive coevolution environment in an EA to train GANs can also be evaluated [9, 10, 14, 46].

In recent years, researchers have been applying the concepts of EAs to improve the performance of GANs with different strategies [1, 9, 10, 14, 47, 46]. The authors found advances not only in the quality of the outcome but also regarding the stability issues in the training of GANs. We present in this chapter the state-of-the-art of these proposals, discussing their main advantages and drawbacks, and presenting further directions for improvements. The following proposals will be described in this chapter: E-GAN [47], Pareto GAN [14], Lippizaner [1], Mustangs [46], and COEGAN [9, 10].

The remainder of this chapter is organized as follows. Section 2 introduces the concepts of GANs, presenting the challenges and advances in this field. Section 3 summarize the possibilities regarding the application of EAs in the context of GANs. Section 4 presents the current proposals that use EAs with GANs. Section 5 discuss the application of EAs in GANs, drawing particular attention to the drawbacks and advantages of each approach, presenting directions for further improvements. Finally, Sect. 6, concludes this chapter with the final considerations about the subject.

2 Generative Adversarial Networks

Generative Adversarial Networks (GAN), proposed by [16], is an adversarial model that became relevant mostly for the performance achieved in generative tasks on the image domain, representing significant improvements over other generative methods. We present in this section the model definition, the common issues found when training a GAN, and how to evaluate and compare GANs using state-of-the-art metrics.

2.1 Definition

A GAN combines two neural networks in a unified training algorithm: a discriminator D and a generator G. The discriminator D aims to distinguish between real and fake examples, given a real data distribution usually in the form of a dataset (e.g., a digits dataset). Therefore, the discriminator outputs the probability of the input to be a real sample, i.e., a sample belonging to the real data distribution. For this, the discriminator is trained with samples from both the real distribution and samples created by the generator. The generator G receives samples from another input distribution (e.g., a uniform or a normal distribution), and outputs a fake sample, attempting to capture the data distribution used in the training of D.

These components are trained in an adversarial manner, creating strong generative and discriminative components. The generator never looks directly into the distribution used to train the discriminator. Therefore, it is expected that the generator does

not output simple copies of the real distribution and presents some novelty on the samples in a successfully trained GAN.

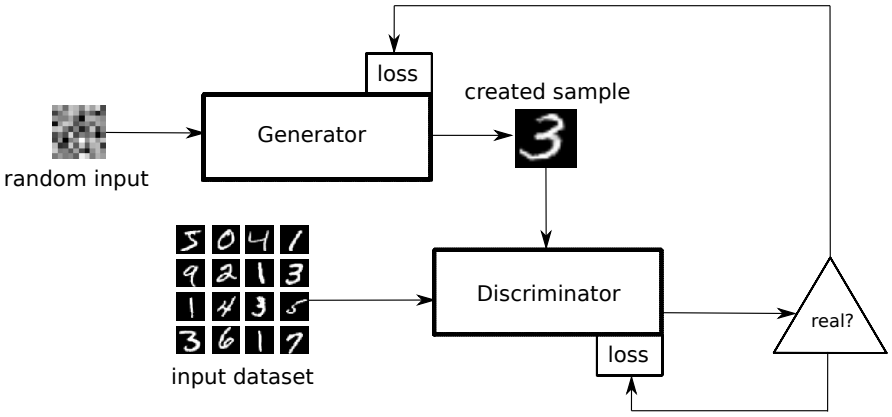

Fig. 1 High-level interaction between the components of a GAN trained with a digits dataset.

Figure 1 represents the high-level interaction between these components when using a digits input dataset, such as the MNIST dataset [24]. Note that the generator does not receive samples from the input dataset used on the discriminator training. Both the discriminator and generator are trained with backpropagation and a gradient descent method. Therefore, different loss functions are used in the GAN components. The loss function of the discriminator is defined as follows:

$$J^{(D)}(D,G) = -\mathbb{E}_{x\sim p_{data}}[\log D(x)] - \mathbb{E}_{z\sim p_z}[\log(1 - D(G(z)))]. \quad (1)$$

For the generator, the non-saturating version of the loss function is defined by:

$$J^{(G)}(G) = -\mathbb{E}_{z\sim p_z}[\log(D(G(z)))]. \quad (2)$$

In Eq. 1, p_{data} represents the dataset used as input to the discriminator. In Eq. 1 and Eq. 2, z is the latent space used as input to the generator, p_z is the latent distribution, G is the generator, and D represents the discriminator.

The classical training procedure of a GAN is presented in Algorithm 1. In lines 2-6, the discriminator is trained for k steps, using a batch of m samples. For performance reasons, the original GAN model used $k = 1$ for the experiments. The generator is trained in lines 7-8. The losses described by Equations 1 and 2 are applied in lines 5 and 8, respectively. The optimization method used for training can be any stochastic gradient descent method, such as the Adam optimizer [23] or RMSprop [45].

GANs are hard to train, and training stability is an issue that systematically affects the results. So, to achieve good outcomes in training, a trial-and-error approach is frequently used. Some works developed a set of techniques to train GANs to improve the probability to achieve convergence. A study about the training of GANs [38]

Algorithm 1 Classical training algorithm for GANs.

for n iterations **do**
  for k steps **do**
    $fake\_samples \leftarrow sample(m, P_z(g))$
    $real\_samples \leftarrow sample(m, P_{data}(x))$
    $train\_discriminator(D, G, fake\_samples, real\_samples)$
  end for
  $fake\_samples \leftarrow sample(m, P_z(g))$
  $train\_generator(G, D, fake\_samples)$
end for

proposes the use of strategies such as label smoothing and minibatch discrimination in order to obtain better results. However, these strategies only minimize the effect of the problems that usually happen in the training process. Several other variations of the original GAN model were proposed to improve the effect of these problems [3, 17, 21, 26, 30]. In Sect. 2.2, we describe some of these problems regarding the training of GANs.

2.2 Common Problems

The vanishing gradient and the mode collapse are amongst the most common problems affecting the stability when training GANs. They are widespread and represent a significant challenge to obtain useful representations for applying GANs in different domains. These issues are often part of a bigger problem: the balance between the discriminator and the generator during the training. Although several approaches tried to minimize those obstacles, they still affect the training and remain unsolved [3, 17, 38]. Following we describe the mode collapse and the vanishing gradient issues, presenting how they affect the training of GANs.

2.2.1 Mode Collapse

In the mode collapse problem, the generator captures only a small portion of the dataset distribution provided as input to the discriminator. This diminished representation is not desirable since it is expected that a generative model reproduces the whole distribution of the data to achieve variability on the output samples.

Figure 2 represents images created by a generator after a failed training of a GAN using the MNIST dataset. The effects of the mode collapse can be clearly seen in these images. We can see in the samples on the left of Fig. 2 that only the digits 9 and 7 are represented. However, in the samples on the right, the digits cannot be identified correctly. The generator creates only a superposed combination of digits.

Fig. 2 Samples created by a GAN after training that resulted in the mode collapse issue. Note that the GAN was training using the MNIST dataset, which contains digits from 0 to 9. However, on the left, the generator can only create samples related to the digits 7 and 9. In the right, the generator failed to create a real digit, outputting the same unrealistic pattern.

The lack of variability demonstrated in these examples characterizes the problem as mode collapse.

2.2.2 Vanishing Gradient

The vanishing gradient occurs when one of the GAN components, i.e., the discriminator or the generator, becomes powerful enough to harm the balance required on the training. For example, the discriminator can become too strong and not be fooled anymore by the generator when distinguishing between fake and real samples. Hence, the loss function is too small, the gradient does not flow through the neural network of the generator, and the GAN progress stagnates. In the GAN training, the equilibrium between the discriminator and generator is essential to the training convergence. The vanishing gradient problem happens when this equilibrium is violated in an irreversible way.

