Analyzing the Components of Distributed Coevolutionary GAN Training

Jamal Toutouh, Erik Hemberg, and Una-May O'Reilly

Massachusetts Institute of Technology, CSAIL, MA, USA
toutouh@mit.edu, {hembergerik,unamay}@csail.mit.edu

Abstract. Distributed coevolutionary Generative Adversarial Network (GAN) training has empirically shown success in overcoming GAN training pathologies. This is mainly due to diversity maintenance in the populations of generators and discriminators during the training process. The method studied here coevolves sub-populations on each cell of a spatial grid organized into overlapping Moore neighborhoods. We investigate the impact on performance of two algorithm components that influence the diversity during coevolution: the performance-based selection/replacement inside each sub-population and the communication through migration of solutions (networks) among overlapping neighborhoods. In experiments on MNIST dataset, we find that the combination of these two components provides the best generative models. In addition, migrating solutions without applying selection in the sub-populations achieves competitive results, while selection without communication between cells reduces performance.

Keywords: Generative adversarial networks · coevolution · diversity · selection pressure · communication.

1 Introduction

Machine learning with Generative Adversarial Networks (GANs) is a powerful method for generative modeling [9]. A GAN consists of two neural networks, a generator and a discriminator, and applies adversarial learning to optimize their parameters. The generator is trained to transform its inputs from a random latent space into "artificial/fake" samples that approximate the true distribution. The discriminator is trained to correctly distinguish the "natural/real" samples from the ones produced by the generator. Formulated as a minmax optimization problem through the definitions of generator and discriminator loss, training can converge on an optimal generator that is able to fool the discriminator.

GANs are difficult to train. The adversarial dynamics introduce convergence pathologies [5,14]. This is mainly because the generator and the discriminator are differentiable networks, their weights are updated by using (variants of) simultaneous gradient-based methods to optimize the minmax objective, that rarely converges to an equilibrium. Thus, different approaches have been proposed to improve the convergence and the robustness in GAN training [7,15,18,20,26].

A promising research line is the application of distributed competitive coevolutionary algorithms (Comp-COEA). Fostering an arm-race of a population of generators against a population of discriminators, these methods optimize the minmax objective of GAN training. Spatially distributed populations (cellular algorithms) are effective at mitigating and resolving the COEAs pathologies attributed to a lack of diversity [19], which are similar to the ones observed in GAN training. `Lipizzaner` [1,22] is a spatial distributed Comp-COEA that locates the individuals of both populations on a spatial grid (each cell contains a GAN). In a cell, each generator is evaluated against all the discriminators of its neighborhood, the same with the discriminator. It uses neighborhood communication to propagate models and foster diversity in the sub-populations. Moreover, the selection pressure helps the convergence in the sub-populations [2].

Here, we evaluate the impact of neighborhood communication and selection pressure on this type of GAN training. We conduct an ablation analysis to evaluate different combinations of these two components. We ask the following research questions: **RQ1:** *What is the effect on the quality of the generators when training with communication or isolation and the presence or absence of selection pressure?*. The quality of the generators is evaluated in terms of the accuracy of the samples generated and their diversity. **RQ2:** *What is the effect on the diversity of the network parameters when training with communication or isolation and the presence or absence of selection pressure?* **RQ3:** *What is the impact on the computational cost of applying migration and selection/replacement?*

The main contributions of this paper are: *i)* proposing distributed Comp-COEA GAN training methods by applying different types of ablation to `Lipizzaner`, *ii)* evaluating the impact of the communication and the selection pressure on the quality of the returned generative model, and *iii)* analyzing their computational cost.

The paper is organized as follows. Section 2 presents related work. Section 3 describes `Lipizzaner` and the ablations the methods analyzed. The experimental setup is in Section 4 and results in Section 5. Finally, conclusions are drawn and future work is outlined in Section 6.

2 Related Work

In 2014, Goodfellow introduced GAN training [9]. Robust GAN training methods are still investigated [5,14]. Competitive results have been provided by several practices that stabilize the training [7]. These methods include different strategies, such as using different cost functions to the generator or discriminator [4,15,18,32,29] and decreasing the learning rate through the iterations [20].

GAN training that involves multiple generators and/or discriminators empirically show robustness. Some examples are: iteratively training and adding new generators with boosting techniques [25]; combining a cascade of GANs [30]; training an array of discriminators on different low-dimensional projections of the data [17]; training the generator against a dynamic ensemble of discriminators [16]; independently optimizing several "local" generator-discriminator pairs so that a "global" supervising pair of networks can be trained against them [6].

Evolutionary algorithms (EAs) may show limited effectiveness in high dimensional problems because of runtime [24]. Parallel/distributed implementations allow EAs to keep computation times at reasonable levels [3,8]. There has been an emergence of large-scale distributed evolutionary machine learning systems. For example: EC-Star [11], which runs on hundreds of desktop machines; a simplified version of Natural Evolution Strategies [31] with a novel communication strategy to address a collection of reinforcement learning benchmark problems [21] or deep convolutional networks trained with genetic algorithms [23].

