Coevolution of Generative Adversarial Networks

Victor Costa(✉), Nuno Lourenço(✉), and Penousal Machado(✉)

CISUC, Department of Informatics Engineering,
University of Coimbra,
Coimbra, Portugal
{vfc,naml,machado}@dei.uc.pt

Abstract. Generative adversarial networks (GAN) became a hot topic, presenting impressive results in the field of computer vision. However, there are still open problems with the GAN model, such as the training stability and the hand-design of architectures. Neuroevolution is a technique that can be used to provide the automatic design of network architectures even in large search spaces as in deep neural networks. Therefore, this project proposes COEGAN, a model that combines neuroevolution and coevolution in the coordination of the GAN training algorithm. The proposal uses the adversarial characteristic between the generator and discriminator components to design an algorithm using coevolution techniques. Our proposal was evaluated in the MNIST dataset. The results suggest the improvement of the training stability and the automatic discovery of efficient network architectures for GANs. Our model also partially solves the mode collapse problem.

Keywords: Neuroevolution · Coevolution · Generative adversarial networks

1 Introduction

Generative adversarial networks (GAN) [1] gained relevance for presenting state-of-the-art results in generative models, mainly in the field of computer vision. A GAN combines two deep neural networks, a discriminator and a generator, in an adversarial training where these networks are confronted in a zero-sum game between them. The generator creates fake samples based on an input distribution. The objective is to deceive the discriminator. On the other hand, the discriminator learns to distinguish between these fake samples and the real input data.

Several works improving the GAN model were recently published, leveraging the quality of the results to impressive levels [2–4]. However, there are still open problems, such as the vanishing gradient and the mode collapse problems, all of them leading to difficulties in the training procedure. Although there are strategies to minimize the effect of those problems, they remain fundamentally unsolved [5,6].

P. Kaufmann and P. A. Castillo (Eds.): EvoApplications 2019, LNCS 11454, pp. 473–487, 2019.
https://doi.org/10.1007/978-3-030-16692-2_32

Another issue, not related only to GANs but also to neural networks in general, is the necessity to define a network architecture previously. In that case, the topology and hyperparameters are usually chosen empirically, thus spending human time in repetitive tasks such as fine-tuning. However, there are approaches that can automatize the design of neural network architectures.

Neuroevolution is the application of evolutionary algorithms to provide the automatic design of neural networks. In neuroevolution, both the network architecture (e.g., topology, hyperparameters and the optimization method) and the parameters (e.g., weights) used in each neuron can be evolved. NeuroEvolution of Augmenting Topologies (NEAT) [7] is a well-known neuroevolution method that evolves the weights and topologies of neural networks. In further experiments, NEAT was also successfully applied in a coevolution context [8]. Moreover, DeepNEAT [9] was recently proposed to expand NEAT to larger search spaces, such as in deep neural networks.

Therefore, this project proposes a new model, called coevolutionary generative adversarial networks (COEGAN), to combine neuroevolution and coevolution in the coordination of the GAN training algorithm. The evolutionary algorithm is based on DeepNEAT. We extended and adapted this model to work on the context of GANs, making use of the competitive characteristic between the generator and discriminator to apply a coevolution model. Hence, each subpopulation of generators and discriminators evolve following its own evolutionary path. To validate our model, experiments were conducted using MNIST [10] as the input dataset for the discriminator component. The results are the improvement of the training stability and the automatic discovery of efficient network topologies for GANs.

The remainder of this paper is organized as follows: Sect. 2 introduces the concepts of GANs and evolutionary algorithms; Sect. 3 presents our approach used to evolve GANs; Sect. 4 displays the experimental results using this approach; finally, Sect. 5 presents conclusions and proposals for further enhancements.

2 Background and Related Works

This section reviews the concepts employed in this paper and presents works related to the proposed model.

2.1 Evolutionary Algorithms

An evolutionary algorithm (EA) is a method inspired by biological evolution that aims to mimic the same evolutionary mechanism found in nature. In EAs, the population is composed of individuals that represent possible solutions for a given problem, using a high-order abstraction to encode their characteristics [11]. The algorithm works by applying variation operators (e.g., mutation and crossover) to the population in order to search for better solutions.

