Spatial Coevolution for Diverse Training of Generative Adversarial Networks

A NON, Some

We study a system, Lipizzaner, that combines spatial coevolution with gradient-based learning to improve the robustness and scalability of generative adversarial network (GAN) training. Our first question is: Are different features of Lipizzaner's evolutionary computation methodology necessary and advantageous? Our ablation experiments determine that communication, selection, hyperparameter optimization and ensemble weight optimization each, and in combination, play critical roles. We also experiment with a GAN-training feature of Lipizzaner: the ability to train simultaneously with different loss functions in the gradient descent parameter learning framework of each GAN at each cell. We choose an image generation problem to ask a second question: Do different loss function combinations result in models with better accuracy and more diversity? We find that a randomized approach with multiple loss function options promotes the best model diversity while requiring a larger grid size to attain adequate accuracy.

Additional Key Words and Phrases: Generative adversarial networks, coevolution, diversity

ACM Reference Format:
A Non. 2018. Spatial Coevolution for Diverse Training of Generative Adversarial Networks. *J. ACM* 37, 4, Article 111 (August 2018), 26 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Generative modeling aims to learn a function that describes a latent, i.e. unknown, distribution. In a common setup, generative models undergo gradient-based training using observations from the latent distribution and they learn either an estimate of this distribution (explicit density) or how to generate samples from it (implicit density). The trained model can then be used to generate unobserved samples.

Generative Adversarial Networks (GANs) consist of two neural network models: a discriminator and a generator [19]. GANs use an adversarial paradigm. The discriminator's training objective is to distinguish real samples in the training data set from samples synthesized by the generator. The training objective of the generator is to fool the discriminator with its samples, synthesized from a latent input space and a non-linear function. This forms a coupled minimization and maximization "*minimax*" problem. The adversarial paradigm introduces GAN training challenges. During unsuccessful training, the networks' parameters stabilize into a sub-optimal equilibrium or continually oscillate between sub-optimal performance levels. After successful training, the generator serves as the generative model [19].

In this contribution we study Lipizzaner, a system originally designed and demonstrated to reliably address the degenerate behaviors of mode and discriminator collapse [1, 55, 62] and efficiently yield generative models. Lipizzaner brings spatially distributed coevolution to GAN training. It consists of an asynchronous competitive coevolutionary algorithm executing on an

Author's address: A Non, Some, a@non.

0004-5411/2018/8-ART111 $15.00
https://doi.org/10.1145/1122445.1122456

2D spatial grid of cells organized into overlapping neighborhoods. Two competitive coevolving sub-populations exist on each cell – one of generators and the other of discriminators, collected from the cell and its adjacent neighbors. Training occurs pairwise across the sub-populations. The neural network models' are updated with stochastic gradient descent (SGD) using the minimax objective, following conventional machine learning methods. The training hyper-parameters such as learning rate of SGD are learned with an evolution-based method. Between training epochs, the competing sub-populations are updated with copies of best neural network models from the cell's local neighborhood. This is a form of implicit communication that relies upon the overlap of the cell neighborhoods. Each individual cells' solutions improve because of the training, while the epoch by epoch spatial propagation of best solutions prevents local pathologies from persisting. Combined, this approach ensures robustness.

While `Lipizzaner` [1] has many typical features of evolutionary computation, it also has many unique features. These features include spatial communication, local (sub-population) selection, and the evolution of generator ensemble weights and the initial value of its networks' learning rate under SGD, which is central to GANs. Our first question, **RQ-1**, is whether each of these different features of `Lipizzaner`'s evolutionary computation methodology are needed and helpful through ablation.

Generally the ideal loss functions for GAN training are unknown. In seeking the best model that spans the entire latent distribution, a developer typically has to train with the same loss function multiple times due to sensitivity to initial conditions. Parameter training arguably could benefit from different loss functions at different times during training, but these are unknown. `Lipizzaner` has the ability to train simultaneously with different loss functions in the gradient descent parameter learning framework of each GAN at each cell. Its evolutionary adaptation, including local selection and spatial communication, change of adversary (when a neighbor's updated parameters are incorporated into a local neighborhood) and initial parameter values for a given loss function (when a cell using a specific loss function trains from neighbors' parameters). We therefore ask a second question, **RQ-2**: Do different loss function combinations result in generative models with better performance and higher diversity of solutions and model parameters? We draw upon four different loss functions, see Section 4.2, to configure six experimental setups of loss function combinations, see Table 5, and compare them. Effectively, using different loss functions explicitly imposes pressure for population-wide diversity in adaptation. We check this diversity by comparing the parameters of different trained models of `Lipizzaner`. As well, we check whether the diversity in loss function choice results in better coverage of the latent space. We deliberately choose an image generation problem that allows us to gauge *total variational distance* [35] and the coverage of every class in the image set. In addition, we measure the generative model performance with Frechet Inception Distance (FID) score [23].

We extend previous work on `Lipizzaner` [1, 55, 62] and provide the following contributions:

1) We describe, in multiple levels of detail, `Lipizzaner`, a scalable, distributed, GAN training system.

2) We establish the necessity of spatial communication, local selection and GAN training hyper-parameter evolution toward better performance by `Lipizzaner`.

3) We describe the multiple loss functions `Lipizzaner` currently supports.

4) We experimentally confirm that combinations of different loss functions, including randomization of loss function choice, while slightly more computationally expensive, are advantageous in the common circumstance of not knowing the ideal loss function.

5) We experimentally determine that combinations of different loss functions result in trained model parameter and output diversity.

The rest of the paper is organized as follows. Section 2 presents related work. Notation is presented in Section 3. Section 4 describes the method. The experimental setup and the results for empirical data are in Section 5. Finally, conclusions are drawn and future work is outlined in Section 6.

2 RELATED WORK

In this section we discuss work related to competitive coevolutionary algorithms for minimax problems (Section 2.1), how evolutionary computation and GANs have been combined in prior work (Section 2.2), scaling evolutionary computation for machine learning (Section 2.3), and common approaches to GAN training (Section 2.4).

2.1 Coevolutionary Algorithms for Minimax Problems

There is extended work on coevolutionary algorithms covering multi-objective optimization [18, 42], active learning [32], large scale optimization [71] and strategic decision support [31]. Competitive coevolutionary algorithms have adversarial populations (usually two) that simultaneously evolve solutions against each other [17, 49] with members engaging in 2-player games. They employ fitness functions that rate a solution relative to its *adversaries* and can sometimes be described as a zero-sum game or, more generally, a minimax optimization [2, 7, 8, 22]. In seminal work, Hillis [25] showed that more efficient *sorting programs* can be produced by competitively co-evolving them versus their *testing programs*.

Without explicit remedies, competitive coevolutionary algorithms experience what we call *pathologies* or *limitations* in the form of non-convergent behavior, i.e. oscillations, and undesired but stable equilibria. As such, they share similarities with GAN training. The GAN training can also be seen as a two-player game – one which is solved using gradient-based updates to simultaneously optimize a minimax objective. Commonly observed GAN training pathologies are *mode collapse* [6] (the generator can only generate one mode of the distribution), *discriminator collapse* [36] (the discriminator can not discriminate any modes), as well as *vanishing gradients* [3] (there are no training gradients, i.e. one adversary is too strong/weak).

Reasons offered for a limitation are usually accompanied by a general remedy and one or more methods to implement the remedy. For example, one limitation is due to *asymmetry*. *Asymmetry* means that the fitness is not symmetrically applied to each of the adversaries. *Asymmetric fitness evaluation* has been applied as a remedy [10, 29, 50]. The equivalent in GAN training is the use of Binary Cross Entropy as a loss metric for a generator and Mean Squared Error as a loss metric for a discriminator. As another example, [15] notes *intransitive (cyclic) superiority* and addresses it with more specific measures of fitness and genotypes to track the evolutionary dynamics. Finally, Watson and Pollack [66] used a collection of simple number games to illustrate three important coevolutionary limitations namely, *focusing*, which is a parallel to GAN mode collapse, *relativism*, which is a parallel to GAN discriminator collapse, and *loss of gradients*, which is a parallel to GAN vanishing gradients.

A general reason offered for these limitations is that, despite the algorithm's stochasticity, the populations lack sufficient solution diversity to disrupt premature convergence in the form of an oscillation or move the search away from an undesired equilibria. Essentially, the population should serve as a source of novelty and a genotypically or phenotypically converged population fails in this respect. One method of preventing convergence is to supplement one or both populations with a memory that holds genotypes that have been previously visited. Memory has been implemented as a hall-of-fame [48] or pareto archive [49]. Solutions in the memory also serve as reference points for measuring the relative quality of a current solution and detecting a diminishing gradient. Solution

diversity has also been explicitly improved with competitive fitness sharing [52], separation, e.g. a spatial topology [43] or a spatial topology and temporal segregation [21].

A spatial (2D toroidal) topology is an effective means of controlling the mixing of adversarial populations in coevolutionary algorithms [43]. In a conventional setup, the members of populations are divided up on a grid of cells and overlapping neighborhoods for each cell are identified, see Figure 1. A neighborhood is defined by the cell itself and its adjacent cells and specified by its size, s. Coevolution proceeds at each cell with sub-populations drawn from the neighborhood. Compared to an algorithm where all members can interact with each other, this reduces the cost of overall interaction from $O(N^2)$ to $O(Ns)$, where N is the size of each population. A Moore neighborhood with five cells (center and four cells respectively North, East, South and West) is commonly used [27], see Figure 1. With this design, each neighborhood can evolve in semi-isolation and more diverse points in the search space are explored [43, 68]. It is this approach: spatial separation via an abstract cellular grid, that `Lipizzaner` applies to GAN training.

2.2 Evolutionary Computation, Neural Nets and GANs

Evolutionary Computation has been used to design neural networks through neuroevolution [58]. E.g. NEAT and HYPER-NEAT [58, 59] have been updated to evolve deep network architectures [16]. In addition, population-based training of neural networks [28] has been investigated. Moreover, a combination of elements from deep learning and artificial life demonstrated that coevolutionary neural population models can simulate population dynamics, and that evolutionary game theory can describe the behavior [44].

One example of combining evolutionary computation and GANs is E-GAN [64]. It evolves a population of three independent loss functions defined according to three metrics (Jenson-Shannon Divergence –JSD, Least Squares – LS distance, and a combination of JSD and Kullback-Leibler (KL) divergence) [64]. Each loss function is independently used to train a generator from some starting condition, over a batch of training data. The generators produced by the loss functions are evaluated by a single discriminator that returns a fitness value for each generator. The best generator is then selected and training continues, with the next training batch, and the three different loss functions. The use of different loss functions overcomes the limitations of a single individual training objective and better adapts the population to the evolution of the discriminator. The results for problems in the image domain have shown that E-GAN is able to obtain higher (better) inception scores, compared to a canonical GAN baseline, while showing comparable stability when it converges. E-GAN relies on the evolutionary population to introduce diversity into the training by its different loss functions and it uses what can be described as evolutionary selection when updating the best generator of a batch. A second example, leading to `Lipizzaner`, is described in [1, 54]. This approach was a competitive coevolutionary algorithm but it did not include the spatial distribution of the population. Interestingly, it included a variation which even evolved (with CMA-ES) the network parameters, though this variant was not selected for `Lipizzaner` due to its inefficiency over using the gradient information that is readily available.