Theoretical studies and empirical results demonstrate that the spatially distributed COEA GAN training mitigates convergence pathologies [1,22,26]. In this study, we focus on spatially distributed Comp-COEA GAN training, such as `Lipizzaner` [1,22]. `Lipizzaner` places the individuals of the generator and discriminator populations on each cell (i.e., each cell contains a generator-discriminator pair). Overlapping Moore neighborhoods determine the communication among the cells to propagate the models through the grid. Each generator is evaluated against all the discriminators of its neighborhood and the same happens with each discriminator. This intentionally fosters diversity to address GAN training pathologies. Mustangs [26], a `Lipizzaner` variant, uses randomly selected loss functions to train each cell for each epoch to increase diversity. Moreover, training the GANs in each cell with different subsets of the training dataset has been demonstrated effective in increasing diversity across the grid [27]. These three approaches return an ensemble/mixture of generators defined by the best neighborhood (sub-population of generators) built using evolutionary ensemble learning [28]. They have shown competitive results on standard benchmarks.

Here, we investigate the impact of two key components for diversity in Comp-COEA GAN training using ablations.

3 Comp-COEA GAN Training Ablations

Here, we evaluate the impact of communication and selection pressure on the performance of distributed Comp-COEA GAN training. One goal of the training is to maintain diversity as a means of resilience to GAN training pathologies. The belief is that a lack of sufficient diversity results in convergence to pathological training states [22] and too much diversity results in divergence. The use of selection and communication is one way of regulating the diversity.

This section presents `Lipizzaner` [22] and describes the ablations applied to gauge the impact of communication/isolation and selection pressure.

3.1 Lipizzaner Training

`Lipizzaner` adversarially trains a population of generators $\mathbf{g} = \{g_1, ..., g_N\}$ and a population of discriminators $\mathbf{d} = \{d_1, ..., d_N\}$, where N is the size of the populations. It defines a toroidal grid in whose cells a pair generator-discriminator, i.e., a GAN, is placed (called *center*). This allows the definition of neighborhoods with sub-populations $\mathbf{g}^k$ and $\mathbf{d}^k$, of $\mathbf{g}$ and of $\mathbf{d}$, respectively. The size of these sub-populations is denoted by s ($s \leq N$). `Lipizzaner` uses the five-cell Moore neighborhood ($s = 5$), i.e., the neighborhoods include the cell itself (*center*) and the cells in the *West*, *North*, *East*, and *South* (see Figure 1).

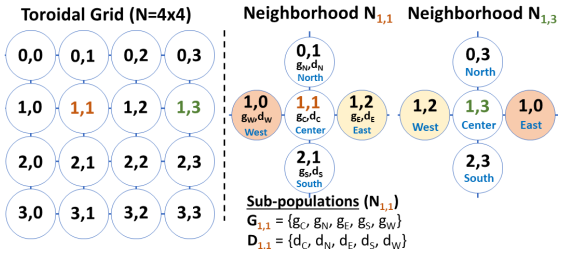


Fig. 1: A 4×4 grid (left) and the neighborhoods $N_{1,1}$ and $N_{1,3}$ (right).

Each cell asynchronously executes in parallel its own learning algorithm. Cells interact with the neighbors at the beginning of each training epoch. This communication is carried out by gathering the latest updated *center* generator and discriminator of its overlapping neighborhoods. Figure 1 illustrates some examples of the overlapping neighborhoods on a *4x4* toroidal grid. The updates in cell $N_{1,0}$ and $N_{1,2}$ will be communicated to the $N_{1,1}$ and $N_{1,3}$ neighborhood.

`Lipizzaner` returns an ensemble of generators that consist of a sub-population of generators $\mathbf{g}^k$ and a mixture weight vector $\mathbf{w} \subset \mathbb{R}^s$, where $w_i \in \mathbf{w}$ represents the probability that a data point is drawn from $\mathbf{g}_i^k$, i.e., $\sum_{w_i \in \mathbf{w}^k} w_i = 1$. Thus, an Evolutionary Strategy, `ES`-(1+1), evolves $\mathbf{w}$ to optimize the weights in order to get the most accurate generative model according to a given metric [22], such as Fréchet Inception Distance (FID) [10].

`Algorithm` 1 summarizes the main steps of `Lipizzaner`. It starts the parallel execution of the training process in each cell. First, the cell randomly initializes single generator and discriminator models. Then, the training process consist of a loop with three main steps: *i)* gathering the GANs (neighbors) to update the neighborhood; *ii)* updating the *center* by applying the training method presented in `Algorithm` 2; and *iii)* evolving mixture of weights by applying `ES`-(1+1) (line 7 in `Algorithm` 1). These three steps are repeated T (training epochs) times. The returned generative model is the best performing neighborhood with its optimal mixture weights, i.e., the best ensemble of generators.

The Comp-COEA training applied to the networks of each sub-population is shown in `Algorithm` 2. The output is the sub-population n with the *center* updated. This method is built on the basis of *fitness evaluation*, *selection and replacement*, and *reproduction* based on gradient descent updates of the weights.

The *fitness* of each network is evaluated in terms of the average *loss function* $\mathcal{L}$ when it is evaluated against all the adversaries. After evaluating all the networks, a *tournament selection operator* is applied to select the parents (a generator and a discriminator) to generate the offspring (lines from 1 to 5).