Neuroevolution. Neuroevolution is the application of evolutionary algorithms in the evolution of neural networks. This approach can be applied to weights, topology and hyperparameters of a neural network. When used to generate a network topology, a substantial benefit is the automation of the architecture design and its parameters [7]. Besides, not only the final architecture is important, but the intermediary models also give their contributions to the final model in form of the transference of their trained weights kept through generations [7]. This automation is even more important with the rise of deep learning, which produces larger models and increases the search space [9,12]. However, large search spaces are also a challenge for neuroevolution. These methods have a high computational complexity that may turn their application unfeasible.

Coevolution. The simultaneous evolution of at least two distinct species is called coevolution [13,14]. 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 are designed to reward this cooperation [15–17]. In competitive coevolution, individuals of different species are competing between them. Consequently, their fitness function directly represents this competition in a way that scores between species are inversely related [8,11,14].

2.2 Generative Adversarial Networks

Generative Adversarial Networks (GAN), proposed in [1], is an adversarial model that became relevant for the performance achieved in generative tasks. A GAN combines two deep neural networks: a discriminator D and a generator G. The generator G receives a noise as input and outputs a fake sample, attempting to capture the data distribution used as input for D. The discriminator D receives the real data and fake samples as input, learning to distinguish between them. These components are trained simultaneously as adversaries, creating strong generative and discriminative components.

The loss function for the discriminator is defined by:

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

The loss function for the generator (non-saturating version proposed in [1]) 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 Eqs. 1 and 2, z, p_z, G and D represent the noise sample (used as input to the generator), the noise distribution, the generator and the discriminator, respectively.

Besides, several variations of the loss function were proposed to improve the GAN model, such as in WGAN [2] and LSGAN [4]. These variations were studied in [18] in order to access the superiority in respect to the original GAN proposal. The study founds no empirical evidence that these variations are superior to the original GAN mode.

There are two common problems regarding training stability in GANs: vanishing gradient and mode collapse. The vanishing gradient occurs when the discriminator D became perfect and do not commit mistakes anymore. Hence, the loss function is zeroed, the gradient does not flow through the neural network of the generator, and the GAN progress stagnates. In mode collapse, the generator captures only a small portion of the dataset distribution provided as input to the discriminator. This is not desirable once we want to reproduce the whole distribution of the data. Recently, several approaches tried to minimize those problems, but they remain unsolved [5,6].

There are other models extending the original GAN proposal that modify not only the loss function but also aspects of the architecture. The method described by [3] uses a simple strategy to evolve 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 proceeds, increasing the resolution of images at each phase. However, these layers added progressively are preconfigured, i.e., they are hand-designed and not produced by a stochastic procedure. Thus, the model evolves in a preconfigured way during the training procedure, but the method used in [3] do not use an evolutionary algorithm in this process. Therefore, we can consider this predefined progression as a first step towards the evolution of generative adversarial models.

A very recent model proposes the use of evolutionary algorithms in GANs [19]. Their approach used a simple model for the evolution, using a mutation operator that can change only the loss function of the individuals. Our proposal differs from them by modeling the GAN as a coevolution problem. Besides, in our case, the evolution occurs in the network architecture, and the loss function is the same during the whole method. Nevertheless, a further proposal can incorporate those ideas to evaluate the benefits.

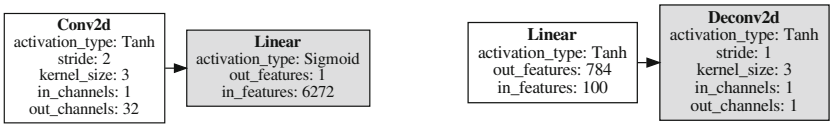
3 Coevolution of Generative Adversarial Networks

We propose a new model called coevolutionary generative adversarial networks (COEGAN). This model combines neuroevolution and coevolution in the coordination of the GAN training algorithm. Our approach is based on DeepNEAT [9], that was extended and adapted to the context of GANs.