2.3 Scaling Evolutionary Computing for Machine Learning

One salient contribution of `Lipizzaner` is its scalability. There are other examples of scalable and large-scale evolutionary computation systems. One is EC-Star [26] which runs on hundreds of desktop machines. In another, the team of [53] applied a simplified version of Natural Evolution Strategies [67] with a novel communication strategy to a collection of reinforcement learning benchmark problems. Due to better parallelization over thousand cores, they achieved fast training times (wall-clock time). They ran their experiments on a computing cluster of 80 machines and 1, 440 CPU cores [53]. Another team [57] showed that deep convolutional networks with over 4 million

parameters trained with genetic algorithms can also reach competitive results. They employed a range of hundreds to thousands of CPU cores (depending on availability). Deep networks have also been evolved on HPC [69].

The bulk of attempts to improve GAN training have been designed to run on a single machine (or a single GPU). The advent of large-scale parallel computation infrastructure enables us to scale them to spatial grids.

2.4 Training GANs

Robust GAN training is still an open research topic [5]. Simple theoretical models have been proposed to provide a better understanding of the problem [36]. These were investigated in [1] which describes preliminary work that establishes that an evolutionary algorithm can ameliorate specific training problems. This work formed the basis of `Lipizzaner`. Coevolution is one approach among many, none offering a complete resolution. One example is hard-coded conditions to decrease the optimizers' learning rate after a given number of iterations [51]. Several heuristic practices have been suggested to stabilize the training [14]. There are also ensemble approaches and approaches that change the generator's or discriminator's cost function.

Heuristic Approaches. Goodfellow in his 2014 paper describes basic GAN training [20]. Some training aspects can be modified, e.g. flipping a maximization objective to a minimization objective to obtain a more informative gradient. See [11] for more details.

Ensemble Approaches. The use of multiple generators and/or discriminators for improving robustness has also been studied. This includes: training a cascade of GANs [65]; sequentially training and adding new generators with boosting techniques [61]; training multiple generators and discriminators in parallel [30]; training an array of discriminators specialized in a different low-dimensional projection of the data [46]; using several adversarial "local" pairs of networks that are trained independently so that a "global" supervising pair of networks can be trained against them [12]; and learning from a dynamic ensemble of discriminators [45]. `Lipizzaner` also takes an ensemble approach. It is distinctive in evolving ensemble at a local (neighborhood) level, co-tuning the ensemble weights and supporting the choice of the best ensemble (among all neighborhoods).

Cost Function Options. Cost function approaches change the original *Jensen-Shannon divergence (JSD)* loss function [20] to alternative functions in an effort to address poor GAN training. There are multiple studies of alternate functions, including e.g. [9] which includes a large table of GAN evaluation measures and [40, 56] which use two measures that approximate the recall (diversity) and precision (quality of the image). [4] introduces the popular Wasserstein loss which we include in our empirical comparison studies. There are also domain-specific measures outside images, e.g. for text probability-based language model metrics [60]. Other cost function approaches include [41, 47, 72]. No single loss function is a solution for all problems.

`Lipizzaner` incorporates the above approaches in its evolutioanry GAN training. We next present our notation and formalize the goal of GAN training.

3 NOTATION AND GOAL OF GAN TRAINING

We adopt notation similar to [5, 36]. Let $\mathcal{G} = \{G_g, g \in \mathbb{G}\}$ and $\mathcal{D} = \{D_d, d \in \mathbb{D}\}$ denote sets of generators and discriminators, where G_g and D_d are functions parameterized by g and d. $\mathbb{G}, \mathbb{D} \subseteq \mathbb{R}^p$ represent the respective parameter space of the generators and discriminators. Finally, let G_* be the unknown target distribution to which we would like to fit our generative model.

Formally, the goal of GAN training is to find parameters g and d in order to optimize the objective function

$$\min_{g \in \mathbb{G}} \max_{d \in \mathbb{D}} \mathcal{L}(g, d) \ , \text{ where}$$

$$\mathcal{L}(g, d) = \mathbb{E}_{x \sim G_*}[\phi(D_d(x))] + \mathbb{E}_{x \sim G_g}[\phi(1 - D_d(x))] \ , \tag{1}$$

and $\phi : [0, 1] \rightarrow \mathbb{R}$, is a concave *measuring function*. In practice, we have access to a finite number of training samples $x_1, \ldots, x_m$ to approximate G_* or G_g. Therefore, an empirical version $\frac{1}{m} \sum_{i=1}^{m} \phi(D_d(x_i))$ is used to estimate $\mathbb{E}_{x \sim G_*}[\phi(D_d(x))]$.

With these formalities in place, in the next section, we describe `Lipizzaner`.

4 THE LIPIZZANER FRAMEWORK

Outline. In this section we describe the framework we call `Lipizzaner` for improved GAN training by promoting diversity. `Lipizzaner` uses spatial distribution to address the quadratic computational complexity of the basic competitive coevolutionary algorithm's adversarial competitions. It also includes:

- gradient-based learning to update the neural nets' parameters,
- Gaussian-based mutations to update the hyperparameters,
- evolutionary selection then replacement and
- overlapping neighborhoods with cells as sub-populations.

First, we take the reader through a walkthrough of `Lipizzaner` considering a specific use case. Then, we more formally describe its algorithms. Finally, we highlight `Lipizzaner`'s different means of promoting diversity.

4.1 High Level Description with Example Use Case

We describe the use of `Lipizzaner` to model the images of the CelebA dataset [38]. `Lipizzaner` sets up an abstract 4×4, 2-dimensional toroidal grid where each cell contains a generator and discriminator. Each grid cell (Center) designates a neighbor to the North, South, West and East. `Lipizzaner` regards the generators and discriminators of the cell and its neighbors as two sub-populations, one of generators and one of discriminators, see Figure 1. The cell also has two hyper-parameters: the learning rate during parameter training and five weights for its neighborhood mixture of solutions.

In parallel but asynchronously, see Figure 2, each cell executes the same generational algorithm which is illustrated in Figure 3. Depending on how long a generation takes to compute, this asynchronicity can have varying impact on the GAN training. Given each cell has its sub-populations of generators and discriminators, any generator (discriminator) can be evaluated pairwise with any discriminator (generator), i.e. GAN training. A generator's fitness is derived from its pairwise performance against all discriminators in hte sub-population (neighborhood). Pairwise performance is measured by using the generator to generate a number of images and summing its errors with respect to its paired discriminator detecting its images as "real" (good for the generator, bad for the discriminator) or "fake" (good for the discriminator, bad for the generator). The discriminator's performance is how frequently it correctly detects a "fake", i.e. image from the generator.

Within the cell's generational algorithm, models are selected (and replicated) based on fitness before they undergo training. The fitness function for the CelebA dataset, because it is images, is Fréchet Inception Distance (FID) score [23]. GAN training proceeds by mini-batch. For each mini-batch, first, a random discriminator is drawn from the sub-population. All generators are trained pairwise with this discriminator. Here, to promote diversity, the models are pairwise randomly assigned a loss function to use during the GAN training. Any or some or all of three loss

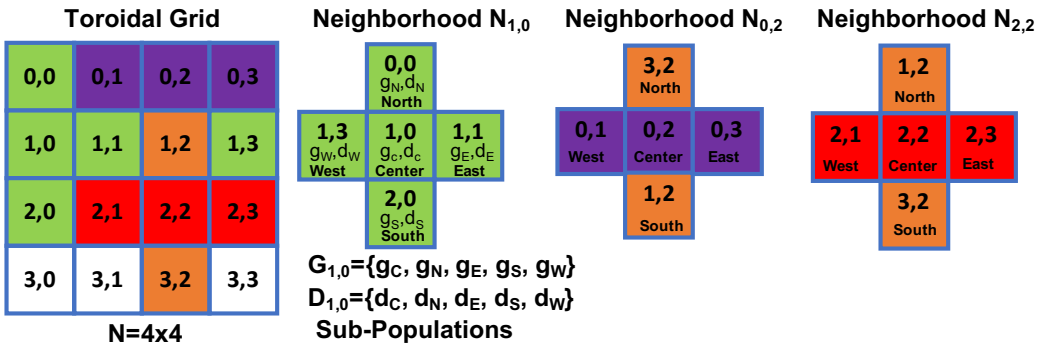


Fig. 1. Illustration of overlapping Moore neighborhoods on a toroidal 4×4 grid, color indicates 3 different neighborhood and similar color in same neighborhood indicates overlap. Note how a cell update at $N_{1,0}$ can be communicated to $N_{1,1}$ when $N_{1,1}$ gathers its neighbors. If $N_{1,1}$ is then updated with the updated value from $N_{1,0}$ the value has propagated. The $N_{1,2}$ value is in the range of both $N_{0,2}$ and $N_{2,2}$. Propagation runs laterally and vertically. We also show an example of a cell's generator and discriminator sub-populations (based on its neighborhood) for $N_{1,0}$.

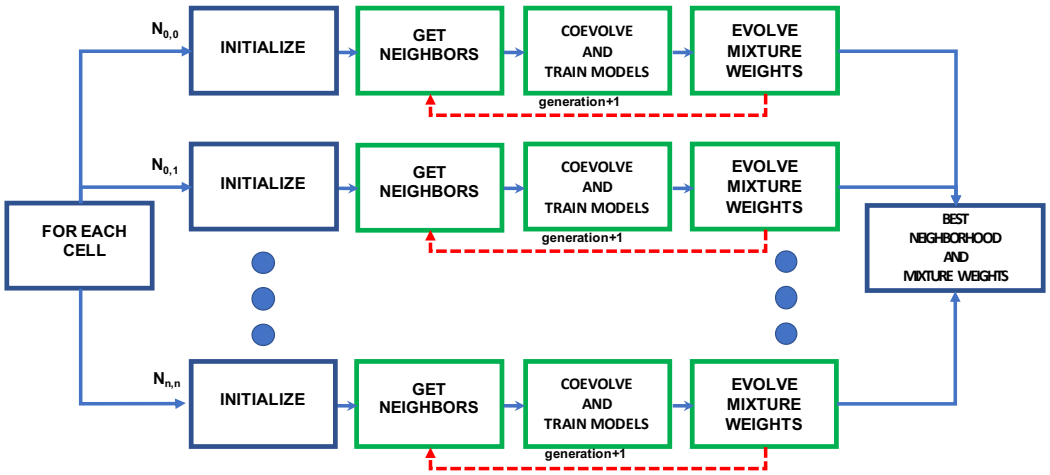


Fig. 2. Flowchart of `Algorithm 1::Lipizzaner`, the different colord boxes are elaborated in Figure 3. Each cell executes in parallel, asynchronously. Each cell then iterates over a number of generations. At the beginning of a generation neighbors are collected while at the end the cell itself is updated, see line 4 of `Algorithm 1::Lipizzaner` and line 19 of `Algorithm 2::CoevolveAndTrainModels`. Information propagation is facilited by each cell updating itself with the best of neighborhood (see `Algorithm 2::CoevolveAndTrainModels` line 19)

functions (with corresponding minimization objectives w.r.t. the generator) are used in the CelebA use case: binary cross-entropy, least square, and a KL divergence- based heuristic mutation [64], see Section 4.2. Then, one of the (trained) generators is randomly drawn and all discriminators are trained pairwise with it. All GAN training is done with stochastic gradient descent (SGD) based upon loss calculated with the model's randomly assigned loss function.