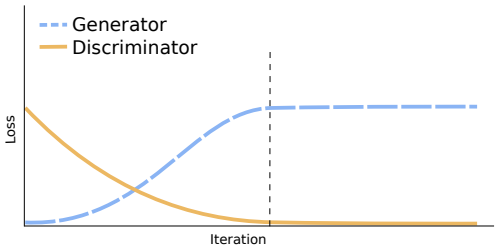

Fig. 3 Losses of the generator and discriminator of a training experiment with the vanishing gradient issue. As the loss of the discriminator approximates to zero, the loss of generator stagnates.

Figure 3 presents an example of a GAN training that suffers from the vanishing gradient problem. We can see in this figure the progression of losses of the generator and discriminator through iterations. Note that when the discriminator loss becomes zero (marked by the dashed vertical line), the generator stops to improve and stagnates until the end of the training. As such, the quality of samples created by the generator will not improve anymore. It is important to note that the divergence between the generator and discriminator, expressed by the losses, does not need to

always decrease [13]. Even when the loss increases, the training can reach a good solution in the end. Therefore, regarding the vanishing gradient, the problem only occurs when the loss approximates to zero. The GAN model tolerates steps with a reduction in the loss without losing convergence capabilities.

2.3 Evaluation Metrics

Several metrics can be used to quantify the performance of a GAN [6, 49]. As the generators are commonly the most relevant component of a GAN, these metrics usually target them. However, the measurement of the performance when executing generative tasks is a relevant problem and there is not a consensus yet in the community about the best metric to use. We highlight here two of the most commonly reported metrics for GANs in the literature: the Inception Score and the Fréchet Inception Distance (FID) score.

Other metrics, such as the skill rating [32], were evaluated and obtained relevant results. Despite this, they are still not widely used by the community, becoming hard to use them in a comparison study to evaluate a proposal with other works. However, they can still be useful to use in the context of EAs. They can be used not only as comparison criteria between the solutions but also as fitness functions to guide the evolution.

2.3.1 Inception Score

The Inception Score (IS) [38] is an automatic metric to evaluate synthetic image samples that were created based on an input dataset. This method uses the Inception Network [43, 44] to get the conditional label distribution of the images created by a generative algorithm, such as a GAN. This network should be previously trained using a dataset, usually the ImageNet dataset [37]. Therefore, the Inception Score is defined as:

$$IS(x, y) = exp(\mathbb{E}_x KL(p(y|x)||p(y))), \tag{3}$$

where x is the input data, y is the label of the data, $p(y)$ is the label distribution, $p(y|x)$ is the conditional label distribution, and KL is the Kullback–Leibler divergence between the distributions $p(y|x)$ and $p(y)$. It is recommended to evaluate the IS metric on a large number of samples, such as 50000, in order to provide enough diversity to the score [38].

The IS metric has some drawbacks, such as the sensitivity to the weights of the Inception Network used in the calculation [4]. Moreover, the network used in the Inception Score, which was trained in the ImageNet dataset, may not be applicable with consistent performance to other datasets.

2.3.2 Fréchet Inception Distance

Fréchet Inception Distance (FID) [18] is the state-of-the-art metric to compare the generative components of GANs. The FID score outperforms other metrics, such as the Inception Score, with respect to diversity and quality [25]. As in the Inception Score, FID also uses a trained Inception Network in the computation process. In the FID score, a hidden layer of Inception Net (also usually trained on ImageNet) is used in the transformation of images into the feature space, which is interpreted as a continuous multivariate Gaussian. This transformation is applied to a subset of the real dataset and samples created by the generative method. The mean and covariance of the two resulting Gaussians are estimated and the Fréchet distance between these Gaussians is given by:

$$FID(x, g) = ||\mu_x - \mu_g||_2^2 + Tr(\Sigma_x + \Sigma_g - 2(\Sigma_x \Sigma_g)^{1/2}). \quad (4)$$

In Eq. 4, μ_x, Σ_x, μ_g, and Σ_g represent the mean and covariance estimated for the real dataset x and fake samples g, respectively. In summary, the FID score is given by the norm of the means and the trace of the covariances between real and fake samples.

3 Exploring the Evolution of GANs

Several aspects that compose the GAN model can be actively used as evolvable components in an evolutionary algorithm. However, it is important to keep in mind that the EA should preserve the balance of these components in order to tackle the issues listed in Sect. 2.2. We discuss in this section the possibilities for the application of EAs to the GAN model. The options related to neuroevolution and the aspects of GANs will be presented as possible choices to design an algorithm.

3.1 Neuroevolution

Neuroevolution is the application of EAs in the evolution of a neural network. It can be applied to evolve weights, topology, and hyperparameters of a neural network [50]. When used to discover the network topology, a substantial benefit is the automation of the architecture design and parameter decision, transforming a manual human effort into an automatic procedure. This automation is even more critical with the rise of deep learning, which is producing deeper models and increasing the search space [27]. However, the increase in the search space is also a challenge for neuroevolution. These methods have high time-consuming executions that may turn their application unfeasible.

Neuroevolution can be fully applied in the context of GANs. The evolution of the topologies of the discriminator and the generator should take into account that the equilibrium between them is paramount to the convergence of the training process. Not only the structure (i.e., the number of layers and the connections between them) but also the internal characteristics of each layer composing a neural network can be the subject of evolution. For example, the type of a layer (e.g., convolution or fully connected), the number of output features, and the activation function (e.g., ReLU, ELU, Tanh). Other aspects relevant to the network can also be a variable of the individual, such as the choice for the optimizer used in the training, the learning rate, the batch size, and the number of the training iterations (e.g., the k parameter of Algorithm 1).

We can also make use of other techniques regarding evolutionary computation in neuroevolution, such as coevolution. Coevolution is the simultaneous evolution of at least two distinct populations (also denominated species) [19, 35]. There are two types of coevolution algorithms: cooperative and competitive. In cooperative coevolution, individuals of different species cooperate in the search for efficient solutions, and the fitness function of each species is designed to reward this cooperation. In competitive coevolution, individuals of different species are competing between them in the search for better solutions. Here, their fitness function directly represents this competition in a way that scores between species are inversely related. For example, NEAT was successfully applied to a competitive coevolution environment [42].

The coevolutionary approach used in an EA can lead to some issues, such as intransitivity and disengagement [2, 29]. The intransitivity occurs when a solution a is better than b and b is better than c, but this does not guarantee that a is better than c. This issue can lead to cycling between these solutions during the evolutionary process, preventing the progress of individuals toward optimal solutions. Disengagement occurs when the equilibrium between the populations is broken. In this case, individuals from one population are much better than individuals from the other, leading to ineffective progression.

GANs can be modeled as a competitive coevolution problem. We can consider a population of discriminators as competitors to a population of generators. Therefore, an EA can make use of competitive coevolution concepts to match individuals from these two populations at the evaluation phase. Furthermore, we can relate problems that frequently affect the training of GANs (Sect. 2.2) to coevolution problems. For example, the vanishing gradient can be linked to the disengagement issue. Thus, the use of coevolution can be explored in combination with other techniques (e.g., neuroevolution) to solve challenges of the GAN training process.

3.2 Variations of GANs

Several advances over the original GAN model were recently proposed. These proposals focused not only on the improvement of the quality of the created samples

but also on the improvement of the training stability. These proposals can be divided into two main categories: architecture improvements and alternative loss functions [33, 48].