The selected generator/discriminator is evaluated against a randomly chosen adversary from the sub-population (lines 8 and 11, respectively). The computed losses are used to mutate the parents (i.e., update the networks' parameters) according to the stochastic gradient descent (SGD), which learning rate values n_δ were updated applying Gaussian-based mutation (line 7).

When the training is completed, all models are evaluated again, the least fit generator and discriminator in the sub-populations are replaced with the fittest ones and sets them as the *center* of the cell (lines from 14 to 20).

Algorithm 1 `Lipizzaner`
Input: T: Training epochs, E: Grid cells, s: Neighborhood size, θ_{EA}: `mixtureEvolution` parameters, θ_{Train}: Parameters for training the models in each cell
Return: g: Neighborhood of generators, $\mathbf{w}$: Mixture weights

parfor $c \in E$ **do** ▷ Asynchronous parallel execution of all cells in grid
  $n, \mathbf{w} \leftarrow$ initializeNeighborhoodAndMixtureWeights(c, s)
  for epoch **do** $\in [1, \ldots, \mathrm{T}]$ ▷ Iterate over training epochs
    $n \leftarrow$ copyNeighbours(c, s) ▷ Gather neighbour networks
    $n \leftarrow$ trainModels(n, θ_{Train}) ▷ Update GANs weights
    $g \leftarrow$ getNeighborhoodGenerators(n) ▷ Get the generators
    $\mathbf{w} \leftarrow$ `mixtureEvolution` $(\mathbf{w}, g, \theta_{EA})$ ▷ Evolve mixture weights by `ES`-(1+1)
end parfor
$(g, \mathbf{w}) \leftarrow$ `bestEnsemble`$(g^*, \mathbf{w}^*)$ ▷ Get the best ensemble
return $(g, \mathbf{w})$ ▷ Cell with best generator mixture

Algorithm 2 `Coevolve and train the networks`
Input: τ: Tournament size, X: Input training dataset, β: Mutation probability, n: Cell neighborhood sub-population, M: Loss functions
Return: n: Cell neighborhood sub-population updated

$\mathbf{B} \leftarrow$ getMiniBatches(X) ▷ Load minibatches
$B \leftarrow$ getRandomMiniBatch$(\mathbf{B})$ ▷ Get a minibatch to evaluate GANs
for $g, d \in \mathbf{g} \times \mathbf{d}$ **do** ▷ Evaluate all GAN pairs
  $\mathcal{L}_{g,d} \leftarrow$ evaluate(g, d, B) ▷ Evaluate GAN
$g, d \leftarrow$ select(n, τ) ▷ Select with min loss($\mathcal{L}$) as fitness
for $B \in \mathbf{B}$ **do** ▷ Loop over batches
  $n_\delta \leftarrow$ mutateLearningRate(n_δ, β) ▷ Update learning rate
  $d' \leftarrow$ getRandomOpponent$(\mathbf{d})$ ▷ Get random discriminator to train g
  $\nabla_g \leftarrow$ computeGradient(g, d') ▷ Compute gradient for g against d'
  $g \leftarrow$ updateNN(g, ∇_g, B) ▷ Update g with gradient
  $g' \leftarrow$ getRandomOpponent$(\mathbf{g})$ ▷ Get uniform random generator to train g
  $\nabla_d \leftarrow$ computeGradient(d, g') ▷ Compute gradient for d against g'
  $d \leftarrow$ updateNN(d, ∇_d, B) ▷ Update d with gradient
for $g, d \in \mathbf{g} \times \mathbf{d}$ **do** ▷ Evaluate all updated GAN pairs
  $\mathcal{L}_{g,d} \leftarrow$ evaluate(g, d, B) ▷ Evaluate GAN
$\mathcal{L}_g \leftarrow average(\mathcal{L}_{\cdot,d})$ ▷ Fitness of g is the average loss value ($\mathcal{L}$)
$\mathcal{L}_d \leftarrow average(\mathcal{L}_{g,\cdot})$ ▷ Fitness of d is the average loss value ($\mathcal{L}$)
$n \leftarrow$ replace$(n, \mathbf{g})$ ▷ Replace the generator with worst loss
$n \leftarrow$ replace$(n, \mathbf{d})$ ▷ Replace the discriminator worst loss
$n \leftarrow$ setCenterIndividuals(n) ▷ Best g and d are placed in the center
return n

3.2 Lipizzaner Ablations Analyzed

We conduct an ablation analysis of `Lipizzaner` to evaluate the impact of different degrees of communication/isolation and selection pressure on its COEA GAN training. The ablations are listed in Table 1. They ablate the use of *sub-populations*, the *communication* between cells, and the application of the *selection/replacement* operator. Thus, we define three variations of `Lipizzaner`:

- Spatial Parallel GAN training (`SPaGAN`): It does not apply selection/replacement. After gathering the networks from the neighborhood, it uses them to only train the *center* (i.e., `SPaGAN` does not apply the operations in lines between 2 and 5 and lines between 14 and 20 of `Algorithm` 2).
- Isolated Coevolutionary GAN training (`IsoCoGAN`): It trains sub-populations of GANs without communication between cells. There is no exchange of networks between neighbors after the creation of the initial sub-population (i.e., the operation in line 4 of `Algorithm` 1 is applied only during the first iteration of its main loop).
- Parallel GAN (`PaGAN`): This trains a population of N GANs in parallel. When all the GAN training is finished, it randomly produces N sub-sets of $s \leq N$ generators selected from the entire population of trained generators to define the ensembles and optimizes the mixture weights with ES-(1+1).