In COEGAN, the genome is represented as an array of genes, which are directly mapped into a phenotype consisting of the sequence of layers in a deep neural network. Each gene represents a linear, convolution or transpose convolution layer. Moreover, each gene also has an activation function, chosen from the following set: ReLU, LeakyReLU, ELU, Sigmoid and Tanh. From the specific

parameters of each type of gene, convolution and transpose convolution layers only have the number of output channels as a random parameter. The stride and kernel size are fixed as 2 and 3, respectively. The number of input channels is calculated dynamically, based on the previous layer. Similarly, the linear layer only has the number of output features as the random parameter. The number of input features is calculated based on the previous layer. Therefore, only the activation function, output features and output channels are subject to the variation operations.

Figures 1(a) and (b) are examples of a discriminator and a generator genotype, respectively. The discriminator genotype is composed of a convolutional section and followed by a linear section (fully connected layers). As in the original GAN approach, the output of discriminators is the probability of the input sample be a real sample drawn from the dataset. Similarly, the generator genotype is composed of a linear section and followed by a transpose convolutional section. The output of the generator is a fake sample, with the same characteristics (i.e., dimension and channels) of a real sample.



(a) A genotype of a discriminator (b) A genotype of a generator

Fig. 1. Example of genotypes of a generator and a discriminator.

The overall population is composed of two separated subpopulations: a population of generators, where each G_i represents a generator component in a GAN; and a population of discriminators where each D_j represents a discriminator in a GAN. Furthermore, a speciation mechanism based on the original NEAT proposal is applied to promote innovation in each subpopulation. The speciation mechanism divides the population into species based on a similarity function (used to group similar individuals). Thus, the innovation, represented by the addition of new genes into a genome, causes the creation of new species in order to fit the individuals containing these new genes. Therefore, these new individuals have the chance to survive through generations and reach the performance of older individuals in the population.

The parameters of the layers in the phenotype (e.g., weights and bias) will be trained by the gradient descent method and will not be part of the evolution. The number of parameters to be optimized are too large and evolving them will increase the computational complexity. Therefore, for now, we are interested only in the evolution of the network topology. We plan to develop a hybrid approach that evolves the weight when the gradient descent training stagnates for a number of generations.

3.1 Fitness

For discriminators, the fitness is based on the loss obtained from the regular GAN training method, i.e., the fitness is equivalent to Eq. 1. We have tried to use the same approach for the generator. However, preliminary experiments evidenced that the loss does not represent a good measure for quality in this case. The loss for generators, represented by Eq. 2, is unstable during the GAN training, making it not suitable to be used as fitness in an evolutionary algorithm.

Thus, we selected the Fréchet Inception Distance (FID) [20] as the fitness for generators. FID is the state-of-the-art metric to compare the generative components of GANs and outperforms other metrics, such as the Inception Score [6], with respect to diversity and quality [18]. In FID, an Inception Net [21] (trained on ImageNet [22]) is used to transform images to the feature space (given by a hidden layer of the network). This feature space is interpreted as a continuous multivariate Gaussian. So, the mean and covariance of two Gaussians are estimated using real and fake samples. The Fréchet distance between these Gaussians is given by:

$$FID(x, g) = ||\mu_x - \mu_g||_2^2 + Tr(\Sigma_x + \Sigma_g - 2(\Sigma_x \Sigma_g)^{1/2}). \quad (3)$$

In Eq. 3, μ_x, Σ_x, μ_g, and Σ_g represent the mean and covariance estimated for the real dataset and for fake samples, respectively.

3.2 Variation Operators

Initially, we have used two types of variation operators to breed new individuals: mutation and crossover. The crossover process uses the transition between convolutional layers and linear (fully connected) layers as the cut point. Figure 2 represents an example of this process. However, preliminary tests evidenced that crossover decreases the performance of the system. We expect to conduct further experiments with other crossover variations to assess the contribution of this operator in our model.

The mutation process is composed of three main operations: add a new layer, remove a layer, and change an existing layer. In the addition operation, a new layer is randomly drawn from the set of possible layers. For discriminators, the available layers are linear and convolution. For generators, the available layers are linear and transpose convolution (also called deconvolution). The remove operation chooses an existing layer and excludes it from the genotype.