In the category of architecture improvements, we have DCGAN [34], a set of constraints and rules that guide the design of the components of a GAN. DCGAN became a reference architecture for the discriminator and the generator in GANs. Some of these rules are:

- Use batch normalization in the generator and discriminator;
- Use the ReLU activation function in all hidden layers of the generator;
- Use LeakyReLU in all layers of the discriminator.

In the experiments presented with DCGAN, the training stability was improved, but there are still issues such as the mode collapse problem in some executions [34].

Other proposals introduced different aspects into the original GAN model [5, 7, 11, 12, 15, 21, 22, 51]. We can use some of these strategies as inspiration for an EA. For example, the method described in [21] uses a predefined strategy to grow a GAN during the training procedure. The main idea is to grow the model progressively, increasing layers in both discriminator and generator. This mechanism will make the model more complex while the training procedure runs, resulting in the generation of higher resolution images at each phase. However, these layers are added progressively in a preconfigured way, i.e., they are not produced by a stochastic procedure. These concepts can be expanded to be used in an EA. Instead of a predefined grow, the progression of the discriminator and the generator can be guided by evolution, using a fitness function that can prevent and discard unfitted individuals.

Other approaches use multiple components instead of only a single generator and a single discriminator. For example, GMAN [11] proposed a model that uses multiple discriminators in the training algorithm. On the other hand, MAD-GAN [15] explored the use of multiple generators in the GAN training. An EA can be aligned with these concepts with the proposal of a solution that contains two entirely different populations of discriminators and generators.

Another strategy to overcome the training issues and improve the original GAN model is the use of alternative loss functions. A variety of alternative loss functions were proposed to minimize the problems and leverage the quality of the results, such as WGAN [3], LSGAN [26], and SN-GAN [30]. WGAN proposes the use of the Wasserstein distance to model the loss functions. LSGAN uses the least-squares function as the loss for the discriminator. SN-GAN proposes the use of spectral normalization to improve the training of the discriminator. An EA can take advantage of these variations and use the loss function as an interchangeable component.

4 Current Proposals

We present in this section the state-of-the-art on the application of evolutionary algorithms in GANs. These proposals are aligned with the possibilities presented in

Sect. 3, presenting solutions to apply them and improve the GAN training process. To the best of our knowledge, these are the proposals that use EAs in the context of GANs: E-GAN [47], Pareto GAN [14], Lippizaner [1], Mustangs [46], and CO-EGAN [9, 10]. In this section we describe these solutions, focusing on the choices concerning the aspects of the EA and the characteristics of GANs. Therefore, we report the characteristics of the algorithms concerning the selection method, fitness functions, variation operators, evaluation, and experiments.

4.1 E-GAN

A model called E-GAN[2] was proposed to use EAs in GANs [47]. The approach applies an EA to GANs using a mutation operator that can only switch the loss function of the generator. Therefore, the evolution occurs only in the generator, and a single-fixed discriminator is used as the adversarial for the population of generators. The network architectures for the generator and the discriminator are fixed and based on DCGAN [34].

The population of generators contains individuals that have different loss functions. Therefore, the mutation operator used in the process can change the loss function of the individual to another one selected from a predefined set. Each loss function in the predefined set focused on an objective to help in the GAN learning process. A minimal population of individuals is used to capture the possibilities of the predefined losses and provide an adaptive objective for the training. In this case, the population of generators is composed of three individuals, each one representing one of the possible losses.

The possibilities for losses are implemented through three mutation operators: minimax, heuristic, and least-squares mutation. The minimax mutation follows the original GAN objective given by Eq. 2, minimizing the probability of the discriminator to detect fake samples. On the other hand, the heuristic mutation aims to maximize the probability of the discriminator to make mistakes regarding fake samples. The least-squares mutation is based on the objective function used in LSGAN [26]. Only these operations are available and crossover is not used in the E-GAN algorithm.

Two criteria were used as fitness in the evaluation phase of the algorithm. The first, called quality fitness score, is defined as:

$$F_q = \mathbb{E}_z[(D(G(z)))], \tag{5}$$

that is similar to the loss function used in the generator of the original GAN model (Eq. 1). The second criteria, called the diversity fitness score, is defined as:

$$F_d = -\log ||\nabla_D - \mathbb{E}_x[\log(D(x))] - \mathbb{E}_z[\log(1 - D(G(z)))], \tag{6}$$

[2] Code available at https://github.com/WANG-Chaoyue/EvolutionaryGAN.

In Eq. 5 and Eq. 6, z, G and D represent the latent space, the generator, and the discriminator, respectively. These two fitness criteria are combined as follows:

$$F = F_q + \gamma F_d, \tag{7}$$

using the γ parameter to regulate the influence of the diversity criteria on the final fitness.

At each generation, the individuals are evaluated following their specific loss function, and only the best-fitted generator survives for the next steps. In the next generation, the survivor individual is used to train the discriminator and to generate the three children for the next evaluation.

The E-GAN model was evaluated on the CIFAR-10, LSUN and CelebA datasets. The Inception Score was used as the metric to analyze the results. As specified in E-GAN, the population used in the experiments consist of a single discriminator and three generators. The authors concluded that E-GAN improved training stability and achieved satisfactory performance, outperforming other methods in some scenarios.

4.2 Pareto GAN

A neuroevolution approach to train GANs was proposed in [14]. Although not named by the authors, we refer to this solution as Pareto GAN[3]. The proposal uses a genetic algorithm to evolve the architecture of the neural networks used for both the generator and the discriminator. A single individual (G_i, D_i) is used to represent both the generator and the discriminator in the EA.

The crossover operator combines two parents exchanging the discriminator and the generator between them. For example, a crossover between the individuals (G_1, D_1) and (G_2, D_2) produces the children (G_1, D_2) and (G_2, D_1). The crossover operator does not change the internal state of the generator and the discriminator in each individual. To accomplish this, a set of possible mutations is applied to individuals when creating a new generation.

Regarding the architecture of the neural networks, the mutation can change, add or remove a layer. Mutation can also change the internal state of a layer, such as the weights or the activation function. Some mutation operators also work on the GAN algorithm level. There is an operator to change the loss function used in the GAN algorithm by using a predefined set of possibilities. Another possibility is to change the characteristics of the algorithm. Here, it is possible to change the number of iterations for the generator and the discriminator when training an individual (e.g., the parameter k of Algorithm 1).

A benchmark for GANs based on the problem of Pareto set approximations was also proposed [14]. The comparison between the Pareto front of a solution and the real front is used to assess the quality of the samples and can also identify issues, such as the mode collapse problem. Therefore, the inverted generational distance (IGD)

[3] Code available at https://github.com/unaigarciarena/GAN_Evolution.

[8] was used as fitness to drive the EA. The IGD measures the smallest distance between points in the true Pareto front and in the Pareto front approximation and is given by:

$$IGD = \frac{1}{|R|}\left(\sum_{r \in R} min_{a \in A} d(r,a)^p\right)^{\frac{1}{p}}, d(r,a) = \left(\sum_{k=1}^{m}(r_k - a_k)\right)^{\frac{1}{2}}, \quad (8)$$

where R is the real Pareto front, A is the Pareto approximation, and m is the number of vectors in R.

The evaluation phase will transform each individual (G_i, D_i) into a concrete GAN, composed of a discriminator and a generator, that will be trained according to the regular GAN algorithm. The fitness is calculated, and the selection uses the Pareto-dominance to compose the offspring that will form the next generation.