Table 1: Key components of the GAN training methods.

Feature	Lipizzaner	SPaGAN	IsoCoGAN	PaGAN
Use of sub-populations	✓	✓	✓	-
Communication between sub-populations	✓	✓	-	-
Application of selection/replacement operator	✓	-	✓	-

4 Experimental Setup

This experimental analysis compares `Lipizzaner`, `SPaGAN`, `IsoCoGAN`, and `PaGAN` in creating generative models to produce samples of the MNIST dataset [12]. This dataset is widely used as a benchmark due to its target space and dimensionality.

The communication among the neighborhoods is affected by the grid size [22]. We evaluate `SPaGAN` and `Lipizzaner` by using three different grid sizes: 3×3, 4×4, and 5×5, to control for the impact of this parameter.

To evaluate the quality of the generated data we use **FID** score. The **network diversity** of the trained models is evaluated by the L_2 **distance between the parameters** of the generators. The **total variation distance (TVD)** [13], which is a scalar that measures class balance, is used to analyze the diversity of the generated data by the generative models. TVD reports the difference between the proportion of the generated samples of a given digit and the ideal proportion (10% in MNIST). Finally, the **computational cost** is measured in terms of run time. All the analyzed methods apply the same number of training-network steps, i.e., updating the networks' parameters according to SGD. All implementations are publicly available[1] and use the same `Python` libraries and versions. The distribution of the results is not Gaussian, so a Mann-Whitney U statistical test is applied to evaluate their significance.

[1] `Lipizzaner` and ablations - `https://github.com/ALFA-group/lipizzaner-gan`

The methods evaluated here are configured according to the settings proposed by the authors of `Lipizzaner` [22]. The experimental analysis is performed on a cloud computing platform that provides 8 Intel Xeon cores 2.2GHz with 32 GB RAM and an NVIDIA Tesla P100 GPU with 16 GB RAM.

5 Results and Discussion

This section discusses the main results of the experiments carried out to evaluate `Lipizzaner`, `SPaGAN`, `IsoCoGAN`, and `PaGAN`.

5.1 Generator Quality

Table 2 summarizes the results by showing the best FID scores for the different methods and grid sizes in the 30 independent runs. `Lipizzaner` obtains the lowest/best **Mean**, **Median**, and **Min** FID scores for all grid sizes. `SPaGAN` and `PaGAN` are the second and third best methods, respectively. `IsoCoGAN` presents the lowest quality by showing the highest (worst) FID scores. The FID values in terms of hundreds indicate the generators are not capable of creating adequate MNIST data samples. The Mann-Whitney U test shows that the methods that exchange individuals among the neighborhoods, i.e., `Lipizzaner` and `SPaGAN`, are significantly better than `PaGAN` and `IsoCoGAN` ($\alpha << 0.001$). These results are confirmed by posthoc statistical analysis. According to this analysis there is no statistical difference between `Lipizzaner` 3×3 and `SPaGAN` for all evaluated grid sizes. In turn, `Lipizzaner` 4×4 and 5×5 outperform all the other methods and `Lipizzaner` 5×5 provides the best generators (lowest FID).

Table 2: FID results (Low FID indicates more quality)

Grid	Method	Mean±Std	Median	Iqr	Min	Max
3×3	`Lipizzaner`	**40.93±8.51**	**39.44**	10.59	**28.04**	62.10
	`SPaGAN`	43.59±5.53	43.09	5.94	30.65	**51.81**
	`IsoCoGAN`	881.79±52.67	871.04	66.81	798.03	998.79
	`PaGAN`	51.15±14.06	46.85	6.40	40.81	112.58
4×4	`Lipizzaner`	**32.84±6.93**	**32.22**	7.23	**19.15**	**46.53**
	`SPaGAN`	37.97±8.89	35.98	13.61	23.69	56.36
5×5	`Lipizzaner`	**28.74±4.91**	**28.14**	7.45	**22.56**	**40.57**
	`SPaGAN`	39.11±4.00	39.61	5.29	27.39	48.03

`IsoCoGAN` does not converge since the individuals of one sub-population are evaluated against randomly chosen individuals of the other one. As the accuracy of their fitness evaluation depends subjectively on the quality of the randomly chosen opponent, it is likely that the fitness value does not correspond to the real quality of the individual, and therefore, the selection/replacement operator does not promote the objectively best solution in the sub-population.

For all grid sizes, `SPaGAN` provides higher/worse FIDs than `Lipizzaner`. `SPaGAN` has a similar quality for all grid sizes, but `Lipizzaner` improves when the grid size is increased. Thus, the larger the grid size, the greater the difference between these two methods. We observe the benefits of increasing diversity (population/grid size) when applying the coevolutionary approach with selection and replacement of `Lipizzaner`. The results of `Lipizzaner` are mainly due to,

first, the larger population sizes' ability to encompass a higher diversity, and second, the selection/replacement process applied by `Lipizzaner` accelerates the convergence of the population to higher quality generators.