The change operation modifies the attributes and the activation function of an existing layer. The activation function is randomly chosen from the set of possibilities. Other specific attributes can be changed depending on the type of the layer. The number of output features and the number of output channels are mutated for the linear and convolution layers, respectively. The mutation of these attributes follows a uniform distribution, with a predefined range limiting the possible values.

It is important to note that if the new gene is compatible with its parent, the parameters (weights and bias) are copied. So, the new individual will carry the training information from the previous generation. This simulates the transfer

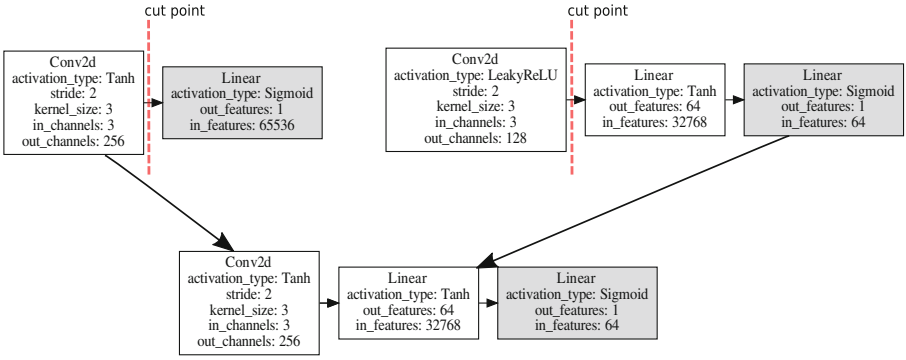


Fig. 2. Example of crossover between discriminators.

learning technique commonly used to train deep neural networks. However, when the attributes of a linear or convolution layer change, the trained parameters are lost. This happens because the setup of weights changes, becoming incompatible with the new layer. Consequently, the weights are not transferred from the parents and the layer will be trained from the beginning.

3.3 Competition Between Generators and Discriminators

In the evaluation step of the evolutionary algorithm, discriminators and generators must be paired to calculate the fitness for each individual in the population. There are several approaches to pair individuals in competitive coevolution, e.g. *all vs. all*, *random* and *all vs. best* [11].

As we want to train the GAN to avoid problems such as the mode collapse and vanishing gradient, we consider the use of the *all vs. best* strategy here. However, we select k individuals rather than only one to promote the diversity in the GAN training. We pair each generator with k best discriminators from the previous generation and, similarly, each discriminator with k best generators. Figure 3 represents an example of this approach with $k = 2$. For the first generation, we assume a random approach, i.e., k random individuals are selected to be paired in the initial evaluation.

The *all vs. all* strategy would also be interesting for our model as it will improve the variability of the environment for both discriminators and generators during the training. However, the trade-off is the time to execute this approach. In *all vs. all*, each discriminator is paired with each generator, resulting in many competitions.

3.4 Selection

For the selection phase, we used a strategy based on NEAT [7]. As in NEAT, we divided the population of generators and discriminators into subpopulations, following a speciation strategy similar to that used in NEAT. Each species contains individuals with similar network structures. For this, we define the

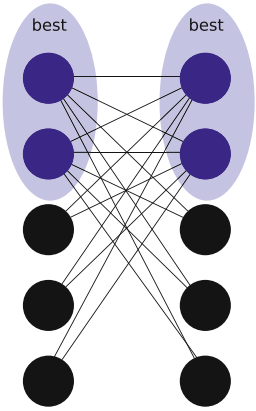


Fig. 3. Representation of the all vs. best competition pattern with $k = 2$.

similarity between individuals based on the parameters of each gene composing the genome. Different of NEAT, we do not use the weights of each layer to differentiate between individuals. Therefore, we calculate the distance δ between two genomes i and j as the number of genes that exist only in i or j. Each species inside the populations of generators and discriminators are clustered based on a threshold δ_t. This threshold is calculated in order to suit the desired number of species. The number of species is a parameter previously chosen.

The selection occurs inside each species. The number of individuals selected inside each species are proportional to the average fitness of the individuals belonging to it. Given the number of individuals to keep in a species, a tournament between k_t individuals is applied to finally select the individuals to breed and compose the next generation.