The proposed solution was evaluated using bi-objective functions as the input data, each one with 10 input variables. A population of 20 individuals, evaluated for 500 generations, was used in the experiments. The authors found that the algorithm was able to found architectures for the discriminator and the generator that improve the Pareto set approximation. The experiments do not include evaluations with image datasets. However, experiments using the same data dimension as the MNIST dataset, i.e., with 784 input variables, were also conducted. The authors concluded that the solution is scalable to this dimension, as the results showed that useful architectures were also found in this case.

4.3 Lippizaner

A model called Lipizzaner[4] defines a coevolutionary framework to train and evolve GANs [1]. In Lipizzaner, the evolution occurs only on the internal parameters of the generator and discriminator, such as the weights of their neural networks. Thus, the network architecture used in both the discriminator and generator is fixed and defined *a priori*. The architecture varies with the dataset used in the experiments. For MNIST, an MLP network composed of four layers and 700 neurons was used. On the other hand, an architecture based on DCGAN was used for the experiments with the CelebA dataset.

The fitness used in Lippizaner for the generators and discriminators is based on the GAN objective function, a variation of Eq. 1 (Sect. 2). At the evaluation step, the value $\mathcal{L}(G_i, D_i)$ is calculated for each pair G_i D_i, and the fitness values are updated as f_{G_i} $-= \mathcal{L}(G_i, D_i)$ and f_{D_i} $+= \mathcal{L}(G_i, D_i)$ for the generator and the discriminator, respectively.

Spatial coevolution was used to design the algorithm that trains and evolve the generators and discriminators. Individuals are distributed over a two-dimensional toroidal grid, where each cell contains individuals from the generator and discrimi-

[4] Code available at https://github.com/ALFA-group/lipizzaner-gan.

Fig. 4 A 3×3 grid representing the spatial coevolution mechanism used in Lipizzaner. The neighborhood of the central cell includes the four-highlighted nodes in the grid. Each cell contains one discriminator, one generator, and a mixture of weights.

nator populations. In the evaluation phase, the EA matches individuals in neighbor cells following a coevolutionary pairing approach. A five-cell neighborhood was used to determine these interactions. Figure 4 displays an example of a $3x3$ grid with the spatial coevolution strategy used in Lipizzaner. The generator is determined as a mixture of generators in this neighborhood.

Lipizzaner uses two mutation operators. The first operator mutates the learning rates of the optimization method used in the generator and the discriminator. In this case, a normal distribution is used to change the learning rate at small steps at each generation. The second operator is a gradient-based mutation that updates the weights of the individuals in the populations of generators and discriminators. This operator uses the Adam optimizer [23] to update the weights together with an evolution strategy to update the mixture weights.

The model was evaluated on the MNIST and CelebA datasets, using a 2×2 grid, forming a population of 4 generators and 4 discriminators. These populations were evolved through 400 generations. The authors found that Lipizzaner was able to avoid the mode collapse problem in most of the experiments. The model can recover from the mode collapse issue and continue to improve as the training advances through the next generations.

4.4 Mustangs

The models E-GAN and Lipizzaner were combined in a hybrid approach to train and evolve GANs, called Mutation Spatial GANs (Mustangs)[5] [46]. As in Lipizzaner and E-GAN, the topologies of the generator and discriminator are fixed during the algorithm, i.e., the architectures are not a target of the EA.

Mustangs combines the mutation operators used in E-GAN and the spatial coevolution mechanism used in Lipizzaner. The goal is to leverage the diversity of genomes in the population. Thus, the loss function of generators can be modified by

[5] Code available at https://github.com/mustang-gan/mustang.

the mutation operator, as in E-GAN. As in Lipizzaner, the match between individuals occurs in a toroidal grid, and the internal weights of the neural networks are calculated based on the neighborhood.

The Mustangs model uses the same fitness strategy used in Lipizzaner, i.e., the fitness is based on the GAN objective function, a variation of Eq. 1 (Sect. 2). Thus, at the evaluation step, the value $\mathcal{L}(G_i, D_i)$ is also calculated for each pair G_i D_i, and the fitness values are also updated as f_{G_i} $-=$ $\mathcal{L}(G_i, D_i)$ and f_{D_i} $+=$ $\mathcal{L}(G_i, D_i)$ for the generator and the discriminator, respectively.

The operators used in Mustangs are a combination of the ones used in Lipizzaner and E-GAN. Therefore, as in E-GAN, the loss function of the individuals can be changed. However, the strategy used here is to randomly select one of the three possibilities for the loss function, instead of evaluating the individuals using all losses. The mutation operators used in Lippizaner are also applied for Mustangs. Thus, Mustangs also applies an evolution strategy to update the weights. Crossover is not used in this proposal.

The evaluation follows the same proposal of Lipizzaner. Mustangs uses spatial coevolution to pair discriminators and generators, using a toroidal grid to spatially distribute the individuals. Therefore, individuals are matched using the grid neighborhood to calculate the fitness and evaluate each individual. As in Lipizzaner, the generator is determined as a mixture of generators in this neighborhood.

Mustangs was evaluated with the MNIST and the CelebA datasets. As the architectures of the neural networks that compose a GAN are fixed and predefined, the authors chose different topologies according to the dataset used in the experiments. A four-layers MLP network with 700 neurons and a DCGAN-based architecture were used for the experiments with the MNIST and the CelebA dataset, respectively. For MNIST, a grid size of $3x3$ was used with a time limit of 9 hours. For CelebA, the experiments were executed with a 2×2 grid for 20 epochs. A comparison between standard GAN, E-GAN, Lipizzaner, and Mustangs was presented. The authors found that Mustang is able to generate the best results in respect of the FID score. They also concluded that spatial coevolution is an efficient way to model the population of generators and discriminators to train GANs.

4.5 COEGAN

Coevolutionary Generative Adversarial Networks (COEGAN)[6], a proposal combining neuroevolution and coevolution to train and evolve GANs, was presented by us in [9, 10]. This approach took inspiration on DeepNEAT [27], adapting and extending the model to the context of GANs.

An array of genes compose the genome of COEGAN. The genotype-phenotype mapping transforms this array into a sequence of layers in a deep neural network. Each gene represents either a linear, convolution or transpose convolution layer (also

[6] Code available at https://github.com/vfcosta/coegan.

known as deconvolution layer). Moreover, each gene also has some common internal parameters, such as the activation function, chosen from the following set: ReLU, Leaky ReLU, ELU, Sigmoid, and Tanh. The genes representing a convolution or transpose convolution layer only have the number of output channels as a variable parameter. The number of input channels is calculated dynamically, based on the setup of the previous layer. Similarly, the linear layer only has the number of output features as a variable parameter. The previous layer is also used to calculate the number of input features. Thus, the parameters subject to variation operations are the activation function, the number of output features, and the number of output channels.

Figures 5 and 6 are examples of the genotypes of a discriminator and a generator, respectively. The genotype of the discriminator is composed of a convolutional section and followed by a linear section (composed of fully connected layers). As in the original GAN model, the discriminator outputs the probability that the input sample is a real sample, drawn from the dataset. Similarly, the genotype of the generator is composed of a linear section and followed by a transpose convolutional section (also known as convolutional section). The generator outputs a fake sample, with the same characteristics (i.e., dimension and channels) of a real sample.