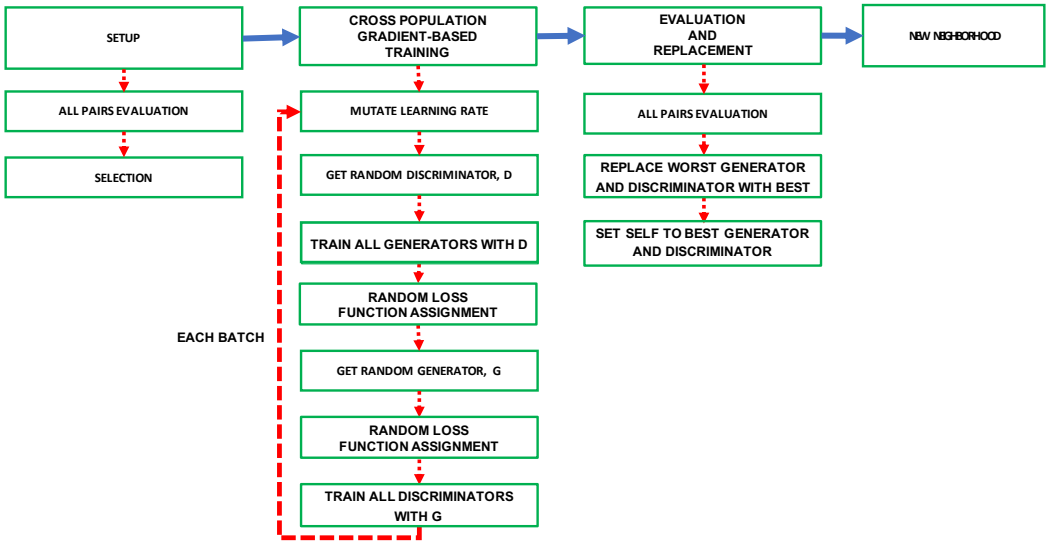


Fig. 3. Flowchart of `Algorithm 2::Lipizzaner` (expansion of the different colored boxes in Figure 2). This cell-centered algorithm is called each generation, after generator and discriminator neighbors from adjoining cells are gathered. At each time step (generation), it performs evolutionary selection and replacement on generators and discriminators that are improved with SGD. Note also the use of random loss function assignment to promote diversity and the mutation of the training learning rate. "Best" is based on a solution concept, a paramterized choice of best, average or worst loss, each model being assigned this based on the loss over all its adversarial models.

After multiple mini-batches, fitness is once again evaluated so that the discriminator and generator of the now trained sub-population with the lowest FID score replace those of the cell and the worst of the sub-population. This sub-population is then considered collectively. `Lipizzaner` evolves "weights" for each of the five generators in the sub-population using random perturbation and keeping the perturbation if there is an improvement (also known as 1+1 Evolutionary Strategy (ES)). The weights are normalized and represent the likelihood a generator will be used to generate an image. The evolution of the weights will reflect the diminished value of badly performing generators and the increased value of good ones. The 1+1 ES measures the fitness of a set of weights with FID score on CelebA.

After some number of generations or when a time limit is reached, evolution within each cell halts. Every cell along with its neighbors is then evaluated as a weighted mixture to identify the best neighborhood of the grid. For this evaluation, each mixture weight is treated as a probability of using one generator versus the other generators. The mixture is used to generate multiple (fake) sample images and these are passed to an external evaluation function which returns a FID score reflecting the quality of the images with respect to the samples. This best mixture is `Lipizzaner`'s answer to the generative modeling problem.

We now proceed to a more formal description which references pseudocode and flowcharts.

4.2 Algorithmic Description of `Lipizzaner`

`Lipizzaner` starts with `Algorithm 1::Lipizzaner` which immediately starts parallel execution on each cell. Each cell randomly initializes single discriminator and generator neural network models (for readability we just refer to these as models when discriminators and generators are

Table 1. Descriptions of Lipizzaner notation (including domain). Parameters, variables and constants are shown. Variables are optimized, parameters also include hyper-parameters that are provided when starting Lipizzaner, and constants are used for the computational complexity analysis.

Parameters	
N	population size, derived from the grid size m as $m \times m$
m	the grid size, $m \in \mathbb{Z}$
s	neighborhood size, derived from the cardinality of the neighborhood $\lvert n \rvert$, $s \in \mathbb{Z}$
M	loss functions, $M \in \mathcal{M}$
T	number of generations (epochs), $T \in \mathbb{Z}$
τ	tournament size, $\tau \in \mathbb{Z}$
β	mutation probability learning rate, $\beta \in \mathbb{R}$
μ	mutation probability mixture weights, $\mu \in \mathbb{R}$
ω_0	initial mixture weights, $\omega_0 \in \mathbb{R}^{sN}$
δ_0	initial learning rate, $\delta_0 \in \mathbb{R}^N$
G^A	generator network architecture, $G^A \in \mathcal{G}^A$, $\lvert G^A \rvert \in \mathbb{Z}$ is the number of weights in the network
D^A	discriminator network architecture, $D^A \in \mathcal{D}^A$, $\lvert D^A \rvert \in \mathbb{Z}$ is the number of weights in the network
f_s	solution concept for selection and replacement, $f_s \in textbest, worst, average$
b	number of mini-batches, $b \in \mathbb{Z}$
e	dataset size, X is the dataset, $X \in \mathcal{X}$
Variables	
w_G	weights of the generator, $w_G \in [0, 1]^{\lvert G^A \rvert}$. The sub-population is $\mathbf{w}_G$
w_D	weights of the discriminator, $w_D \in [0, 1]^{\lvert D^A \rvert}$. The sub-population is $\mathbf{w}_D$
δ	learning rate of a neural network (generator/discriminator), a hyper-parameter, $\delta \in [0, 1]$
ω	mixture weights of the generator mixture (ensemble), a hyper-parameter, $\omega \in [0, 1]^s$
Constants	
C	GAN training cost, used for computational complexity analysis, $C \in \mathbb{Z}$

interchangeable). Thus two global (grid-scale) adversarial populations emerge, $\mathbf{g} = \{g_1, \ldots, g_N\}$ a population of generators and $\mathbf{d} = \{d_1, \ldots, d_N\}$ a population of discriminators, where N is the grid (population) size. These populations are one source of diversity in Lipizzaner, and we refer to this diversity as occurring in genome space. Each cell also initializes its hyperparameters – learning rate and mixture weights.

Additional Notation. A cell's *neighborhood* allows it to form two sub-populations of models: $\mathbf{g}$ and $\mathbf{d}$. We denote the size of this neighborhood by s and the number of neighborhoods as s_n. Without loss of generality, we consider $m \times m$ square grids. Given a $m \times m$ toroidal grid, there are m^2 neighborhoods. In our study, we use a five-cell Moore neighborhood, i.e, one center and four adjacent cells (see Figure 1).

For the k-th neighborhood in the grid, we refer to the generator in its center cell by $\mathbf{g}^{k,1} \subset \mathbf{g}$ and the set of generators in the rest of the neighborhood cells by $\mathbf{g}^{k,2}, \ldots, \mathbf{g}^{k,s}$, respectively. Furthermore, we denote the union of these sets by $\mathbf{g}^k = \cup_{i=1}^{s_n} \mathbf{g}^{k,i} \subseteq \mathbf{g}$, which represents the kth generator neighborhood.

One way of expressing Lipizzaner is therefore as a function f:

$$f : \mathcal{X} \times \Upsilon \rightarrow \mathcal{G} \times \mathcal{D}, \mathbf{G_g}, D_d = f(\mathbf{X}, \Upsilon) \tag{2}$$

the variables that are optimized are $\Upsilon = (\mathbf{w_G}, \mathbf{w_D}, \delta, \omega)$ see Table 1. In this paper we focus on finding the best ensemble of generators stated as:

$$\arg\min_{\Upsilon \in \mathbf{Y}} h(f(\mathbf{X}, \Upsilon)|\Theta) \tag{3}$$

where, within f, there is a min-max optimization of the GAN training (Eq. (1)), h is a measuring function, e.g. FID for generative models of images and Θ is the set of hyper-parameters. The hyper-parameters $\Theta = (m, s, M, T, \tau, \beta, \mu, \omega_0, \delta_0, G^A, D^A, f_s)$ are shown in Table 1.

Next we describe the steps in Lipizzaner in more detail.

Algorithm 1 `Lipizzaner`: In parallel, for each cell, initialize settings then iterate over each generation. Each generation, retrieve neighbor cells to build generator and discriminator sub-populations, evolve generators and discriminators trained with SGD, replace best with worst and update self with best, and finally evolve weights for a neighborhood mixture model.
Input: T : Total generations, E : Grid cells, k : Neighborhood size, θ_{EA} : Parameters for `MixtureEA`, θ_{COEV} : Parameters for `CoevolveAndTrainModels`
Return: n : neighborhood, ω : mixture weights

parfor $c \in E$ **do** ▷ Asynchronous parallel execution of all cells in grid
  $n, \omega \leftarrow$ initializeNeighborhoodAndMixtureWeights(c, k) ▷ Uniformly random initialization of settings
  for generation **do** $\in [0, \ldots, \mathrm{T}]$ ▷ Iterate over generations
    $n \leftarrow$ copyNeighbours(c, k) ▷ collect neighbour cells individuals for the sub-populations
    $n \leftarrow$ CoevolveAndTrainModels(n, θ_{COEV}) ▷ Coevolve GANs using `Algorithm` 2
    $\omega \leftarrow$ `MixtureEA`(ω, n, θ_{EA}) ▷ Evolve mixture weights, `Algorithm` 4
end parfor
return $(n, \omega)^*$ ▷ Cell with best generator mixture

Generation Loop. `Algorithm` 1::`Lipizzaner` directs each cell executing in parallel to iterate over a generational loop. Each generation, a cell starts by copying the latest versions of its neighbors to set up its sub-populations $\mathbf{g}^k$ and $\mathbf{d}^k$. These localized, spatially overlapping, smaller sub-populations are a source of diversity in `Lipizzaner`. Each cell next independently executes `Algorithm` 2::`CoevolveAndTrainModels`. This algorithm terminates with the cell updating its own models, i.e. $\mathbf{g}^{k,1}$, $\mathbf{d}^{k,1}$. These are two key steps that facilitate information propagation – at the beginning of generation each cell obtains updated copies of its neighbors and, at the end of each generation the cell updates itself.

In between, the cell coevolves with selection, trains models, then replaces the worst model with the best to update its sub-populations. Upon each cell returning its sub-populations to `Algorithm` 1::`Lipizzaner`, the mixture weights are evolved (see following paragraph). After all generations are completed (or a computational deadline is reached), `Algorithm` 1::`Lipizzaner` then selects the best performing neighborhood across the entire grid (see following paragraph) and returns it, along with its mixture weights, as `Lipizzaner`'s solution.

Selection and Replacement. Selection promotes fitter models over less fit ones when updating a sub-population. In `Lipizzaner` we use tournament selection. `Lipizzaner` first applies selection after sub-populations have been formed with updated copies of all neighbors. After all GAN training is completed and all models are evaluated again, the least fit generator and discriminator in the sub-populations are replaced with the fittest ones.