5.2 Quality (FID) Evolution

Fig. 2 shows the evolution of the median FID when using `PaGAN`, `SPaGAN`, and `Lipizzaner` in a 3×3 grid. According to this figure, `Lipizzaner` improves the performance of the generators faster than `SPaGAN`. After the first 75 to 100 training epochs, the FID does not show such a reduction in `Lipizzaner`, but `SPaGAN` is able to keep reducing it until the end of the training process. The faster convergence of `Lipizzaner` is also illustrated by Fig. 4 that shows the FID scores in the grid of a given independent run at the epoch number 25, 50, 75, and 100. This is mainly due to the capacity of exploitation of this method when using selection/replacement. `PaGAN` shows a fast FID reduction at the beginning of the training process. But, the FID sharply oscillates during the first 75 iterations and it converges to worse FID scores than `Lipizzaner` and `SPaGAN`.

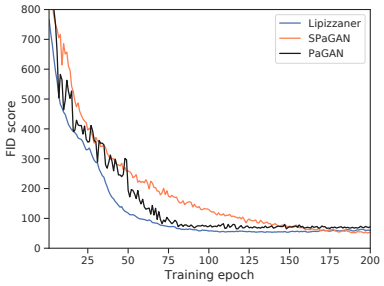


Fig. 2: Median FID evolution through the 200 epochs (3×3).

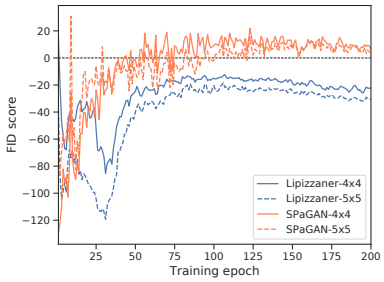


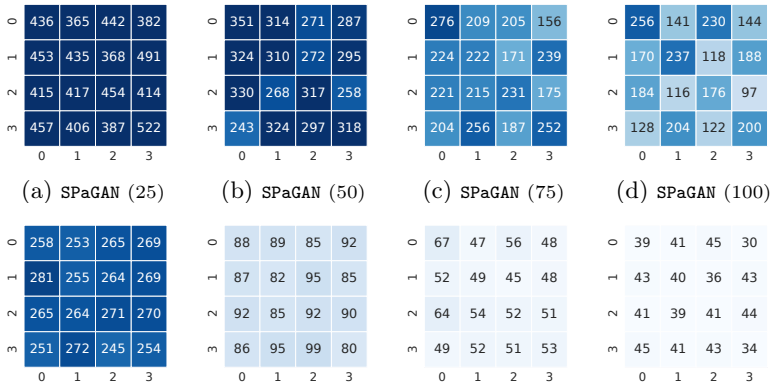
Fig. 3: FID differences when using differnt grids ($diff\_FID_i^{m\times m}$).

Fig. 3 illustrates the differences between the FID score when using 3×3 and 4×4 or 3×3 and 5×5 for the same method. This difference for the grid size m×m at the i epoch is computed as $diff\_FID_i^{m\times m} = FID_i^{3\times 3} - FID_i^{m\times m}$. This figure allows evaluating the impact on the FID evolution when using different grid (population) sizes. It shows how `Lipizzaner` provides lower FIDs and it is able to converge faster as the grid size increases during the first iterations. In contrast, `SPaGAN` gets better FIDs when increasing the grid size only during the first 50 epochs. `Lipizzaner` is able to take advantage of the diversity generated when the grid size increase to converge faster to lower FIDs than `SPaGAN`. Thus, in terms of convergence, selection/replacement helps the convergence of the coevolutionary training method when exchanging solutions with the neighborhoods.

5.3 Generator Output Diversity

The output diversity reports the class distribution of the fake data produced by a generative model. Table 3 summarizes the results in terms of TVD.

For the 3×3 grid experiments, `IsoCoGAN` shows the worst (highest) TVD values. `PaGAN` provides good TVDs but it is statistically less competitive than



(a) SPaGAN (25) (b) SPaGAN (50) (c) SPaGAN (75) (d) SPaGAN (100)

(e) Lipizzaner (25) (f) Lipizzaner (50) (g) Lipizzaner (75) (h) Lipiz. (100)

Fig. 4: FID score distribution through the grid at different epochs (expresed between parentesis) of an independent run for SPaGAN and Lipizzaner in a 4×4 grid. Lighter blues represent lower (better) FID scores.

Table 3: TVD results (Low TVD indicates more diversity).