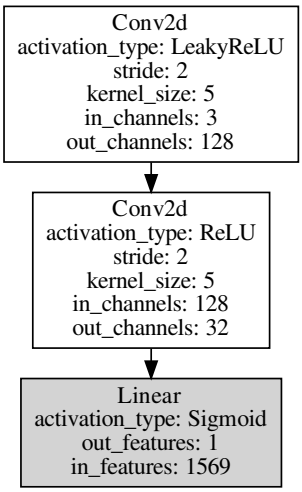

Fig. 5 A genotype of a discriminator. In this case, the genotype is composed of three genes: two convolutional and one linear gene. The phenotype transformation creates a network with three layers in the same linear sequence as displayed in the genome. The output layer is represented by the linear gene and outputs the probability of the input sample to be real or fake.

Competitive coevolution was used to model the algorithm. Therefore, COEGAN is composed of two separated subpopulations: a population of generators, where each G_i represents a generator; and a population of discriminators where each D_j represents a discriminator. A speciation mechanism, inspired by the strategy used in NEAT, was used in each subpopulation to promote innovation. The speciation mechanism ensures that recently modified individuals will have the chance to survive for enough generations to be as powerful as individuals from previous generations. For this, each population is divided into species based on a similarity function (used

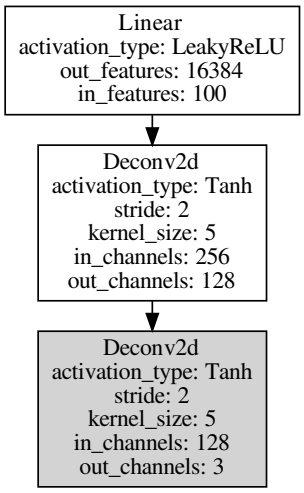

Fig. 6 A genotype of a generator. The genotype is composed of three genes: a linear gene and two deconvolution (also known as transpose convolution) genes. In general, the genotype transformation follows the same structure as in the discriminator. However, in this case, the output layer is a deconvolution layer that outputs the samples created by the generator.

to group similar individuals). Thus, the innovation, represented by the addition of new genes into a genome, may cause the creation of new species in order to fit the individuals containing these new genes. The individuals belonging to new species will have a higher chance to survive because they will not directly compete with more powerful individuals from other species.

COEGAN is only interested in the evolution of the neural network architectures. Thus, the parameters of the layers in the phenotype (e.g., weights and bias) are not part of the evolution, being modified by the training with a gradient descent method. The variation operators are focused on the evolution of the network topology.

Different fitness functions for the generator and the discriminator were used in COEGAN. For discriminators, the fitness is based on the loss function of the original GAN model, i.e., the fitness is equivalent to Eq. 1 (Sect. 2). The same approach was tested on the generator using Eq. 2 (Sect. 2), but preliminary results presented instabilities when using this strategy, making it not suitable to be used as fitness. Thus, the generator uses the FID score [18] as fitness, i.e., the fitness is represented by Eq. 4 (Sect. 2.3). FID is the state-of-the-art metric to compare GANs and outperforms other metrics, such as the Inception Score [38]. The use of the FID score as fitness puts selection pressure in COEGAN and directs the evolution of the population towards the creation of better generators.

Only mutation is used as the variation operator for COEGAN. The mutation process is composed of three operations: add a new layer, remove a layer, and change an existing layer. In the addition operation, a new layer is randomly selected from the set of possible layers (linear or convolution for discriminators and linear or transpose convolution for generators). The remove operation randomly selects an existing layer and excludes it from the genotype, adjusting the connections between the previous and the next layers. The change operation acts on the activation function and the specific attributes of a layer. The activation function is randomly drawn from the set

of possibilities. The number of output features and the number of output channels can be mutated for the linear and convolution layers, respectively. These attributes are mutated using a uniform distribution with a predefined range to limit the possible values. Crossover was also experimented and evaluated in preliminary experiments but it was discarded as it promotes instability, decreasing the performance of the system.

COEGAN keeps the parameters (weights and bias) of the genes involved in a mutation operator when possible. So, the new individual will carry the information from previous generations and the training continues from the last state, simulating the transfer learning mechanism used in deep neural networks. However, in some cases these parameters cannot be kept, such as when the change occurs in the parameters of a linear or a convolution layer. In these cases, the new setup of the layer is incompatible with the previous configuration, and the new layer will be trained from the beginning.

In the evaluation step of the EA, individuals from the populations of discriminators and generators must be paired to be trained and to calculate the fitness for the individuals. The pairing strategy is crucial to coevolution, and some challenges can be related to the issues occurred in the GAN training (see Sect. 3.1). Two pairing strategies were used to evaluate COEGAN: *all vs. all* and *all vs. k-best*.

In *all vs. all*, each discriminator is paired with each generator, resulting in all possible matches. In this case, the fitness for discriminators is the average of the losses obtained by the training with each generator. As the FID score does not use the discriminator in the calculation, the pairing strategy does not affect the fitness for generators. The *all vs. all* strategy is important to promote diversity in the GAN training and improve the variability of the environment for both discriminators and generators. However, the trade-off is the time to execute this approach.

In *all vs. k-best*, k individuals are selected from one population to be matched against all individuals in the other population. Therefore, each generator is paired with k best discriminators from the previous generation and, similarly, each discriminator with k best generators. For the first generation, a random approach is used, i.e., k random individuals are selected for pairing in the initial evaluation. This approach provides less variability in the training but is more efficient, as fewer matches will be executed per generation.

For the selection phase, COEGAN uses a strategy based on NEAT [41]. The populations of generators and discriminators are divided into subpopulations using a speciation strategy based on the used in NEAT. Each species is composed of individuals with similar genomes, i.e., similar network structures. Therefore, the similarity between individuals is based only on the parameters of each gene composing the genome, excluding the weights of the similarity calculation. The distance δ between two genomes i and j is defined as the number of genes that exist only in i or j. The speciation approach uses the distance to cluster individuals based on a δ_t threshold. This threshold is calculated at each generator in order to fit the previously chosen number of species. Fitness sharing is used to adjust the fitness of individuals inside each species. Individuals are selected in proportion to the average fitness of the species they belong to. Besides this process, a tournament between k_t individuals

is applied in each species to finally select the individuals to breed and compose the next population.

To evaluate the COEGAN proposal, experiments using the MNIST and the Fashion MNIST datasets were presented. These experiments compare COEGAN, a DCGAN-based solution, and a random search method using the FID Score, Inception Score, and the root mean square error (RMSE) metrics. The experiments ran for 50 generations, using 10 individuals for the populations of generators and discriminators.

Figure 7 presents the results of the FID score on the MNIST dataset (lower is better). We can see that COEGAN outperforms the other approaches. The random approach presented high variability and worse results in terms of this metric. This is evidence that the choices for the fitness functions for COEGAN provide enough evolutionary pressure to guide the evolution to better outcomes.

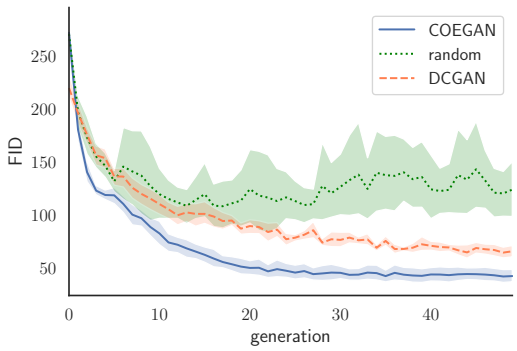

Fig. 7 The FID score on the MNIST dataset comparing COEGAN, the DCGAN-based architecture, and the random search method. Note that, as expected, the random search does not achieve good results and presents high variability on the FID score. The DCGAN-based result shows the convergence of the GAN training. However, COEGAN presents the best results and a smooth decreasing pattern on the FID score.