Fitness Evaluation. Evolutionary algorithms (EAs) are black box algorithms capable of solving problems viewed in terms of their inputs and outputs, without any knowledge of internal workings. In `Lipizzaner`, GAN evaluation is a black box component because the algorithm can query an opaque module with some inputs and get back outputs that describe performance. We make this point to emphasize the generality of the hybridization between GANs and a competitive evolutionary algorithm. *Any* GAN model evaluation component could work within `Lipizzaner`. We implement *particular* GAN examples, without loss of generality.

Integral to the sub-populations approach in `Lipizzaner` is the notion that a model's performance depends on its adversary. Calculated with some loss function M, performance can be expected to vary from discriminator to discriminator. To accommodate this, in `Lipizzaner`, fitness $\mathcal{L}$ of a model ($g_i \in \mathbf{g}$ or $d_j \in \mathbf{d}$) is based on some solution concept [49], f_s, that can be its best, worst or average performance against all its adversaries.

Algorithm 2 `CoevolveAndTrainModels`: Select a new sub-population from the current one. Assign all models a loss function at random. Each mini-batch train discriminators against a randomly drawn generator and generators against a randomly drawn discriminator, using SGD. Evaluate all against each other, using the solution concept, $f_s^{\min}$, of minimum loss as value to choose best to replace worst and update center. Return this new neighborhood.
Input: τ : Tournament size, X : Input training dataset, β : Mutation probability, n : Cell neighborhood sub-population, M : Loss functions, f_s Solution concept
Return: n : Cell neighborhood sub-population

1: $\mathbf{B} \leftarrow$ getMiniBatches(X) ▷ Load minibatches
2: $\mathcal{L} \leftarrow$ `EvaluateGANPairs`(n, $f_s^{\min}$) ▷ Evaluate all updated GAN pairs, Alg. 3
3: $\mathbf{g}, \mathbf{d} \leftarrow$ tournamentSelect(n, τ) ▷ Select using loss($\mathcal{L}$) as fitness
4: **for** $B \in \mathbf{B}$ **do** ▷ Loop over batches
5: $n_\delta \leftarrow$ mutateLearningRate(n_δ, β) ▷ Update neighborhood learning rate with with Gaussian mutation
6: $d \leftarrow$ getRandomOpponent($\mathbf{d}$) ▷ Get uniform random discriminator
7: **for** $g \in \mathbf{g}$ **do** ▷ Evaluate generators and train with SGD
8: $g, d \leftarrow$ pick_random(M, g, d) ▷ Random draw of loss function for each generator and discriminator
9: $\nabla_g \leftarrow$ computeGradient(g, d) ▷ Compute gradient for neighborhood center
10: $g \leftarrow$ updateNN(g, ∇_g, B) ▷ Update with gradient
11: $g \leftarrow$ getRandomOpponent($\mathbf{g}$) ▷ Get uniform random generator
12: **for** $d \in \mathbf{d}$ **do** ▷ Evaluate discriminator and train with SGD
13: $g, d \leftarrow$ pick_random(M, g, d) ▷ Random draw of loss function for each generator and discriminator
14: $\nabla_d \leftarrow$ computeGradient(d, g) ▷ Compute gradient for neighborhood center
15: $d \leftarrow$ updateNN(d, ∇_d, B) ▷ Update with gradient
16: $\mathcal{L} \leftarrow$ `EvaluateGANPairs`(n, $f_s^{\min}$) ▷ Evaluate all updated GAN pairs, Alg. 3
17: $n \leftarrow$ replace(n, $\mathbf{g}$) ▷ Replace the generator with worst loss
18: $n \leftarrow$ replace(n, $\mathbf{d}$) ▷ Replace the discriminator worst loss
19: $n \leftarrow$ setCenterIndividuals(n) ▷ Best generator and discriminator are placed in the center
20: **return** n

Algorithm 3 `EvaluateGANPairs`: Evaluate all models against each other using a solution concept
Input: $\mathbf{B}$: Minibatches, n : Cell neighborhood sub-population, f_s Solution concept
Return: $\mathcal{L}$: Generator and discriminator losses

1: $B_r \leftarrow$ getRandomMiniBatch($\mathbf{B}$) ▷ Get random minibatch
2: **for** $g, d \in \mathbf{g} \times \mathbf{d}$ **do** ▷ Evaluate all GAN pairs
3: $\mathcal{L}_{g,d} \leftarrow$ evaluate(g, d, B_r) ▷ Evaluate GAN
4: $\mathcal{L}_g \leftarrow f_s^{\min}(\mathcal{L}_{\cdot,d})$ ▷ Fitness for generator is the minimum loss value ($\mathcal{L}$)
5: $\mathcal{L}_d \leftarrow f_s^{\min}(\mathcal{L}_{g,\cdot})$ ▷ Fitness for discriminator is the minimum loss value ($\mathcal{L}$)
6: **return** $\mathcal{L}$

GAN training. In `Lipizzaner` GAN training is also a black box component because, as long as model parameters are updated and a learning rate can be specified parametrically, the coevolutionary algorithm is oblivious to how they are updated or used.

In this version of `Lipizzaner`, we step into the black box to deliberately randomize and vary the loss function of the training at the cell level. This will, in part address, the dilemma of not knowing what function is ideal and being forced into only choosing one. Another positive, secondary effect, of multiple loss functions could be that they promote better diversity in the trained models samples. The hypothesis is that when all models always use the same loss function, there is a risk that `Lipizzaner` will converge prematurely. Therefore we set up three loss functions with corresponding optimization objectives and randomly assign one to a model after selection. They are (written for generators):

1) Binary cross entropy (BCE) loss where the model's objective is to minimize the Jensen–Shannon divergence (JSD) between the real(p) and fake(q) data distributions, i.e., $JSD(p \parallel q)$.

$$M_G^{BCE} = \frac{1}{2}\,\mathbb{E}_{x\sim G_g}[log(1 - D_d(x))] \tag{4}$$

2) Mean squared error (MSE) loss where the model's objective is to minimize the average of the squares of the errors, e.g. [41].

$$M_G^{MSE} = \mathbb{E}_{x\sim G_g}[log(D_d(x) - 1)^2] \tag{5}$$

3) Heuristic loss (HEU) objective maximizes the probability of the discriminator being mistaken by minimizing the objective function [64]:

$$M_G^{HEU} = \frac{1}{2}\,\mathbb{E}_{x\sim G_g}[log(D_d(x))] \tag{6}$$

This objective is equal to minimizing $[KL(p \parallel q) - 2JSD(p \parallel q)]$

In this study, we also investigate *Wasserstein GANs* [4] that minimizes the *Wasserstein Distance* between the two probability distributions. However, it is practically intractable. Thus, this method optimizes *Wasserstein (WASS) loss* objective.

$$M_G^{WASS} = \mathbb{E}_{x\sim G_g}[f_w(x)] \tag{7}$$

Where f_w comes from a family of *K-Lipschitz continuous functions* parameterized by w [4].

Evolving Generator Mixture Weights. `Lipizzaner` at the highest level searches for and returns a mixture of generators composed from a neighborhood. It evolves a mixture weight vector $\boldsymbol{\omega}$ for each neighborhood using an ES-(1+1) algorithm [39, Algorithm 2.1] which optimizes for generator performance. The s-dimensional mixture weight vector $\boldsymbol{\omega}$ is defined as follows

$$\mathrm{g}^*, \boldsymbol{\omega}^* = \underset{\mathrm{g}^k, \boldsymbol{\omega}^k : 1\le k\le m^2}{\arg\max}\; g\Big(\sum_{g_i\in \mathrm{g}^k \omega_i\in\boldsymbol{\omega}^k} \omega_i G_{g_i}\Big)\,, \tag{8}$$

where ω_i represents the probability that a data point comes from the ith generator in the neighborhood, with $\sum_{\omega_i\in\boldsymbol{\omega}^k} \omega_i = 1$.

4.3 Discussion of Algorithmic Design and Implementation

`Lipizzaner` is open source MIT licensed. It is written in Python and uses PyTorch. The source is available from https://github.com/ALFA-group/lipizzaner-gan.

Communication. The only active communication in `Lipizzaner` is each cell's read of its four adjacent cells to form the model sub-populations at the start of each generation. There is no explicit inter-cell synchronization. This makes the software convenient to deploy on heterogenous hardware and training that may exhibit varying execution times. Passive communication occurs when each cell, as center, updates itself, thus allowing the cells which consider it to be their neighbor to access its new state. Communication injects diversity to the degree it propagates new local results to other cells.

Complexity Analysis. Using the notation in Table 1, the computational cost of `Lipizzaner` is the product of costs at the cell level and the number of cells in the grid, N and the number of generations T. Cell level cost can be decomposed into fitness evaluations before and after training, GAN training itself per generation and measurement score calculation when evaluating the performance of the mixture weights. Other computation at the cell is insignificant. The cost of fitness evaluations before `Algorithm 2` and after training is the twice the product of evaluating the loss of every model against its adversary. This entails s^2 loss calculations. Batch training iterates to use the entire

dataset, X, once. Each iteration there are $2s$ GAN trainings. Each GAN training cost C entails back propagation to obtain gradient information and network parameter updates.

The computational complexity of cell level effort is dominated by GAN training. Therefore the computation complexity of `Lipizzaner` is $O(s^2NTCe)$. If we consider the Moore neighborhood size a constant, $s = 5$, the complexity reduces to $O(NTCe)$. We note that the quadratic factor in coevolutionary algorithm complexity arising from all population members being competed against their adversaries is reduced to a linear factor in `Lipizzaner` due to the use of neighborhoods.

Algorithm 4 `MixtureEA`: Evolve mixture weights ω with a 1+1 Evolutionary Strategy.
Input: μ : Mutation rate, n : Cell neighborhood sub-population, ω : Mixture weights
Return: ω : mixture weights

1: $\omega' \leftarrow$ mutate(ω, μ) ▷ Gaussian mutation of mixture weights
2: $\omega'_f \leftarrow$ evaluateMixture(ω', n) ▷ Evaluate generator mixture score, e.g. FID for images
3: **if** $\omega'_f < \omega_f$ **then** ▷ Replace if new mixture weights are better
4: $\omega \leftarrow \omega'$ ▷ Update mixture weights
5: **return** ω

5 EMPIRICAL EXPERIMENTS AND RESULTS

We now address the two research questions stated in Section 1. Section 5.1 describes our general experimental configurations. Section 5.2 details the ablation experiments to address **RQ-1**: whether each of the different features of `Lipizzaner`'s evolutionary computation methodology are needed? Section 5.3 presents experiments with different loss functions to address **RQ-2**: Do different loss function combinations result in models with better performance and higher diversity of solutions and model parameters?

5.1 Experimental Configurations

Experiment GAN and coevolutionary algorithm parameter settings are shown in Table 2. `E-GAN` and `GAN-BCE` both use the Adam optimizer with an initial learning rate (0.0002). We follow [64] to set up the remaining parameters of `E-GAN`. All methods have been implemented in `Python3` and `pytorch`[1]. The experiments are performed on a cloud that provides 8 Intel Xeon cores 2.2GHz with 32 GB RAM and a NVIDIA Tesla T4 GPU with 16 GB RAM. All implementations use the same `Python` libraries and versions to minimize computational differences that could arise from using the cloud. We consider the variances that could arise to be insignificant.