Grid	Method	Mean±Std	Median	Iqr	Min	Max
3×3	Lipizzaner	0.12±0.03	**0.12**	0.04	**0.08**	0.18
	SPaGAN	**0.12±0.02**	**0.12**	0.02	**0.08**	**0.16**
	IsoCoGAN	0.83±0.08	0.82	0.14	0.69	0.91
	PaGAN	0.14±0.02	0.14	0.04	0.10	0.19
4×4	Lipizzaner	**0.11±0.02**	**0.11**	0.02	**0.05**	**0.16**
	SPaGAN	0.12±0.02	0.12	0.03	0.08	**0.16**
5×5	Lipizzaner	**0.10±0.02**	**0.11**	0.02	**0.06**	0.16
	SPaGAN	0.11±0.02	**0.11**	0.03	0.07	**0.15**

SPaGAN and Lipizzaner according to the Mann-Whitney U test. SPaGAN and Lipizzaner show the best results obtaining the same **Mean**, **Median**, and **Min** values. Therefore, when using communication between the neighborhoods during the training, the generators are able to produce more diverse data samples.

When increasing the grid size, SPaGAN and Lipizzaner improve their results. However, SPaGAN does not show statistical differences between the same algorithm with different grid sizes. Lipizzaner 5×5 provides statistically better results than Lipizzaner 3×3 and than SPaGAN. The coevolutionary approach used in Lipizzaner takes advantage of the divergence generated when increasing the grid size to train generators that create more diverse data samples.

According to these results and the ones in Section 5.1, we can answer **RQ1:** *What is the effect on the quality of the generators when training with communication or isolation and the presence or absence of selection pressure?* Communication and selection pressure allowed Lipizzaner to converge to generators with the best quality (FID and TVD). The diversity resulting from communication resulted in the most competitive results, i.e., Lipizzaner and SPaGAN ended with better generators than IsoCoGAN and PaGAN. Isolation in training converged to good solutions when the cell was optimizing only one GAN (PaGAN).

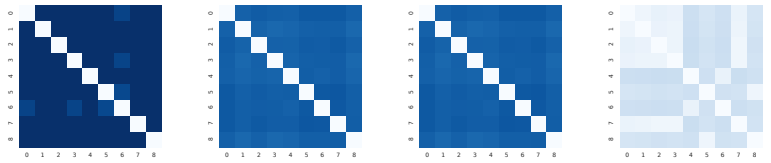
However, when isolation is coupled with a sub-population that applies selection/replacement, the algorithm was not able to converge, and therefore, the quality of the generators returned is the worst.

5.4 Genome Space Diversity

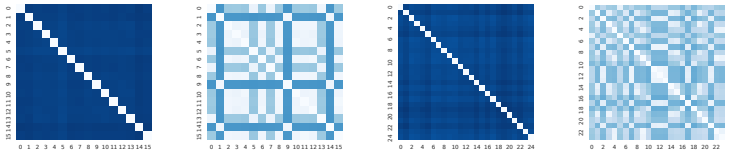
We next investigate the diversity of the parameters of the evolved networks. Table 4 summarizes the L_2 distance results for the population at the end of the independent run that returned the median FID. Fig. 5 shows the L_2 distances between all the generators in the grid (x and y axes are the cell number).

Table 4: Diversity of population (whole grid) in *genome* space. L_2 distances between the generators at the final generation.

Grid	Method	Mean±Std	Median	Iqr	Min	Max
3×3	`Lipizzaner`	13.85±8.29	18.06	15.32	4.09	23.46
	`SPaGAN`	72.70±25.75	81.72	2.66	78.32	84.82
	`IsoCoGAN`	71.07±25.29	79.42	5.67	74.00	86.01
	`PaGAN`	**109.98±41.15**	**128.26**	27.89	**90.82**	**138.02**
4×4	`Lipizzaner`	30.99±24.28	38.09	55.82	2.72	61.34
	`SPaGAN`	**87.90±22.71**	**93.75**	1.36	**91.44**	**95.95**
5×5	`Lipizzaner`	31.98±15.31	36.71	21.32	4.28	52.14
	`SPaGAN`	**87.96±18.06**	**91.60**	3.30	**87.22**	**96.87**



(a) `PaGAN` (3×3) (b) `IsoCoGAN` (3×3) (c) `SPaGAN` (3×3) (d) `Lipiz.` (3×3)



(e) `SPaGAN` (4×4) (f) `Lipiz.` (4×4) (g) `SPaGAN` 5×5 (h) `Lipiz.` (5×5)

Fig. 5: Diversity of population in *genome* space. Heatmap of L_2 distance between the generators at the final generation. Dark indicates more diversity.

The highest generator diversity is shown by `PaGAN`. This is because this method trains a single generator against a single discriminator and it does not use any type of information exchange or communication during the training process. Therefore, each cell converges to different points of the search space.

`SPaGAN` and `IsoCoGAN` provide networks with similar diversity in 3×3 grid, which is higher than the diversity provided by `Lipizzaner`. This is mainly due to the combination of both communication and the use of selection/replacement in `Lipizzaner` provokes the sub-populations to converge to similar accurate individuals, limiting diversity.

The genome diversity in Lipizzaner is increased when increasing the size of the grid (see in Fig. 5 how the heatmap gets darker as the grid size is larger). SPaGAN increases the L_2 distances with the grid size, but in a lower proportion, e.g., from 3×3 to 4×4 Lipizzaner increases 123.75% and SPaGAN 20.90%. The diversity in the genome space helps the creation of ensembles that are able to provide better accuracy. For this reason, Lipizzaner can improve results with an increasing grid size (and diversity).