Figure 8 displays the samples created for the generator after the COEGAN training process. We can see the samples in this figure resembling the data in MNIST. No evidence of the vanishing gradient was found in the experiments with COEGAN, and the mode collapse occurred only partially in some executions. COEGAN avoids these issues by using the evolutionary pressure to discard failed individuals from the population. As these individuals will perform worse than others, they will eventually not be selected, and their issues will not persist through generations. The diversity provided by the population of generators and discriminators is also a factor that prevents these issues from happening. The variability of the training with multiple instances of generators and discriminators, instead of a single generator and discriminator, can be a way to provide a stronger and stable training for GANs.

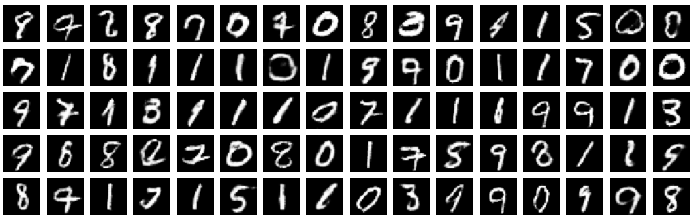

Fig. 8 Samples generated by COEGAN when training on the MNIST dataset.

Table 1 Aspects of the GAN used in the evaluated proposals.

Algorithm	Discriminator	Generator	Architecture	Loss Function
E-GAN	single-fixed	three	DCGAN-based	evolvable[1]
Pareto GAN	many	many	evolvable	evolvable[1]
Lippizaner	many	many	MLP and DCGAN-based[2]	original GAN
Mustangs	many	many	MLP and DCGAN-based[2]	evolvable[1]
COEGAN	many	many	evolvable	original GAN

[1] The loss function is selected using a predefined set of possibilities.
[2] The DCGAN-based architecture was used with the CelebA dataset and a simpler approach with the MNIST dataset (see Sect. 4.3 and Sect 4.4)

5 Discussion

Section 4 presented the current proposals that apply evolutionary algorithms in the context of GANs. We can see that a variety of techniques frequently used in EAs, and introduced here in Sect. 3, were used in these proposals. Following we present and discuss these characteristics regarding the aspects of the GAN model used in the proposals, the choices concerning the EA, and the experimental results.

5.1 Characteristics of the GAN model

Table 1 presents choices with respect to the GAN model used in each proposal. These proposals are compared under the perspective of four attributes: the number of discriminators used in the algorithm, the number of generators, the architecture of each component, and the loss function used to train the GAN.

Except for E-GAN, all proposals used multiple discriminators in their model. For the generators, all proposals used multiple generators, with E-GAN using a fixed number of three generators, corresponding to the number of possible loss functions designed in the algorithm. Thus, E-GAN works with small populations, limiting the evolutionary options that can emerge through generations. On the other hand, Mus-

Table 2 Aspects of the evolutionary algorithm used in the evaluated proposals.

Algorithm	Pairing	Variation Operators	Fitness	Selection
E-GAN	one-vs-three	mutation (loss)	custom	best individual
Pareto GAN	–	crossover and mutation	IGD	Pareto dominance
Lippizaner	spatial coevolution	mutation (weights)	GAN objective	spatial
Mustangs	spatial coevolution	mutation (weights, loss)	GAN objective	spatial
COEGAN	all vs. (all | k-best)[1]	mutation (architecture)	FID and loss	NEAT-based

[1] COEGAN presented experiments using both the *all vs. all* and the *all vs. k-best* approaches.

tangs adapted successfully the E-GAN model in the context of a larger population, using the spatial coevolution approach of Lipizzaner to handle the individuals.

Regarding the architecture, only the Pareto GAN and COEGAN used an evolvable approach. The other proposals used a predefined and fixed architecture for the neural networks of generators and discriminators. Therefore, Pareto GAN and COEGAN work with larger search spaces, as the architectures that can emerge from the EA have a high number of possibilities. They are also potentially able to enhance the balance between generators and discriminators, as the complexity of the architecture is determined by the algorithm.

Lippizaner and COEGAN use a fixed loss function for the GAN training. E-GAN, Pareto GAN, and Mustangs use an evolvable approach to the loss function. This approach uses a set of predefined possibilities to select and attribute a loss function to an individual. A more flexible approach can also be used instead of a predefined set, using genetic programming to discover better loss functions for GANs. However, the proposals analyzed in this chapter did not explore this approach.

5.2 Aspects of the Evolutionary Algorithm

Table 2 presents a comparison between the solutions presented in Sect. 4, focusing on the aspects of the evolutionary algorithm. Four aspects of the EA were analyzed: the pairing approach, the variation operators, the fitness function, and the selection method.

As multiple generators and/or discriminators are used in all proposals and the GAN training occurs using generators and discriminators as adversarial, an approach has to be used to pair the individuals. With the exception of Pareto GAN, all other solutions use separated individuals to represent discriminators and generators. In E-GAN, as there are only a single discriminator and three generators, the policy for pairing is to use the discriminator to evaluate all three generators. Lippizaner and Mustangs use the same spatial coevolution strategy to match generators and discriminators. In COEGAN, the *all vs. all* and *all vs. k-best* were used.

The variation operators are paramount to provide diversity in the search for good solutions in an EA. Pareto GAN uses crossover and mutation as operators. It is

also the solution that provides the most variability regarding the elements that can be evolved through generations in the EA. As Pareto GAN models its individual as a representation of the entire GAN, i.e., encoding both the discriminator and the generator into the genotype, the crossover works exchanging the generator and the discriminator between two parents to form the offspring. The other solutions modeled the GAN with independent genotypes to represent the generator and the discriminator. Therefore, this approach is not applicable to them. COEGAN also evaluated a strategy to apply crossover, using a cut point to share parts of the neural network between parents. However, this strategy proved to be not efficient for the method.

COEGAN and Pareto GAN are the only solutions that have evolvable neural network architectures. The mutation operator is used to provide small changes in these architectures that are built through generations to produce strong discriminators and generators. E-GAN, Lippizaner, and Mustangs use a restricted mutation strategy. In E-GAN, only the loss function can be switched. In Lippizaner, the mutation is applied to the weights to assist in the GAN training. Mustangs combines the operators of E-GAN and Lippizaner. Different from Lippizaner and Mustangs, COEGAN does not apply a mutation operator directly to the weights. However, this option can be explored to develop a hybrid approach that evolves the weight when the gradient descent training stagnates for a number of generations.

The choice for fitness is diverse among the proposals. E-GAN uses a custom function that represents the quality and diversity of the created samples. As only the generator is subject to evolution, the discriminator does not have a fitness associated. Pareto GAN based its fitness on the concepts of the Pareto front, using the inverted generational distance (IGD) to represent the fitness value. Lippizaner and Mustangs use the GAN objective function to calculate the fitness for the individuals. COEGAN follows a distinct approach for the fitness function. The loss function of discriminators of the original GAN model is used as fitness for them. In the generator, the FID score is used as fitness. COEGAN takes advantage of the capabilities in the FID distance to represent the diversity and quality of the created samples. As the FID is commonly used by researchers to compare GANs, the implementation of this metric into an EA is a way to provide automatic insight about the solutions produced by the method.