We use two common image datasets from the GAN literature: MNIST [33] and CelebA. MNIST has been widely used and is especially appropriate to investigate mode collapse due to its limited target space (namely the characters 0-9). The larger CelebA dataset [38] contains 200,000 images of faces. To obtain an absolute measure of model performance, we draw fake image samples from the generators and score them with Frechet inception distance (FID) [24]. FID score is a black box, discriminator-independent, measure that expresses image similarity to the samples used in training.

For `Lipizzaner` we also investigate the diversity of the evolved networks to see whether one network is replicated over the entire grid. One measure of the spatial topology property is the max neighbourhood distance ($\Delta_n^{\max}$). Grids smaller than 4×4 always have neighborhood overlap, i.e. the $\Delta_n^{\max}$is 0. For 4×4 the $\Delta_n^{\max}$is 1, $(4-3)$, and for 5×5 the $\Delta_n^{\max}$is 2 $(5-3)$.

[1]Pytorch Website - `https://pytorch.es/`

Table 2. Setup for experiments conducted with the Lipizzaner system on MNIST and CelebA datasets.

Parameter	MNIST	CelebA
Coevolutionary settings		
Generations (epochs)	200	20
Population size per cell	1	1
Tournament size	2	2
Grid size	See Table 6	
Mixture mutation scale	0.01	0.05
Hyperparameter mutation		
Optimizer	Adam	Adam
Initial learning rate	0.0002	0.00005
Mutation rate	0.0001	0.0001
Mutation probability	0.5	0.5
Network topology		
Network type	MLP	DCGAN
Input neurons	64	100
Number of hidden layers	2	4
Neurons per hidden layer	256	$16,384 - 131,072$
Output neurons	784	$64 \times 64 \times 3$
Activation function	$tanh$	$tanh$
Training settings		
Batch size	100	128

Table 3. Lipizzaner feature ablation. Columns indicate ✓if the ablation is using the feature. Row indicate the different ablations.

Ablation	Communication	Selection	Ensemble	Learning Rate Mutation
Baseline	✓	✓	✓	✓
`Learning Rate Mutation`	✓	✓	✓	
`Ensemble Weight Optimization`	✓	✓		✓
`Local Selection`	✓		✓	✓
`Selection and Communication`			✓	✓
`All But Ensemble Weight Optimization`			✓	

5.2 Ablation Experiments

The goal of the **RQ-1** and experiments in this section is to establish the relevance of Lipizzaner's features . We proceed by means of ablation. In Section 5.2.1 we outline our ablation setups and in Section 5.2.2 we show and discuss the results.

5.2.1 Ablation Setup. We conduct the following ablation experiments, all using BCE as the loss function for both the generator and discriminator. The baseline is Lipizzaner with all features present. The ablations are listed in Table 3. They consist of single ablations for each of selection, ensemble weight optimization, and learning rate mutation. Eliminating communication only is meaningless, therefore we ablate both selection and communication together. Finally, we ablate all but ensemble weight optimization to gauge its standalone merit.

The ablations, see Table 3, are:

- `Learning Rate Mutation`: Lipizzaner without evolutionary mutation of the initial value of the Adam optimizer learning rate.
- `Local Selection`: Lipizzaner without selection at the local neighborhood scale.
- `Ensemble Weight Optimization`: Lipizzaner without ensemble weight optimization though still with an ensemble solution. The generative model returned is defined by the five generators in the most competitive neighborhood with the same mixture weights (i.e. ω_i=0.2).
- `Selection and Communication`: Lipizzaner without selection and communication but using learning rate mutation and ensemble weight optimization. We run Lipizzaner 1×1

Table 4. FID results for Lipizzaner ablation experiments on *MNIST*-4×4 given fixed number(200) of generations. The baseline, where all features are present, has the statistically significant lowest (best) FID score, minimum, median and mean.

Ablation	Mean	Std	Median	IQR	Min	Max
Baseline: all features present	29.11	5.20	29.41	6.51	19.56	40.42
Learning Rate Mutation	45.64	13.65	41.43	9.88	30.75	99.23
Local Selection	37.97	8.89	35.98	13.61	23.69	56.36
Ensemble Weight Optimization	36.57	5.41	35.90	3.91	26.35	54.60
Selection and Communication	35.94	2.72	36.30	3.80	28.03	39.32
All but Ensemble Weight Optimization	49.44	4.08	43.13	6.27	49.16	58.50

(i.e., 1×1-DCGAN) 30 times with learning rate mutation. Then, we randomly select subsets of five generators to create mixtures by applying ensemble weight optimization.

- All But Ensemble Weight Optimization: Our goal is to approximate Lipizzaner executing with random search. Therefore we ablate selection, communication, and learning rate mutation but use ensemble weight optimization. Thus, we train 30 GANs (DCGAN) individually. Then, we randomly select subsets of five generators to create mixtures by applying ensemble weight optimization.

We choose the 4×4 grid size to have minimal overlap of the neighborhoods to isolate the effects of communication component in Lipizzaner. We measure the importance of the different variants using FID score.

5.2.2 Ablation Results. The results from the ablation experiments on *MNIST*-4×4 are shown in Table 4. For any single ablation the experimental average mean, median, minimum, and maximum FID score are worse than the baseline. Moreover, the baseline is statistically better than all the other methods according to pairwise Wilcoxon Rank Sum statistical test ($\alpha < 0.01$).

As would be expected, the impact of ablating mutation of the initial value of the Adam learning rate is the most severe. Both GAN and network model training convergence and performance are extremely sensitive to initial conditions. When both selection and communication are eliminated, while learning rate mutation and ensemble weight optimization remain, we obtain an average mean FID score which is closer to the baseline than any single ablation. Finally, when we retain only ensemble weight optimization, bottom row of Table 4, we can see the importance of the learning rate mutation in providing better results. These findings verify the importance of each of the components in Lipizzaner for its performance.

5.3 Using Different Loss Functions

RQ-2 asks whether different loss function combinations result in models with better performance and more diversity. This provides guidance as to whether developers, lacking knowledge of the ideal loss function to train with, should try different Lipizzaner combinations. In Section 5.3.1 we set up pairwise combinations of generator and discriminator loss functions that are used at every cell throughout the evolutionary run. We also experiment with randomizing the combination used each generation by any cell. In Section 5.3.2 we compare the results and in Section 5.3.3 we check for diversity in the trained model parameters and in the distribution estimated by the best trained ensemble.

5.3.1 Experimental Setup. We configure different GAN training setups. The baselines are not spatially distributed (WGAN, GAN-BCE, and E-GAN). WGAN and GAN-BCE are also the only combinations without a population, see [13] for a comprehensive analysis of GANs used on MNIST. The Lipizzaner variants use either the same loss functions (see Section 4.2) for generator-discriminator for all training cycles (BCE-BCE, MSE-BCE, MSE-MSE, HEU-HEU, WASS-WASS) or not (RAND-COMBOS).

Table 5. Comparative setups using Lipizzaner's ability to train with different loss function combinations and the loss functions used generator and deiscriminator.

Name	Spatial Distribution	Loss Function Diversity	Generator	Discriminator
Baselines				
GAN-BCE			BCE	BCE
WGAN [4]			WASS	WASS
E-GAN [64]		✓	BCE, MSE, HEU	BCE
Loss Function Combinations				
BCE-BCE	✓		BCE	BCE
MSE-BCE	✓		MSE	BCE
MSE-MSE	✓		MSE	MSE
HEU-HEU	✓		HEU	BCE
WASS-WASS	✓		WASS	WASS
RAND-COMBOS	✓	✓	BCE, MSE, HEU	BCE, MSE

Table 6. Lipizzaner grid sizes used for the experiments. Sizes were selected based on a facet-wise approach. The name indicates which loss functions (Section 4.2) that were used for generator and discriminator.

	MNIST			*CelebA*
Name	3×3	4×4	5×5	2×2
BCE-BCE	✓	✓	✓	✓
MSE-BCE	✓			
MSE-MSE	✓	✓		✓
HEU-HEU	✓	✓		✓
WASS-WASS		✓		
RAND-COMBOS	✓	✓	✓	✓

The setups are described in Table 5. Note that the combinations extend [62] which did not randomize the loss function of discriminator and instead always used BCE for the discriminator. E-GAN [64] which we describe in Section 2 is also included.

To compare the impact of different loss functions, we execute all methods with a computational budget of nine hours. The spatial coevolutionary methods of RAND-COMBOS, BCE-BCE, MSE-MSE, and HEU-HEU use a grid size of 3×3, and are able to train nine networks in parallel. Thus, they are simultaneously executed for one hour.

In addition, we also further investigated different grid sizes. Table 6 shows what size grids were investigated for each of the different methods. Computational costs do not allow a full grid comparison so we isolate specific combinations and sizes.

We quantify the distributional coverage of the trained ensemble's fake images in two ways. Per [35] we report their Total Variation Distance (TVD):

$$TVD = \frac{1}{2} \sum_{l \in L} | \, \|X_l\| - \|d(z)_l\| \, | \tag{9}$$

where L are the labels (classes), $\|X_l\|$ the frequency (number) of labels in the data (X), and $\|d(z)_l\|$ the frequency (number) of labels in the generated data ($d(z)$).

We also visually inspect the appearances of fake images for each class of the dataset, noting that this method depends on a dataset with classes. Finally, to assess whether the different combinations result in different networks, we calculate the L_2 distance between network parameters.

To gauge the benefits of the different variants, if any, we execute 30 independent runs of each method in Table 5 on the MNIST problem with equivalent computational budgets. At the end of each run, we report the performance of the best performing mixture model, see Table 7 as FID score where lower is better.

5.3.2 Loss Function Combination Results. Preliminarily, the benefit of a population over a single GAN is discernible. The FID scores of GAN-BCE, the single GAN trained for 9 hours, and E-GAN,

where a single GAN is replicated and trained with three loss functions before selection, are surpassed by all of Lipizzaner's population-based combinations. GAN-BCE's mean FID score is approximately 10 times higher (worse) and its lowest/best FID score is significantly higher (worse), approximately 10 times. These differences are statistically significant according to a wilcoxon ranksum test. All of the Lipizzaner methods perform statistically better than E-GAN, GAN-BCE and WGAN. Focusing on GAN-BCE and BCE-BCE which train with the same loss function pair, the advantage of distributed spatial evolution is clearly apparent. They have comparable FID scores during the first 30 training epochs, however BCE-BCE improves in performance over the remaining epochs (not shown).

We note that the mean and median FID scores of GAN-BCE are better than E-GAN and E-GAN's performance demonstrates a larger variance of FID scores compared to GAN-BCE. It should however be noted that, in the original paper [64] which used a computational budget of 30h compared to the 9h we use here, E-GAN performance keeps improving. The results provided by Lipizzaner variants that generate diversity only by a spatial population (BCE-BCE, HEU-HEU, and MSE-MSE) are significantly more competitive than E-GAN, which experiments with loss function diversity but does not evolve a population from one generation to the next. Note that the performance of E-GAN compared to Lipizzaner provided basis for our decision to focus our computational resources on investigating Lipizzaner.