4 Experiments

In this section, we will evaluate the performance of our method on the MNIST dataset. Normally, the network would be training for several epochs using the whole dataset in the procedure. As this would be an intensive computational task, we will use only a subset of the dataset per generation. This strategy, combined with the transfer of parameters between generations, was sufficient to produce an evolutionary pressure towards efficient solutions and to promote the GAN convergence.

There is no consensus on the metric to represent the quality of samples generated by generative models. However, the Fréchet Inception Distance (FID) was proved to be the best metric when comparing the quality of samples generated by GANs [18]. Therefore, we used the FID score, the same metric used as fitness for generators, to compare our results with the state of the art.

4.1 Experimental Setup

Table 1 describes the parameters used in all experiments reported in this paper.

Table 1. Experimental parameters.

Evolutionary parameters	Value
Number of generations	100
Population size (generators)	20
Population size (discriminators)	20
Crossover rate	0%
Add Layer rate	30%
Remove Layer rate	10%
Change Layer rate	10%
Output features range	[32, 1024]
Output channels range	[16, 128]
k (all vs. best)	3
Tournament k_t	2
FID samples	1000
Genome Limit	4
Species	4
GAN parameters	Value
Batch size	64
Batches per generation	20
Optimizer	RMSProp
Learning rate	0.001

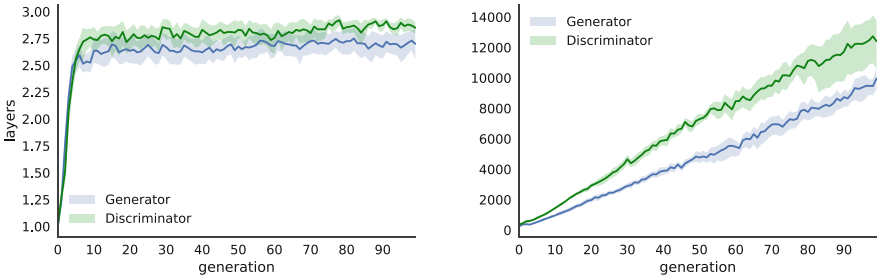
For evolutionary parameters, we chose to execute our experiments for 100 generations. After this number of generations, the fitness stagnates and we expect no improvement of the results. We used 20 individuals for the population of both generators and discriminators. A larger population will probably achieve better results, but the computational cost would be too large. The size of the genome was limited to four layers, also to reduce the computational cost. The number of species used was four, permitting an average of five individuals per species in each subpopulation (generators and discriminators). We empirically defined a probability of 30%, 10% and 10% for the add, remove and change mutations, respectively. As stated before, crossover was not used in the experiments reported in this section.

For the GAN parameters, we choose 64 as batch size, running 20 batches per generation. This amounts to 1280 samples per generation to train discriminators. The optimizer used in the training method was RMSProp [23]. We have also conducted preliminary experiments with Adam [24], but the best results were achieved with RMSProp.

The MNIST dataset was used and we executed each experiment 10 times to achieve the results within a confidence interval of 95%.

4.2 Results

Figure 4 shows the progression of the network through generations. We can see in Fig. 4(a) the average number of layers in the population of generators and discriminators. Because we have limited the genotype to a maximum of four genes, the number of layers rapidly saturates. This is an indication of premature optimization. We can overcome this issue by either increasing the limit or decreasing the growth rate (i.e., reduce the mutation probability). In our tests with crossover activated this problem became even more evident. Figure 4(b) shows the number of genes with the parameters reused in each generation. The linear grow in the amount of reused genes is evidence of the transference mechanism explained in Sect. 3.2. Because we use a strategy similar to transfer learning to keep the trained parameters, this reuse is important to pass the trained weights trough generations.