The selection method used in E-GAN is based on the choice of the best generator. As E-GAN has only three generators, each one with a specific loss function, the fitness guide the evolution by selecting the function that fits the best generator for the current environment. The switches between functions thought generations give to E-GAN the means to provide enough diversity for a convergence. In Pareto GAN, Pareto dominance is used as the strategy to select individuals to form the next generation. Lippizaner and Mustangs have a selection based on the spatial coevolution mechanism used in the evaluation phase. The neighborhood is used to evaluate and replace the individual in the center of a neighborhood according to the fitness. COEGAN uses an approach based on classical NEAT selection. Therefore, speciation is used to ensure that individuals from different species will have the opportunity to develop the skills needed to survive. Some of these strategies can be combined into a single solution to build a stronger algorithm. For example, the

Table 3 Comparison of the experiments presented in the proposals.

Algorithm	Dataset	Population (D x G)	Generations	Metric
E-GAN	CIFAR-10, LSUN, CelebA	1×3	200000	Inception Score
Pareto GAN	bi-objective functions	20[1]	500	IGD
Lippizaner	MNIST, CelebA	4×4	400	–
Mustangs	MNIST, CelebA	4×4, 9×9	time-limited, 20	FID score
COEGAN	MNIST, Fashion MNIST	10×10	50	FID score

[1] In Pareto GAN one individual completely represents a GAN, i.e.,it contains both a generator and a discriminator.

mechanism that guides the selection for Lippizaner and Mustangs can be applied in COEGAN to reduce the complexity of the evaluation phase and bring the advantages given by spatial coevolution.

5.3 Experiments and Results

Table 3 compare the proposals under the perspective of the experimental setup used to assess the contributions of each solution. Four experimental attributes are presented: the dataset used in the training, the number of generators and discriminators in the populations, the number of generations used in training, and the metric used to evaluate the results.

Except for Pareto GAN, all proposals used image datasets in the experiments. Pareto GAN uses bi-objective functions to validate the model, also including a function that simulates the data dimension of the MNIST dataset. In the category of images, MNIST is a simple dataset and should be used carefully to draw generic conclusions about the performance of a solution. The CelebA dataset is perhaps the most commonly used data to validate GANs. Therefore, it would be important to assess the performance of Pareto GAN and COEGAN in this dataset.

The populations used in the experiments vary a lot among the proposals. Except for E-GAN, the solutions used multiple individuals for both populations in the experiments. Although it is possible to use more individuals in E-GAN, the experiments used only a single discriminator and three possibilities for generators (representing each possible loss function). In Pareto GAN, one individual completely represents a GAN. Therefore, 20 individuals were used, meaning that 20 independent GANs with their own generator and discriminator was trained through generations. Lipizzaner and Mustangs use spatial coevolution to distribute the individuals in a grid of 2×2 for the MNIST dataset. For CelebA, Mustangs used a grid of 3×3. As these grids hold a single generator and discriminator in each cell, the population is composed of 4 and 9 individuals for the 2×2 and 3×3 setups, respectively. Spatial coevolution reduces the number of iterations needed to evaluate the individuals. Thus, a larger number of individuals can be used to evaluate Lipizzaner and Mustangs. Besides,

COEGAN can adopt the spatial coevolution approach to reduce the training time and also increase the number of individuals in the experiments.

The number of generations used to evaluate each approach also present high variability. Each approach adapted the experiments to use a number of generations respecting their internal characteristics. For example, as E-GAN works with smaller populations, the number of generations needed to converge is much higher than the others. On the other hand, COEGAN uses only 50 generations on the experiments. As COEGAN uses a population of 10 individuals for generators and discriminators with the *all vs. all* pairing approach, each individual will execute the training process for ten times at each generation.

A metric is commonly used to evaluate the samples created by the generator. COEGAN and Mustangs use the FID score to report and analyze the results. As discussed in Sect. 2.3, the FID score is currently the state-of-the-art metric used to evaluate and compare GANs. The Inception Score, the former most used metric for GANs, was applied in the E-GAN experiments. Pareto GAN adopted the IGD as the metric, that is adequate to its approach that is based on the Pareto set approximations. Lipizzaner analyzed the results through visual inspections and does not present an evaluation with respect to some objective measurement.

As the proposals use different metrics, we can not directly compare the results between all proposals. Only COEGAN and Mustangs share the same metric in the evaluation of the results. The average FID reported for COEGAN [10] and Mustangs [46] are 49.2 and 42.235, respectively. Further experiments for COEGAN [9] achieved an average of 42.6 for the FID score. However, the difference between the average FID scores of COEGAN and Mustangs is small and experiments with equal conditions should be made to better compare these solutions.

6 Conclusions

We present in this chapter the state-of-the-art of evolutionary algorithms applied to Generative Adversarial Networks (GANs). An overview of GANs introduces the challenges of the training method and how the common problems affect the resulting performance. We also explore the applicability of concepts related to evolutionary computation in the context of GANs, showing components that can be evolved and participate actively in an EA. These concepts are materialized into the state-of-the-art proposals of EAs applied to GANs that can be found in the literature. We discuss the characteristics of these proposals, demonstrating the drawbacks and possible improvements for further research.

Despite the recent advances in GANs, it is possible to see that there are still open problems. The stability of training remains a challenge, being tackled by researches using different approaches, such as the proposal of new loss functions and/or alternative architectures. GAN is a relatively new model, and the use of EAs in this context is in its early years. With the rise of the computational power and new methods to apply EAs with robust machine learning techniques (e.g., deep learning),

EAs can be viewed as a strong way to train and evolve GANs. In this way, the proposals presented in this chapter showed advantages in the union between EAs and GANs. A set of different techniques was used by them, with different choices concerning the GAN model and the EA. The diversity of strategies present in GANs and also in evolutionary computation compose a large number of open possibilities for exploration.

As future work, the techniques used in the proposals presented in this chapter can be combined in the development of new solutions. For example, the spatial coevolution strategy used in Mustangs and Lipizzaner can be adapted to the other proposals. On the other hand, the neuroevolution techniques used in Pareto GAN and COEGAN can also be evaluated in the other solutions. Besides, the proposed solutions can be explored in larger experiments. The algorithms can run on a larger number of generations and, when possible, with a larger population of generators and discriminators. These experiments can make possible to evaluate the quality of the outcome and also the scalability of the proposals. Complex datasets can also be used to assess the robustness of the proposed solutions. Different techniques related to GANs can also be incorporated into the algorithm. For example, the use of alternative loss functions (as in WGAN [3]), spectral normalization [30], or the self-attention module for GANs [51]. Other techniques concerning neural networks can also be experimented, such as the recently proposed competitive gradient descent algorithm [39]. Alternative fitness functions can also be investigated to better guide the progress of GANs in an EA. For example, the skill rating metric [32] uses the mechanism that classifies the skill of players in a game to quantify the performance of generators and discriminators in GANs. The adversarial characteristics of GANs and a competitive coevolution environment can leverage the advantage with the use of this metric, providing an efficient evaluation of individuals in the population of generators and discriminators.