The experiments of Table 7, along with Table 8 which shows FID scores for different grid sizes, allow us next to answer whether explicitly promoting diversity with RAND-COMBOS makes it superior among the other Lipizzaner variants in terms of performance. Recall that they use different, *static*, pairwise combinations of loss functions for gradient-based GAN training. While we did not exhaustively try every combination of static pairs, it is arguably possible that one static pairing will surpass RAND-COMBOS because randomization introduces noise. We do not observe this in most measures in Table 7. An exception is RAND-COMBOS's IQR of the which is outscored by BCE-BCE. A Wicoxon ranksum test with Holm correction confirms that the difference between RAND-COMBOS and the other methods for *MNIST-3×3* is significant at confidence levels between α=0.001 and α=0.05. Across three grid sizes, per Table 8, rank ordering changes among the methods however, again, no statistically superior method emerges. However, because RAND-COMBOS both explicitly promotes diversity by randomly assigning loss functions *dynamically*, i.e. before each training batch and it avoids the necessity of exhaustive, static pairwise investigation, it is arguably a better choice.

A secondary question is how efficiently each method uses the same computational budget of 9h. Lipizzaner variants, including RAND-COMBOS execute asynchronously and in parallel for all the cells. There is no bottleneck between cells. E-GAN has a synchronization bottleneck because the three copies of the generator are trained sequentially with a single copy of the discriminator. Because Lipizzaner variants execute with asynchronous parallelism, the number of training epochs performed by each cell of the grid in the same run varies. The number of epochs are normally distributed for all the algorithms (not shown). As expected GAN-BCE, a serially trained, single GAN trains for the most epochs. It performs about two times the number of generations of E-GAN, which evaluates three generator networks each generation. The spatially distributed coevolutionary algorithms performed significantly fewer training epochs than E-GAN. However, during each epoch for 3×3 these methods evaluate 45 GANs, i.e., neighborhood size of 5 on 9 cells, which is 15 times more generator and discriminator networks than E-GAN.

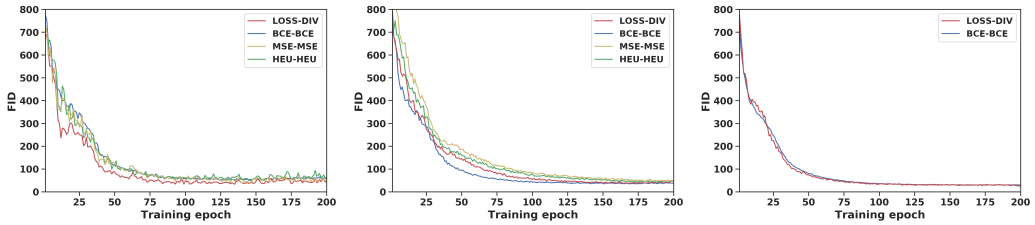
The best FID for the different grid sizes and loss functions are shown in Table 8. As expected the larger grid sizes impact the FID more than the use of loss functions. RAND-COMBOS has consistently a good performance and low variance for the different grid sizes, i.e. it makes good use of the increase in population size (and implicitly the number of fitness evaluations) and max neighborhour distance, $\Delta_n^{\max}$. Note that we only use the two best Lipizzaner versions for 5×5.

Table 7. FID values for 30 independent runs of *MNIST-3×3* 9h compute time for different GAN training methods. The lowest value in each column is indicated by the highlighted (yellow) cell. RAND-COMBOS has the significantly best performance. (**IQR** is the interquartile range)

	Mean	Std	Median	IQR	Min	Max
E-GAN [64]	466.11	48.89	481.61	69.33	277.47	504.86
GAN-BCE [19]	448.31	28.69	453.53	21.69	326.11	474.56
WGAN [4]	180.89	104.97	161.48	13.22	146.05	733.87
BCE-BCE	48.96	10.00	46.07	4.66	38.01	80.88
MSE-MSE	52.30	8.89	54.13	14.98	37.51	65.09
HEU-HEU	52.52	9.20	52.73	9.77	34.87	72.43
RAND-COMBOS	42.24	5.53	43.18	7.59	30.77	48.98

Table 8. FID results for *MNIST* experiments with different grid sizes for Lipizzaner variants given fixed number(200) of generations. RAND-COMBOS has consistently a good performance and low variance.

	Mean	Std	Median	IQR	Min	Max
MNIST-3×3 (Population 9)						
RAND-COMBOS	39.51	5.24	39.48	8.54	25.40	48.13
BCE-BCE	40.78	7.74	40.27	12.59	26.68	58.76
MSE-MSE	49.79	8.65	50.76	8.12	35.59	66.96
HEU-HEU	43.26	7.22	42.33	5.05	30.88	60.40
MNIST-4×4 (Population 16)						
RAND-COMBOS	30.29	3.71	30.46	4.13	21.23	38.29
BCE-BCE	29.11	5.20	29.41	6.51	19.56	40.42
MSE-MSE	29.14	4.32	28.98	6.36	22.31	37.45
HEU-HEU	30.28	8.32	31.14	13.04	18.57	55.22
WASS-WASS	148.36	7.92	150.47	9.97	127.86	161.47
MNIST-5×5 (Population 25)						
RAND-COMBOS	26.78	2.47	27.65	2.03	21.85	29.46
BCE-BCE	28.16	4.05	28.13	5.81	22.56	40.01



(a) MNIST 3×3 (population 9) (b) MNIST 4×4 (population 16) (c) MNIST 5×5 (population 25)

Fig. 4. Results on MNIST dataset for different grid sizes. Median FID (y-axis) evolution through the epochs (X-axis), each line is a different Lipizzaner variant.

From these results we observe: *A)* WGAN improves GAN-BCE. *B)* WASS-WASS improves WGAN. *C)* WASS-WASS does not improve Lipizzaner. This can be due to observations from the literature [34] that reported that MNIST WGAN is less stable than GAN-BCE as can be seen by the Std in Table 7. Note, WASS-WASS is not included in the FID boxplot to avoid losing information due to the differences in scale.

The median FID over each training epoch is shown in Figure 4. The FID values are smoother the larger the population size. Note that the total number of fitness evaluations (FE) are different for each grid size, 3×3 has 1,800 FE, 4×4 has 3,200 FE, and 5×5 has 5,000 FE. When looking at the FIDs for the different grid sizes we see that the median values improvement is correlated to the increase in number of fitness evaluations.

Table 9. Results on CelebA dataset on a 2×2. FID results in terms of best mean, normalized standard deviation, median and interquartile range (IQR). (Low FID indicates good performance)

	Mean	Std	Median	IQR	Min	Max
RAND-COMBOS	36.17	0.78	35.83	0.97	35.35	37.29
BCE-BCE	36.18	1.93	35.39	1.64	34.64	39.75
MSE-BCE	163.11	118.34	123.13	191.54	55.41	310.79
HEU-HEU	38.35	2.76	39.17	4.50	34.94	40.93

Table 10. TVD measurements on MNIST dataset. Each row shows mean (± std) TVD values, low TVD indicates more diversity, for different grid sizes. The columns show the Lipizzaner variants. RAND-COMBOS consistently has the lowest TVD value.

Experiment	RAND-COMBOS	BCE-BCE	HEU-HEU	MSE-MSE	WASS-WASS
MNIST-3×3	0.122 ± 0.016	0.138 ± 0.028	0.144 ± 0.034	0.145 ± 0.032	NA
MNIST-4×4	0.121 ± 0.021	0.124 ± 0.028	0.128 ± 0.031	0.131 ± 0.032	0.142 ± 0.018
MNIST-5×5	0.108 ± 0.022	0.118 ± 0.020	NA	NA	NA

Finally, we compare the spatially distributed coevolutionary Lipizzaner methods as applied to generate the CelebA dataset. We chose these methods since they had good performance on the MNIST dataset. Table 9 summarizes the results over 6 independent runs. RAND-COMBOS provides the lowest median FID and MSE-MSE the highest one. BCE-BCE and HEU-HEU provide median and average FID scores close to the RAND-COMBOS ones. However, RAND-COMBOS is more robust to the varying performance of the methods that apply a unique loss function (see Std and IQR in Table 9). Note that Lipizzaner does not achieve state-of-the-art CelebA performance with these experimental settings and GAN architecture, the point is to show the impact of Lipizzaner variants.

In summary, generally these results indicate that spatially distributed coevolutionary training is the best choice to train GANs, even when there is no knowledge about the best loss function to the problem. RAND-COMBOS, the best method for promoting diversity, is best at utilizing the increase in fitness evaluations through the increase of grid size to improve the FID.

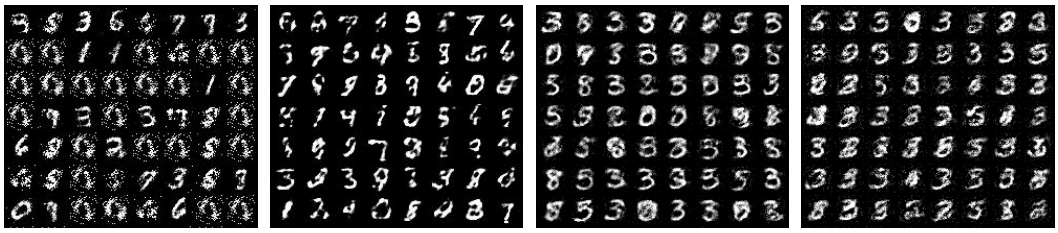
5.3.3 Diversity through Different Loss Function Combinations. We start by examining the extent of diversity in different methods. We contrast the Lipizzaner variants that generate diversity by a spatial population (BCE-BCE, HEU-HEU, and MSE-MSE) to E-GAN which explicitly promotes additional diversity by loss function randomization each epoch. We use two measures of *output* diversity, neither better than the other, simply different in what characteristic of diversity they tally: TVD (lower is better) and class balance. For MNIST, ideally each digit class is represented with the same frequency in the fake image samples. Additionally, we use a measure of *model* diversity where we look at network parameters from one cell to the others.

In considering diversity in a set of generator-derived output samples (images in this case), our hypothesis is that the RAND-COMBOS will exhibit higher output diversity than the other Lipizzaner methods because the randomization of the training loss function impacts the convergence rate of training. Figure 5 shows a random selection of generator sample fake images for each method. Each Lipizzaner method, a spatially distributed coevolutionary algorithm, is able to produce a mixture that generates what visually appears to be well-balanced and accurate looking digits.

The total variation distance (TVD) for each algorithm [35] is shown in Table 10 and Figure 6. Decrease in TVD values, i.e. diversity, increases with the grid size for each method. Moreover, RAND-COMBOS consistently has the lowest TVD value, i.e. diversity of the generative model at every grid size vs the other Lipizzaner methods that do not promote diversity through loss function randomization.

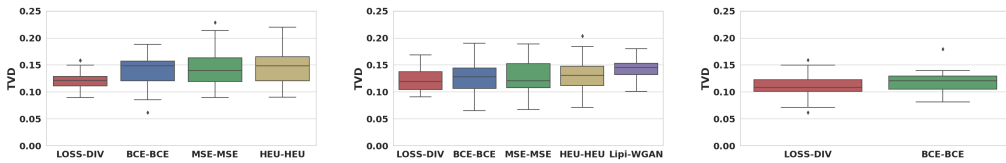
The distributions of each digit class for generated samples show that HEU-HEU and RAND-COMBOS generate samples spanning across different classes. The two methods (E-GAN, GAN-BCE) that do not

(a) Mode collapse (b) RAND-COMBOS (c) BCE-BCE (d) HEU-HEU



(e) MSE-BCE (f) WASS-WASS (g) E-GAN (h) GAN-BCE

Fig. 5. Sequence of samples generated of MNIST dataset. (a) mode collapse, the generator is focused on the character 0. It illustrates samples generated by the best generator (in terms of FID) by RAND-COMBOS (b), BCE-BCE (c), HEU-HEU (d), MSE-MSE (e), WASS-WASS (f), E-GAN (g), and GAN-BCE (h).