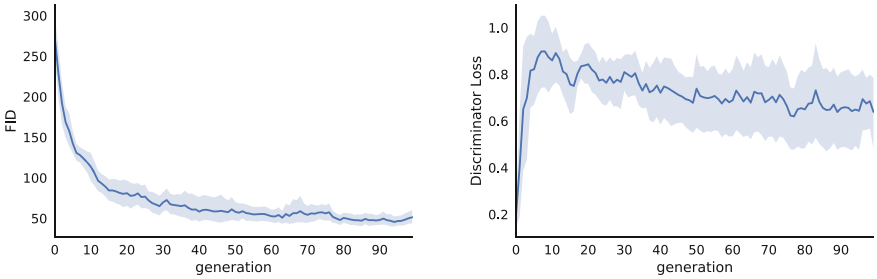


(a) Layers per generation (b) Reused genes

Fig. 4. Progression of layers and the reuse of parameters with a 95% confidence interval

Figure 5 shows the progression of the fitness for best generators and discriminators. In Fig. 5(a), we can see the fitness for generators reducing through generations with reduced noise. Hence, the chosen fitness, i.e., the FID score, is evidenced as suitable to be used in our evolutionary algorithm. For discriminators (Fig. 5(b)), we can see much more noise, which can harm the selection process in the evolutionary algorithm. This suggests that the choice for the discriminator fitness could be improved.

In the final generation, the mean FID was 49.2, with a standard deviation of 10.5. The high standard deviation clearly shows that we need to increase the number of runs in order to get a more representative value for the FID. Besides that, we can see that this score is much worse than state-of-the-art results.



(a) Fitness for generators (b) Fitness for discriminators

Fig. 5. Fitness for discriminators and generators with a 95% confidence interval

For example, the FID for MNIST was reported in [18] as 6.7, with a standard deviation of 0.3. However, our results showed that the model did not collapse into a single point from the input distribution, which is a common problem in GANs.

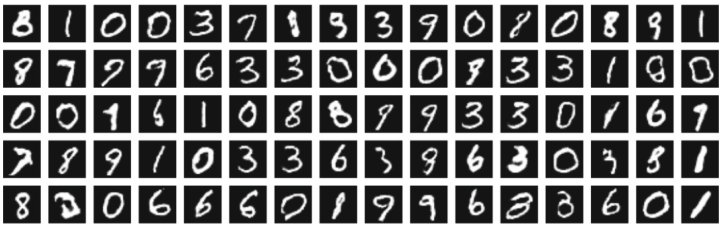
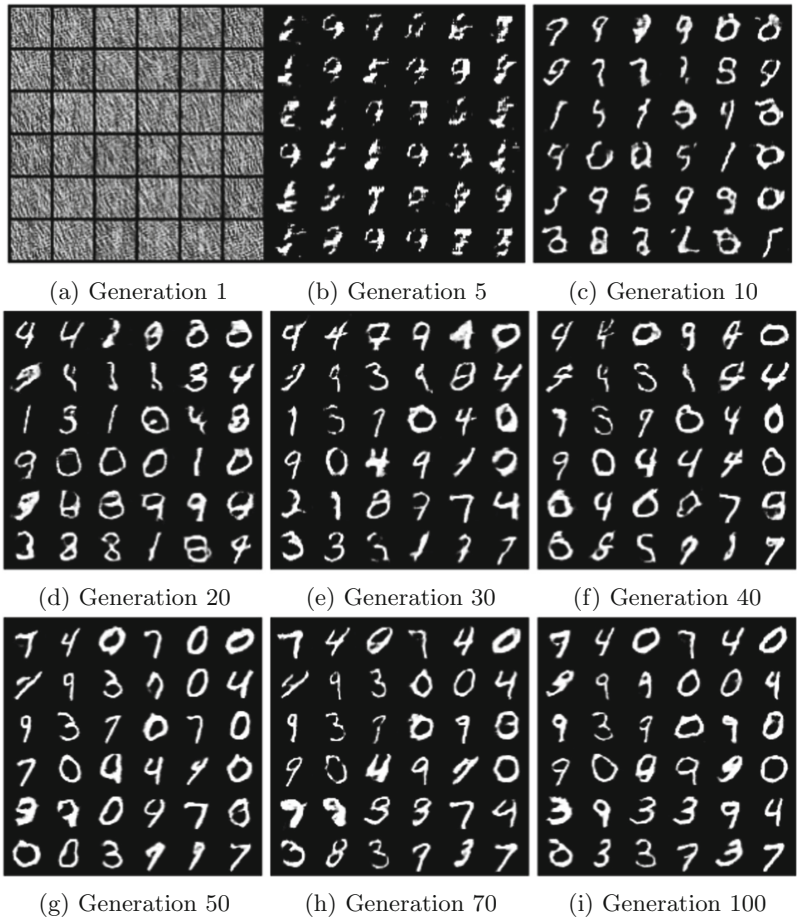


Fig. 6. Samples created by a generator after the evolutionary algorithm.