References

1. Al-Dujaili, A., Schmiedlechner, T., Hemberg, E., O'Reilly, U.M.: Towards distributed coevolutionary gans. In: AAAI 2018 Fall Symposium (2018)
2. Antonio, L.M., Coello, C.A.C.: Coevolutionary multiobjective evolutionary algorithms: Survey of the state-of-the-art. IEEE Transactions on Evolutionary Computation **22**(6), 851–865 (2018)
3. Arjovsky, M., Chintala, S., Bottou, L.: Wasserstein generative adversarial networks. In: International Conference on Machine Learning, pp. 214–223 (2017)
4. Barratt, S., Sharma, R.: A note on the inception score. arXiv preprint arXiv:1801.01973 (2018)
5. Berthelot, D., Schumm, T., Metz, L.: Began: Boundary equilibrium generative adversarial networks. arXiv preprint arXiv:1703.10717 (2017)
6. Borji, A.: Pros and cons of gan evaluation measures. Computer Vision and Image Understanding **179**, 41–65 (2019)
7. Brock, A., Donahue, J., Simonyan, K.: Large scale GAN training for high fidelity natural image synthesis. In: International Conference on Learning Representations (2019)
8. Coello, C.A.C., Sierra, M.R.: A study of the parallelization of a coevolutionary multi-objective evolutionary algorithm. In: Mexican International Conference on Artificial Intelligence, pp. 688–697. Springer (2004)

9. Costa, V., Lourenço, N., Correia, J., Machado, P.: Coegan: Evaluating the coevolution effect in generative adversarial networks. In: GECCO 2019@ Prague The Genetic and Evolutionary Computation Conference (in press, 2019)
10. Costa, V., Lourenço, N., Machado, P.: Coevolution of generative adversarial networks. In: International Conference on the Applications of Evolutionary Computation (Part of EvoStar), pp. 473–487. Springer (2019)
11. Durugkar, I., Gemp, I., Mahadevan, S.: Generative multi-adversarial networks. arXiv preprint arXiv:1611.01673 (2016)
12. Elgammal, A., Liu, B., Elhoseiny, M., Mazzone, M.: Can: Creative adversarial networks, generating" art" by learning about styles and deviating from style norms. arXiv preprint arXiv:1706.07068 (2017)
13. Fedus, W., Rosca, M., Lakshminarayanan, B., Dai, A.M., Mohamed, S., Goodfellow, I.: Many paths to equilibrium: GANs do not need to decrease a divergence at every step. In: International Conference on Learning Representations (2018)
14. Garciarena, U., Santana, R., Mendiburu, A.: Evolved gans for generating pareto set approximations. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '18, pp. 434–441. ACM, New York, NY, USA (2018)
15. Ghosh, A., Kulharia, V., Namboodiri, V.P., Torr, P.H., Dokania, P.K.: Multi-agent diverse generative adversarial networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 8513–8521 (2018)
16. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: NIPS. Curran Associates, Inc. (2014)
17. Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., Courville, A.C.: Improved training of wasserstein gans. In: Advances in Neural Information Processing Systems, pp. 5769–5779 (2017)
18. Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., Hochreiter, S.: Gans trained by a two time-scale update rule converge to a local nash equilibrium. In: Advances in Neural Information Processing Systems, pp. 6629–6640 (2017)
19. Hillis, W.D.: Co-evolving parasites improve simulated evolution as an optimization procedure. Physica D: Nonlinear Phenomena **42**(1-3), 228–234 (1990)
20. Isola, P., Zhu, J.Y., Zhou, T., Efros, A.A.: Image-to-image translation with conditional adversarial networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1125–1134 (2017)
21. Karras, T., Aila, T., Laine, S., Lehtinen, J.: Progressive growing of GANs for improved quality, stability, and variation. In: International Conference on Learning Representations (2018)
22. Karras, T., Laine, S., Aila, T.: A style-based generator architecture for generative adversarial networks. arXiv preprint arXiv:1812.04948 (2018)
23. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: International Conference on Learning Representations (ICLR) (2015)
24. LeCun, Y.: The mnist database of handwritten digits. http://yann. lecun. com/exdb/mnist/ (1998)
25. Lucic, M., Kurach, K., Michalski, M., Gelly, S., Bousquet, O.: Are gans created equal? a large-scale study. arXiv preprint arXiv:1711.10337 (2017)
26. Mao, X., Li, Q., Xie, H., Lau, R.Y., Wang, Z., Smolley, S.P.: Least squares generative adversarial networks. In: 2017 IEEE International Conference on Computer Vision (ICCV), pp. 2813–2821. IEEE (2017)
27. Miikkulainen, R., Liang, J., Meyerson, E., Rawal, A., Fink, D., Francon, O., Raju, B., Navruzyan, A., Duffy, N., Hodjat, B.: Evolving deep neural networks. arXiv preprint arXiv:1703.00548 (2017)
28. Mirza, M., Osindero, S.: Conditional generative adversarial nets. arXiv preprint arXiv:1411.1784 (2014)
29. Mitchell, M.: Coevolutionary learning with spatially distributed populations. Computational intelligence: principles and practice (2006)
30. Miyato, T., Kataoka, T., Koyama, M., Yoshida, Y.: Spectral normalization for generative adversarial networks. In: International Conference on Learning Representations (2018)

31. Odena, A., Olah, C., Shlens, J.: Conditional image synthesis with auxiliary classifier gans. In: Proceedings of the 34th International Conference on Machine Learning-Volume 70, pp. 2642–2651. JMLR. org (2017)
32. Olsson, C., Bhupatiraju, S., Brown, T., Odena, A., Goodfellow, I.: Skill rating for generative models. arXiv preprint arXiv:1808.04888 (2018)
33. Pan, Z., Yu, W., Yi, X., Khan, A., Yuan, F., Zheng, Y.: Recent progress on generative adversarial networks (gans): A survey. IEEE Access **7**, 36322–36333 (2019)
34. Radford, A., Metz, L., Chintala, S.: Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434 (2015)
35. Rawal, A., Rajagopalan, P., Miikkulainen, R.: Constructing competitive and cooperative agent behavior using coevolution. In: Computational Intelligence and Games (CIG), 2010 IEEE Symposium on, pp. 107–114 (2010)
36. Reed, S., Akata, Z., Yan, X., Logeswaran, L., Schiele, B., Lee, H.: Generative adversarial text to image synthesis. arXiv preprint arXiv:1605.05396 (2016)
37. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al.: Imagenet large scale visual recognition challenge. International Journal of Computer Vision **115**(3), 211–252 (2015)
38. Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., Chen, X.: Improved techniques for training gans. In: Advances in Neural Information Processing Systems, pp. 2234–2242 (2016)
39. Schäfer, F., Anandkumar, A.: Competitive gradient descent. arXiv preprint arXiv:1905.12103 (2019)
40. Sims, K.: Evolving 3d morphology and behavior by competition. Artificial life **1**(4), 353–372 (1994)
41. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evolutionary computation **10**(2), 99–127 (2002)
42. Stanley, K.O., Miikkulainen, R.: Competitive coevolution through evolutionary complexification. Journal of Artificial Intelligence Research **21**, 63–100 (2004)
43. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1–9 (2015)
44. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2818–2826 (2016)
45. Tieleman, T., Hinton, G.: Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural networks for machine learning **4**(2), 26–31 (2012)
46. Toutouh, J., Hemberg, E., O'Reilly, U.M.: Spatial evolutionary generative adversarial networks. arXiv preprint arXiv:1905.12702 (2019)
47. Wang, C., Xu, C., Yao, X., Tao, D.: Evolutionary generative adversarial networks. arXiv preprint arXiv:1803.00657 (2018)
48. Wang, Z., She, Q., Ward, T.E.: Generative Adversarial Networks: A Survey and Taxonomy. arXiv preprint arXiv:1906.01529 (2019)
49. Xu, Q., Huang, G., Yuan, Y., Guo, C., Sun, Y., Wu, F., Weinberger, K.: An empirical study on evaluation metrics of generative adversarial networks. arXiv preprint arXiv:1806.07755 (2018)
50. Yao, X.: Evolving artificial neural networks. Proceedings of the IEEE **87**(9), 1423–1447 (1999)
51. Zhang, H., Goodfellow, I., Metaxas, D., Odena, A.: Self-attention generative adversarial networks. arXiv preprint arXiv:1805.08318 (2018)