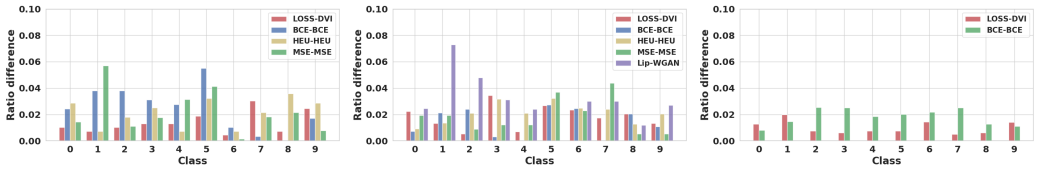
(a) MNIST 3×3 (b) MNIST 4×4 (c) MNIST 5×5

Fig. 6. Boxplots of TVD results on MNIST dataset for different grid sizes (Low TVD indicates more diversity). TVD is on Y-axis and Lipizzaner variant on Y-axis.

promote diversity via spatial distribution suffer from what is possibly mode collapse. About half of the samples are the digit 3 and they do not generate samples of digits 4 and 7.

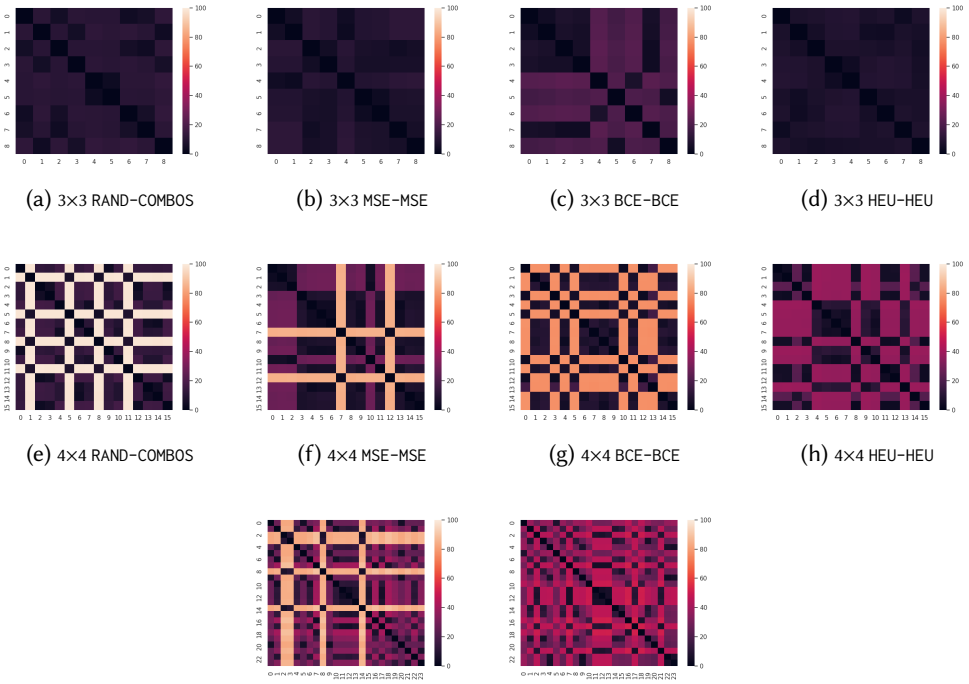
We next measure the output diversity of generators trained with MNIST by measuring the class distributions. Figure 7 shows the absolute value of the difference between the proportion of the generated samples and the ideal proportion (10%) for each class. The difference in original and generated class distribution is decreases when the size of the grid increases, implying that spatial distribution contributes to diversity. RAND-COMBOS, which explicitly adds an additional means of promoting diversity, consistently has the lowest absolute difference, i.e. of all methods, it best replicates the 10% frequency of images per class in the original distribution.

Finally, we investigate the diversity of the evolved networks to see whether one network is replicated over the entire grid and neighborhoods. We hypothesis that the max neighborhour distance, $\Delta_n^{\max}$, property impacts the information exchange between the neighborhood sub-populations and manifests itself in the diversity of the population's network parameters. The result for generators



(a) MNIST 3×3 (b) MNIST 4×4 (c) MNIST 5×5

Fig. 7. Absolute per class value (x-axis) of the difference between the ratio of the generated distribution and the ratio of the ideal distribution (y-axis), the ideal ratio is 10%. Color bar shows `Lipizzaner` variant.



(a) 3×3 RAND-COMBOS (b) 3×3 MSE-MSE (c) 3×3 BCE-BCE (d) 3×3 HEU-HEU

(e) 4×4 RAND-COMBOS (f) 4×4 MSE-MSE (g) 4×4 BCE-BCE (h) 4×4 HEU-HEU

(i) 5×5 RAND-COMBOS (j) 5×5 BCE-BCE

Fig. 8. Diversity of of population in *genome* space. Heatmap of L_2 distance between the generators in the population on MNIST at the final generation. X-axis and Y-axis show each grid cell, and dark indicates small distance.

shows that there is no significant replication of networks across the generator population as shown in Figure 8. We use the L_2 distance to measure the distance between neural network parameters, this is a rough measure that provides insight whether neural networks are identical or not. The diversity fluctuates across the setups and methods. In Table 11 we can observe that the average diversity increases when $\Delta_n^{\max}$is greater than 0. This is likely due to the increase the propagation time for the information exchange. In addition, RAND-COMBOS has the largest diversity variance for 4×4 and 5×5, i.e. there are some networks that are more diverse than others.

Table 11. Results on MNIST dataset. Mean (± std) distance values between the weights of the networks in different cells, high values indicates more networks diversity in the grid. RAND-COMBOS consistently has the highest distance value.

Experiment	RAND-COMBOS	BCE-BCE	HEU-HEU	MSE-MSE
MNIST-3×3	14.003 ± 6.659	13.850 ± 8.291	6.426 ± 2.739	7.320 ± 3.565
MNIST-4×4	43.681 ± 42.579	40.101 ± 34.917	23.260 ± 15.083	29.884 ± 30.588
MNIST-5×5	40.227 ± 29.700	31.984 ± 15.313	NA	NA

In summary, spatial grids with populations mapped across them exhibit diversity. This diversity can be increased by a supplementary method that randomizes loss functions used for training. The increase is due to the change in loss function impacting the training convergence of model parameters.

6 CONCLUSION & FUTURE WORK

In this paper, we have introduced a scalable, robust, distributed GAN training framework called Lipizzaner. Lipizzaner combines the advantages of gradient-based optimization for GANs with those of coevolutionary systems, and allows scaling over a distributed spatial grid topology. A relatively small spatial grid is sufficient to overcome the common limitations of GANs and improves performance compared to other evolutionary GAN training methods in the literature, due to the spatial separation and asynchronous evaluation. Moreover, the performance can improve with increased grid size. The utility of the Lipizzaner components was demonstrated by an ablation study.

In addition, we investigated diversity in the context of Lipizzaner's spatial coevolutionary (in particular competitive) populations. We considered an explicit means to further promote diversity which is to probabilistically choose one of three loss functions each round of training. This method, called RAND-COMBOS was tested on the MNIST and CelebA data sets. It showed increased diversity across grid sizes in comparison to the other Lipizzaner methods. It also showed no appreciable loss in performance. Given it is often hard to know the loss function with which to train a GAN, RAND-COMBOS avoids an uninformed decision or exhaustive combinatoric experimentation.

The reported Lipizzaner CelebA FID score are not state-of-the-art GAN architectures [37, 63, 70]. However, Lipizzaner can support any type of GAN, so we plan to experiment with more heterogeneous cell configurations, i.e. architectures. Finally, other advancements in evolutionary algorithms that can improve the robustness of GAN training, e.g. temporal evolutionary training, will be considered.

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 MINECO and FEDER projects TIN2016-81766-REDT and TIN2017-88213-R and the Systems that Learn Initiative at MIT CSAIL.

REFERENCES

[1] Abdullah Al-Dujaili, Tom Schmiedlechner, Erik Hemberg, and Una-May O'Reilly. 2018. Towards distributed coevolutionary GANs, In AAAI 2018 Fall Symposium. *arXiv preprint arXiv:1807.08194*.

[2] Abdullah Al-Dujaili, Shashank Srikant, Erik Hemberg, and Una-May O'Reilly. 2018. On the Application of Danskin's Theorem to Derivative-Free Minimax Optimization. *Int. Workshop on Global Optimization* (2018).

[3] Martin Arjovsky and Léon Bottou. 2017. Towards principled methods for training generative adversarial networks. *arXiv preprint arXiv:1701.04862* (2017).

[4] Martin Arjovsky, Soumith Chintala, and Léon Bottou. 2017. Wasserstein Generative Adversarial Networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70 (ICML'17)*. JMLR.org, 214–223.

[5] Sanjeev Arora, Rong Ge, Yingyu Liang, Tengyu Ma, and Yi Zhang. 2017. Generalization and Equilibrium in Generative Adversarial Nets (GANs). *arXiv preprint arXiv:1703.00573* (2017).

[6] Sanjeev Arora, Andrej Risteski, and Yi Zhang. 2018. Do GANs learn the distribution? Some Theory and Empirics. In *International Conference on Learning Representations*. https://openreview.net/forum?id=BJehNfW0-

[7] Helio JC Barbosa. 1999. A coevolutionary genetic algorithm for constrained optimization. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, Vol. 3. IEEE, 1605–1611.

[8] Helio J. C. Barbosa. 1997. A Coevolutionary Genetic Algorithm for a Game Approach to Structural Optimization. In *ICGA*.

[9] Ali Borji. 2019. Pros and cons of GAN evaluation measures. *Computer Vision and Image Understanding* 179 (2019), 41–65.

[10] Jürgen Branke and Johanna Rosenbusch. 2008. New approaches to coevolutionary worst-case optimization. In *International Conference on Parallel Problem Solving from Nature*. Springer, 144–153.

[11] Jason Brownlee. 2019. *Generative Adversarial Networks with Python Deep Learning Generative Models for Image Synthesis and Image Translation*. Machine Learning Mastery.

[12] Tatjana Chavdarova and François Fleuret. 2018. SGAN: An Alternative Training of Generative Adversarial Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 9407–9415.

[13] Keyang Cheng, Rabia Tahir, Lubamba Kasangu Eric, and Maozhen Li. 2020. An analysis of generative adversarial networks and variants for image synthesis on MNIST dataset. *Multimedia Tools and Applications* (2020), 1–28.

[14] Soumith Chintala, Emily Denton, Martin Arjovsky, and Michael Mathieu. 2016. How to Train a GAN? Tips and tricks to make GANs work. https://github.com/soumith/ganhacks.

[15] Dave Cliff and Geoffrey F Miller. 1995. Tracking the red queen: Measurements of adaptive progress in co-evolutionary simulations. In *European Conference on Artificial Life*. Springer, 200–218.

[16] Victor Costa, Nuno Lourenço, and Penousal Machado. 2019. Coevolution of Generative Adversarial Networks. In *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)*. Springer, 473–487.

[17] Dario Floreano and Claudio Mattiussi. 2008. *Bio-inspired artificial intelligence: theories, methods, and technologies*. MIT press.