Figure 6 represents samples generated in one execution, evidencing that our model does not collapse into a single point of the input distribution. We can see this behavior occurring in all executions, which leave us to conclude that our model solves, at least partially, the mode collapse problem. Moreover, all executions reached convergence with equilibrium between the discriminator and the generator, and the vanishing gradient never occurs. This evidences that our proposal brings stability to the training procedure of GANs. Furthermore, our experiments were restricted to a maximum of four layers. A similar number of layers was used by [18]. This evidences that our method does not outperform architectures designed by hand when taking into account only the FID score.

For some executions, the generator captured only a subset of the distribution, which is a form of the mode collapse problem. See in Fig. 6 examples of images created by a generator after the whole evolutionary algorithm. Note that only half of the digits are represented in these samples.



(a) Generation 1 (b) Generation 5 (c) Generation 10

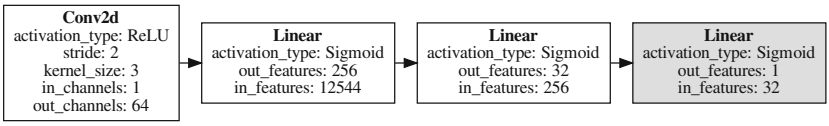
(d) Generation 20 (e) Generation 30 (f) Generation 40

(g) Generation 50 (h) Generation 70 (i) Generation 100

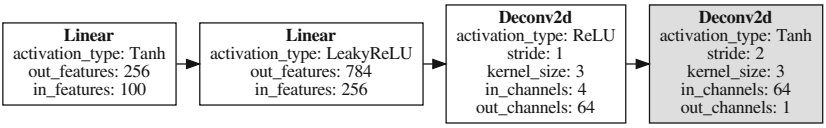
Fig. 7. The progression of samples created by the best generator in generations (a) 1, (b) 5, (c) 10, (d) 20, (e) 30, (f) 40, (g) 50, (h) 70, and (i) 100.

Figure 7 contains generated samples selected to represent the progression of the generator during the evolutionary algorithm. We can see in the first generation only noisy samples, without any structure resembling a digit. After 5 generations (Fig. 7(b)) we can see some structure emerging to form the digits. From the generation 10 onwards we can start to distinguish between the digits, with a progressive improvement of the quality.

Figure 8 presents the network architecture discovered after the last generation. We can see that both components reached the limit of four layers imposed in the experiments. Furthermore, both the generator and the discriminator were composed by a combination of convolutional and linear layers with different activation functions.



(a) Genotype of a discriminator



(b) Genotype of a generator

Fig. 8. Best (a) discriminator and (b) generator found after the final generation.

5 Conclusions

Generative adversarial networks (GAN) achieved important results for generative models in the field of computer vision. However, there are stability problems in the training method, such as the vanishing gradient and the mode collapse problems.

We present in this paper a model that combines neuroevolution and coevolution in the coordination of the GAN training algorithm. To design the model, we took inspiration on previous evolutionary algorithms, such as NEAT [8] and DeepNeat [9], and on recent advances in GANs, such as [3].

We made experiments using the MNIST dataset. In all executions, the system reached an equilibrium and convergence, not falling into the vanishing gradient problem. We also evidenced the FID score as a good fitness metric for generators. We used the loss function as fitness for the population of discriminators. However, the results evidenced that a better metric may be necessary to ensure the proper evolution of this population. The results showed that our model partially solves the mode collapse problem. In our experiments, the generated samples always presented some diversity, partially preserving the characteristics of the input distribution. Besides that, our proposal did not outperform the state-of-the-art results, such as presented [18].

Therefore, as future works, we will experiment other metrics to be used as fitness for the discriminator component, such as area under the curve (AUC). We will also assess that the proposed model does not suffer from cyclic issues commonly seen in coevolution models [25,26]. More experiments should be made with larger populations as well as larger genotype limits. In addition, the model

will also be evaluated with the CelebA dataset [27]. We will make experiments with a reduced growth rate of the networks to avoid premature optimization. Finally, we will conduct ablation studies to assess the individual contribution of each aspect of the proposed algorithm (e.g., speciation, mutation, crossover).