Answering **RQ2:** *What is the effect on the diversity of the network parameters when training with communication or isolation and the presence or absence of selection pressure?* The combination of communication and selection pressure permits Lipizzaner to converge to similar high-quality generators. Complete isolation and no-population based GAN training (PaGAN) converge to highly different generators, which is expected. SPaGAN illustrates how the absence of selection pressure in the populations keep the individuals diverse in the grid, although there is communication. In 3×3 grid experiments, IsoCoGAN and SPaGAN have similar diversity but highly different quality. This shows how maintaining diversity is not enough to ensure robust GAN training.

5.5 Computational Efficiency

Now, we analyze the computational efficiency of the GAN training methods, taking into account that they use the same number of training epochs but used different components. All these methods apply asynchronous parallelism and the time required by each cell of the grid to perform the same number of training epochs varies. Thus, we report the computational time of a run as the time required by each independent run to finish and return the best ensemble of generators found by all the cells.

Table 5: Computation time in minutes.

Grid	Method	Mean±Std	Median	Iqr	Min	Max
3×3	Lipizzaner	87.89±1.15	87.73	1.23	85.57	90.23
	SPaGAN	87.20±0.31	87.15	0.34	86.72	88.02
	IsoCoGAN	81.88±4.55	81.34	9.41	74.40	88.19
	PaGAN	**38.07±2.73**	**37.51**	3.49	**33.57**	**44.28**
4×4	Lipizzaner	91.30±0.94	91.07	1.01	90.23	94.26
	SPaGAN	**90.72±0.58**	**90.89**	0.39	**88.97**	**91.27**
5×5	Lipizzaner	105.64±3.25	107.22	4.48	100.25	111.05
	SPaGAN	**101.88±1.64**	**100.91**	2.10	**100.18**	**105.52**

The computational time of PaGAN includes the whole process, i.e., the time of training the GANs plus the time of optimizing the ensemble weights by using the ES-(1+1). This method requires the shortest run time (see Table 5). This is because PaGAN trains a single network in each cell and there is no communication among the different cells. According to the Mann-Whitney U and posthoc statistical tests, Lipizzaner requires the longest computation times. It employs both communication and selection/replacement algorithm components.

When comparing among `Lipizzaner`, `SPaGAN`, and `IsoCoGAN`, the impact on the computation time of applying communication in `Lipizzaner` and `SPaGAN` is higher than the use of selection/replacement in `Lipizzaner` and `IsoCoGAN` (see Table 5 *3×3-Grid*). This increase in the computation cost may increase in problems that require the use and exchange among cells of bigger models (networks) for the generators and discriminators.

Therefore, answering **RQ3:** *What is the impact on the computational cost of applying migration and selection/replacement?*, the effect of applying selection/replacement is negligible when comparing it with the impact of communicating among the cells, when running the methods on 3×3 grid. However, it increases as the grid size is larger. Moreover, when bigger networks need to be trained to handle harder problems, they will have many more parameters and more need for communications.

6 Conclusions and Future Work

We have empirically shown that the spatially distributed coevolutionary training applied by `Lipizzaner` is the best choice among options with/without communication and selection/replacement components to train GANs. The combination of selection pressure that promotes convergence in the sub-populations and communication with the overlapped neighborhoods maintains enough diversity to robustly train the networks in the sub-population. Moreover, the use of these two operations does not entail a very significant increase in the computation time (about four minutes in the larger grid size).

`SPaGAN` illustrates the importance of the communication among the cells (i.e., fostering diversity). It is able to converge to high-quality solutions (generators) although it does not apply selection. This emphasizes the value of exchanging the best individuals even they are just used to train the center of the cells. `IsoCoGAN` provides the least competitive results. Training the networks with a coevolutionary flavor does not ensure convergence, even when it is done in sub-populations and it uses selection/replacement.

Future work will include further analysis of coevolutionary GAN training in other benchmarks (i.e., problems and data sets). We will evaluate the scalability of this type of training by using larger grids. We will study the effect of training the GANs with different kinds of loss functions. We will assess the impact on the robustness when using different types of neighborhoods. Finally, we will analyze the evolution of the network weights through the generations to better understand the dynamics of this type of GAN training.

Acknowledgments

This research was partially funded by European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No 799078, by the Junta de Andaluía UMA18-FEDERJA-003, European Union H2020-ICT-2019-3, and the Systems that Learn Initiative at MIT CSAIL.