[18] Chi-Keong Goh and Kay Chen Tan. 2009. A Competitive-Cooperative Coevolutionary Paradigm for Dynamic Multiobjective Optimization. *IEEE Transactions on Evolutionary Computation* 13, 1 (Feb 2009), 103–127. https://doi.org/10.1109/TEVC.2008.920671

[19] Ian Goodfellow. 2016. NIPS 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160* (2016).

[20] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*. 2672–2680.

[21] Robin Harper. 2014. Evolving robocode tanks for Evo robocode. *Genetic Programming and Evolvable Machines* 15, 4 (2014), 403–431.

[22] Jeffrey W Herrmann. 1999. A genetic algorithm for minimax optimization problems. In *CEC*, Vol. 2. IEEE, 1099–1103.

[23] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, and Bernhard Nessler. 2017. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. *arXiv preprint arXiv:1706.08500* (2017).

[24] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, Günter Klambauer, and Sepp Hochreiter. 2017. GANs trained by a two time-scale update rule converge to a nash equilibrium. *arXiv preprint arXiv:1706.08500* 12, 1 (2017).

[25] W. Daniel Hillis. 1990. Co-evolving parasites improve simulated evolution as an optimization procedure. *Physica D: Nonlinear Phenomena* 42, 1 (1990), 228 – 234. https://doi.org/10.1016/0167-2789(90)90076-2

[26] Babak Hodjat, Erik Hemberg, Hormoz Shahrzad, and Una-May O'Reilly. 2014. Maintenance of a long running distributed genetic programming system for solving problems requiring big data. In *Genetic Programming Theory and Practice XI*. Springer, 65–83.

[27] Phil Husbands. 1994. Distributed coevolutionary genetic algorithms for multi-criteria and multi-constraint optimisation. In *AISB Workshop on Evolutionary Computing*. Springer, 150–165.

[28] Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, et al. 2017. Population based training of neural networks. *arXiv preprint arXiv:1711.09846* (2017).

[29] Mikkel T Jensen. 2003. A new look at solving minimax problems with coevolutionary genetic algorithms. In *Metaheuristics: computer decision-making*. Springer, 369–384.

[30] Daniel Jiwoong Im, He Ma, Chris Dongjoo Kim, and Graham Taylor. 2016. Generative Adversarial Parallelization. *arXiv preprint arXiv:1612.04021* (2016).

[31] Rodney W Johnson, Michael E Melich, Zbigniew Michalewicz, and Martin Schmidt. 2005. Coevolutionary optimization of fuzzy logic intelligence for strategic decision support. *IEEE Transactions on Evolutionary Computation* 9, 6 (Dec 2005), 682–694. https://doi.org/10.1109/TEVC.2005.856208

[32] Daniel Le Ly and Hod Lipson. 2014. Optimal Experiment Design for Coevolutionary Active Learning. *IEEE Transactions on Evolutionary Computation* 18, 3 (June 2014), 394–404. https://doi.org/10.1109/TEVC.2013.2281529

[33] Yann LeCun. 1998. The MNIST database of handwritten digits. *http://yann. lecun. com/exdb/mnist/* (1998).

[34] Timothée Lesort, Andrei Stoian, Jean-François Goudou, and David Filliat. 2019. Training discriminative models to evaluate generative ones. In *International Conference on Artificial Neural Networks*. Springer, 604–619.

[35] Chengtao Li, David Alvarez-Melis, Keyulu Xu, Stefanie Jegelka, and Suvrit Sra. 2017. Distributional Adversarial Networks. *arXiv preprint arXiv:1706.09549* (2017).

[36] Jerry Li, Aleksander Madry, John Peebles, and Ludwig Schmidt. 2017. Towards Understanding the Dynamics of Generative Adversarial Networks. *arXiv preprint arXiv:1706.09884* (2017).

[37] Chieh Hubert Lin, Chia-Che Chang, Yu-Sheng Chen, Da-Cheng Juan, Wei Wei, and Hwann-Tzong Chen. 2019. COCO-GAN: generation by parts via conditional coordinating. In *Proceedings of the IEEE International Conference on Computer Vision*. 4512–4521.

[38] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. 2015. Deep Learning Face Attributes in the Wild. In *Proceedings of International Conference on Computer Vision (ICCV)*.

[39] Ilya Loshchilov. 2013. *Surrogate-assisted evolutionary algorithms*. Ph.D. Dissertation. University Paris South Paris XI; National Institute for Research in Computer Science and Automatic-INRIA.

[40] Mario Lucic, Karol Kurach, Marcin Michalski, Sylvain Gelly, and Olivier Bousquet. 2018. Are gans created equal? a large-scale study. In *Advances in neural information processing systems*. 700–709.

[41] Xudong Mao, Qing Li, Haoran Xie, Raymond YK Lau, Zhen Wang, and Stephen Paul Smolley. 2017. Least squares generative adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision*. 2794–2802.

[42] L. Miguel Antonio and C. A. Coello Coello. 2018. Coevolutionary Multiobjective Evolutionary Algorithms: Survey of the State-of-the-Art. *IEEE Transactions on Evolutionary Computation* 22, 6 (Dec 2018), 851–865. https://doi.org/10.1109/TEVC.2017.2767023

[43] Melanie Mitchell. 2006. Coevolutionary Learning with Spatially Distributed Populations. *Computational Intelligence: Principles and Practice* (2006).

[44] Nick Moran and Jordan Pollack. 2018. Coevolutionary neural population models. In *Artificial Life Conference Proceedings*. MIT Press, 39–46.

[45] Gonçalo Mordido, Haojin Yang, and Christoph Meinel. 2018. Dropout-gan: Learning from a dynamic ensemble of discriminators. *arXiv preprint arXiv:1807.11346* (2018).

[46] Behnam Neyshabur, Srinadh Bhojanapalli, and Ayan Chakrabarti. 2017. Stabilizing GAN training with multiple random projections. *arXiv preprint arXiv:1705.07831* (2017).

[47] Tu Nguyen, Trung Le, Hung Vu, and Dinh Phung. 2017. Dual discriminator generative adversarial nets. In *Advances in Neural Information Processing Systems*. 2670–2680.

[48] Stefano Nolfi and Dario Floreano. 1998. Coevolving predator and prey robots: Do "arms races" arise in artificial evolution? *Artificial life* 4, 4 (1998), 311–335.

[49] Elena Popovici, Anthony Bucci, R Paul Wiegand, and Edwin D De Jong. 2012. Coevolutionary principles. In *Handbook of natural computing*. Springer, 987–1033.

[50] Xin Qiu, Jian-Xin Xu, Yinghao Xu, and Kay Chen Tan. 2017. A New Differential Evolution Algorithm for Minimax Optimization in Robust Design. *IEEE transactions on cybernetics* (2017).

[51] Alec Radford, Luke Metz, and Soumith Chintala. 2015. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. *arXiv preprint arXiv:1511.06434* (2015).

[52] Christopher D Rosin and Richard K Belew. 1997. New methods for competitive coevolution. *Evolutionary computation* 5, 1 (1997), 1–29.

[53] Tim Salimans, Jonathan Ho, Xi Chen, and Ilya Sutskever. 2017. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv:1703.03864* (2017).

[54] Tom Schmiedlechner. 2018. *Industrial-Scale Evolutionary Machine Learning*. Master's thesis. University of Applied Sciences Upper Austria.

[55] Tom Schmiedlechner, Ignavier Ng Zhi Yong, Abdullah Al-Dujaili, Erik Hemberg, and Una-May O'Reilly. 2018. 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. *arXiv preprint arXiv:1811.12843*.

[56] Konstantin Shmelkov, Cordelia Schmid, and Karteek Alahari. 2018. How good is my GAN?. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 213–229.

[57] Kenneth O. Stanley and Jeff Clune. 2017. Welcoming the Era of Deep Neuroevolution - Uber Engineering Blog. https://eng.uber.com/deep-neuroevolution/.

[58] Kenneth O Stanley, Jeff Clune, Joel Lehman, and Risto Miikkulainen. 2019. Designing neural networks through neuroevolution. *Nature Machine Intelligence* 1, 1 (2019), 24–35.

[59] Kenneth O Stanley and Risto Miikkulainen. 2002. Evolving neural networks through augmenting topologies. *Evolutionary computation* 10, 2 (2002), 99–127.

[60] Guy Tevet, Gavriel Habib, Vered Shwartz, and Jonathan Berant. 2018. Evaluating text gans as language models. *arXiv preprint arXiv:1810.12686* (2018).

[61] Ilya O Tolstikhin, Sylvain Gelly, Olivier Bousquet, Carl-Johann Simon-Gabriel, and Bernhard Schölkopf. 2017. Adagan: Boosting generative models. In *Advances in Neural Information Processing Systems*. 5430–5439.

[62] Jamal Toutouh, Erik Hemberg, and Una-May O'Reilly. 2019. Spatial Evolutionary Generative Adversarial Networks. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '19)*. Association for Computing Machinery, New York, NY, USA, 472–480. https://doi.org/10.1145/3321707.3321860

[63] Thomas Unterthiner, Bernhard Nessler, Calvin Seward, Günter Klambauer, Martin Heusel, Hubert Ramsauer, and Sepp Hochreiter. 2017. Coulomb GANs: Provably optimal nash equilibria via potential fields. *arXiv preprint arXiv:1708.08819* (2017).

[64] Chaoyue Wang, Chang Xu, Xin Yao, and Dacheng Tao. 2019. Evolutionary generative adversarial networks. *IEEE Transactions on Evolutionary Computation* (2019).

[65] Yaxing Wang, Lichao Zhang, and Joost van de Weijer. 2016. Ensembles of generative adversarial networks. *arXiv preprint arXiv:1612.00991* (2016).

[66] Richard A Watson and Jordan B Pollack. 2001. Coevolutionary dynamics in a minimal substrate. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*. Morgan Kaufmann Publishers Inc., 702–709.

[67] Daan Wierstra, Tom Schaul, Jan Peters, and Juergen Schmidhuber. 2008. Natural evolution strategies. In *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*. IEEE, 3381–3387.

[68] Nathan Williams and Melanie Mitchell. 2005. Investigating the success of spatial coevolution. In *Proceedings of the 7th annual conference on Genetic and evolutionary computation*. ACM, 523–530.

[69] Steven R Young, Derek C Rose, Travis Johnston, William T Heller, Thomas P Karnowski, Thomas E Potok, Robert M Patton, Gabriel Perdue, and Jonathan Miller. 2017. Evolving deep networks using hpc. In *Proceedings of the Machine Learning on HPC Environments*. ACM, 7.

[70] Han Zhang, Zizhao Zhang, Augustus Odena, and Honglak Lee. 2019. Consistency Regularization for Generative Adversarial Networks. *arXiv preprint arXiv:1910.12027* (2019).

[71] Xin-Yuan Zhang, Yue-Jiao Gong, Ying Lin, Jie Zhang, Sam Kwong, and Jun Zhang. 2019. Dynamic Cooperative Coevolution for Large Scale Optimization. *IEEE Transactions on Evolutionary Computation* (2019), 1–1. https://doi.org/10.1109/TEVC.2019.2895860

[72] Junbo Zhao, Michael Mathieu, and Yann LeCun. 2016. Energy-based generative adversarial network. *arXiv preprint arXiv:1609.03126* (2016).