Acknowledgments. This article is based upon work from COST Action CA15140: ImAppNIO, supported by COST (European Cooperation in Science and Technology): www.cost.eu.

References

1. Goodfellow, I., et al.: Generative adversarial nets. In: NIPS (2014)
2. Arjovsky, M., Chintala, S., Bottou, L.: Wasserstein generative adversarial networks. In: International Conference on Machine Learning, pp. 214–223 (2017)
3. 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)
4. 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)
5. 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)
6. 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)
7. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evol. Comput. **10**(2), 99–127 (2002)
8. Stanley, K.O., Miikkulainen, R.: Competitive coevolution through evolutionary complexification. J. Artif. Intell. Res. **21**, 63–100 (2004)
9. Miikkulainen, R., et al.: Evolving deep neural networks. arXiv preprint arXiv:1703.00548 (2017)
10. LeCun, Y.: The MNIST database of handwritten digits (1998). http://yann.lecun.com/exdb/mnist/
11. Sims, K.: Evolving 3D morphology and behavior by competition. Artif. Life **1**(4), 353–372 (1994)
12. Assunção, F., Lourenço, N., Machado, P., Ribeiro, B.: Evolving the topology of large scale deep neural networks. In: Castelli, M., Sekanina, L., Zhang, M., Cagnoni, S., García-Sánchez, P. (eds.) EuroGP 2018. LNCS, vol. 10781, pp. 19–34. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-77553-1_2
13. Hillis, W.D.: Co-evolving parasites improve simulated evolution as an optimization procedure. Physica D **42**(1–3), 228–234 (1990)
14. Rawal, A., Rajagopalan, P., Miikkulainen, R.: Constructing competitive and cooperative agent behavior using coevolution. In: 2010 IEEE Symposium on Computational Intelligence and Games (CIG), pp. 107–114 (2010)
15. García-Pedrajas, N., Hervás-Martínez, C., Muñoz-Pérez, J.: Covnet: a cooperative coevolutionary model for evolving artificial neural networks. IEEE Trans. Neural Netw. **14**(3), 575–596 (2003)

16. García-Pedrajas, N., Hervás-Martínez, C., Ortiz-Boyer, D.: Cooperative coevolution of artificial neural network ensembles for pattern classification. IEEE Trans. Evol. Comput. **9**(3), 271–302 (2005)
17. Gomez, F., Schmidhuber, J., Miikkulainen, R.: Accelerated neural evolution through cooperatively coevolved synapses. J. Mach. Learn. Res. **9**, 937–965 (2008)
18. Lucic, M., Kurach, K., Michalski, M., Gelly, S., Bousquet, O.: Are GANS created equal? a large-scale study. arXiv preprint arXiv:1711.10337 (2017)
19. Wang, C., Xu, C., Yao, X., Tao, D.: Evolutionary generative adversarial networks. arXiv preprint arXiv:1803.00657 (2018)
20. 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)
21. 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)
22. Russakovsky, O., et al.: Imagenet large scale visual recognition challenge. Int. J. Comput. Vis. **115**(3), 211–252 (2015)
23. Tieleman, T., Hinton, G.: Lecture 6.5-rmsprop: divide the gradient by a running average of its recent magnitude. COURSERA: Neural Netw. Mach. Learn. **4**(2), 26–31 (2012)
24. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. In: International Conference on Learning Representations (ICLR) (2015)
25. Ficici, S.G., Pollack, J.B.: A game-theoretic memory mechanism for coevolution. In: Cantú-Paz, E., et al. (eds.) GECCO 2003. LNCS, vol. 2723, pp. 286–297. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-45105-6_35
26. Monroy, G.A., Stanley, K.O., Miikkulainen, R.: Coevolution of neural networks using a layered pareto archive. In: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, pp. 329–336. ACM (2006)
27. Liu, Z., Luo, P., Wang, X., Tang, X.: Deep learning face attributes in the wild. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 3730–3738 (2015)