References

1. Al-Dujaili, A., Schmiedlechner, T., Hemberg, E., O'Reilly, U.M.: Towards distributed coevolutionary GANs. In: AAAI 2018 Fall Symposium (2018)
2. Alba, E., Dorronsoro, B.: Cellular genetic algorithms, vol. 42. Springer Science & Business Media (2009)
3. Alba, E., Luque, G., Nesmachnow, S.: Parallel metaheuristics: recent advances and new trends. International Transactions in Operational Research **20**(1), 1–48 (2013)
4. Arjovsky, M., Chintala, S., Bottou, L.: Wasserstein gan. arXiv preprint arXiv:1701.07875 (2017)
5. Arora, S., Ge, R., Liang, Y., Ma, T., Zhang, Y.: Generalization and equilibrium in generative adversarial nets (gans). arXiv preprint arXiv:1703.00573 (2017)
6. Chavdarova, T., Fleuret, F.: Sgan: An alternative training of generative adversarial networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 9407–9415 (2018)
7. Chintala, S., Denton, E., Arjovsky, M., Mathieu, M.: How to train a gan? tips and tricks to make gans work. https://github.com/soumith/ganhacks (2016)
8. Essaid, M., Idoumghar, L., Lepagnot, J., Brévilliers, M.: Gpu parallelization strategies for metaheuristics: a survey. International Journal of Parallel, Emergent and Distributed Systems **34**(5), 497–522 (2019)
9. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: Advances in neural information processing systems. pp. 2672–2680 (2014)
10. 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 pp. 6626–6637 (2017)
11. Hodjat, B., Hemberg, E., Shahrzad, H., O'Reilly, U.M.: Maintenance of a long running distributed genetic programming system for solving problems requiring big data. In: Genetic Programming Theory and Practice XI, pp. 65–83. Springer (2014)
12. LeCun, Y.: The mnist database of handwritten digits. http://yann. lecun. com/exdb/mnist/ (1998)
13. Li, C., Alvarez-Melis, D., Xu, K., Jegelka, S., Sra, S.: Distributional adversarial networks. arXiv preprint arXiv:1706.09549 (2017)
14. Li, J., Madry, A., Peebles, J., Schmidt, L.: Towards understanding the dynamics of generative adversarial networks. arXiv preprint arXiv:1706.09884 (2017)
15. Mao, X., Li, Q., Xie, H., Lau, R.Y., Wang, Z., Paul Smolley, S.: Least squares generative adversarial networks. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 2794–2802 (2017)
16. Mordido, G., Yang, H., Meinel, C.: Dropout-gan: Learning from a dynamic ensemble of discriminators. arXiv preprint arXiv:1807.11346 (2018)
17. Neyshabur, B., Bhojanapalli, S., Chakrabarti, A.: Stabilizing gan training with multiple random projections. arXiv preprint arXiv:1705.07831 (2017)
18. Nguyen, T., Le, T., Vu, H., Phung, D.: Dual discriminator generative adversarial nets. In: Advances in Neural Information Processing Systems. pp. 2670–2680 (2017)
19. Popovici, E., Bucci, A., Wiegand, R.P., De Jong, E.D.: Coevolutionary principles. In: Handbook of natural computing, pp. 987–1033. Springer (2012)
20. Radford, A., Metz, L., Chintala, S.: Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434 (2015)

21. Salimans, T., Ho, J., Chen, X., Sutskever, I.: Evolution strategies as a scalable alternative to reinforcement learning. arXiv:1703.03864 (2017)
22. Schmiedlechner, T., Yong, I.N.Z., Al-Dujaili, A., Hemberg, E., O'Reilly, U.M.: Lipizzaner: A System That Scales Robust Generative Adversarial Network Training. In: the 32nd Conference on Neural Information Processing Systems (NeurIPS 2018) Workshop on Systems for ML and Open Source Software (2018)
23. Stanley, K.O., Clune, J.: Welcoming the era of deep neuroevolution - uber engineering blog. https://eng.uber.com/deep-neuroevolution/ (December 2017)
24. Talbi, E.G.: Metaheuristics: from design to implementation, vol. 74. John Wiley & Sons (2009)
25. Tolstikhin, I.O., Gelly, S., Bousquet, O., Simon-Gabriel, C.J., Schölkopf, B.: Adagan: Boosting generative models. In: Advances in Neural Information Processing Systems. pp. 5430–5439 (2017)
26. Toutouh, J., Hemberg, E., O'Reilly, U.M.: Spatial evolutionary generative adversarial networks. In: Proceedings of the Genetic and Evolutionary Computation Conference. pp. 472–480. GECCO '19, ACM, New York, NY, USA (2019). https://doi.org/10.1145/3321707.3321860
27. Toutouh, J., Hemberg, E., O'Reilly, U.M.: Data dieting in gan training. In: Iba, H., Noman, N. (eds.) Deep Neural Evolution: Deep Learning with Evolutionary Computation, pp. 379–400. Springer Singapore, Singapore (2020)
28. Toutouh, J., Hemberg, E., O'Reilly, U.M.: Re-purposing heterogeneous generative ensembles with evolutionary computation. In: Proceedings of the Genetic and Evolutionary Computation Conference. GECCO '2020, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3377930.3390229, `https://doi.org/10.1145/3377930.3390229`
29. Wang, C., Xu, C., Yao, X., Tao, D.: Evolutionary generative adversarial networks. IEEE Transactions on Evolutionary Computation **23**(6), 921–934 (2019)
30. Wang, Y., Zhang, L., van de Weijer, J.: Ensembles of generative adversarial networks. arXiv preprint arXiv:1612.00991 (2016)
31. Wierstra, D., Schaul, T., Peters, J., Schmidhuber, J.: Natural evolution strategies. In: Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on. pp. 3381–3387. IEEE (2008)
32. Zhao, J., Mathieu, M., LeCun, Y.: Energy-based generative adversarial network. arXiv preprint arXiv:1609.03126 (2016)