Coevolutionary Computation for Adversarial Deep Learning

Jamal Toutouh & Una-May O'Reilly
ALFA: AnyScale Learning for All Research Group
MIT CSAIL
toutouh@mit.edu, unamay@csail.mit.edu

http://lipizzaner.csail.mit.edu/

GECCO '21 Companion, July 10–14, 2021, Lille, France

ACM ISBN 978-1-4503-8351-6/21/07.
https://doi.org/10.1145/3449726.3461419

Instructors

- **Jamal Toutouh** is a MSCA Postdoctoral Fellow at the ALFA group (CSAIL-MIT) and the NEO team (University of Málaga). He obtained his Ph.D. in Computer Engineering at the University of Malaga. The dissertation, "Natural Computing for Vehicular Networks," was awarded the 2018 Best Spanish Ph.D. Thesis in Smart Cities. The dissertation focused on analyzing and devising machine learning (ML) methods inspired by Nature to address Smart Mobility problems. His current research explores the combination of Nature-inspired methods applied to ML and deep learning. He uses his novelty methods to Smart Cities and Climate Change.

- **Una-May O'Reilly** is leader of the AnyScale Learning For All (ALFA) group at MIT CSAIL. ALFA focuses on evolutionary algorithms, machine learning and frameworks for large scale knowledge mining, prediction and analytics. The group has projects in cyber security using coevolutionary algorithms to explore adversarial dynamics in networks and malware detection. Una-May received the EvoStar Award for Outstanding Achievements in Evolutionary Computation in Europe in 2013. She is a Junior Fellow (elected before age 40) of the International Society of Genetic and Evolutionary Computation, which has evolved into ACM Sig-EVO. She now serves as Vice-Chair of ACM SigEVO. She served as chair of the largest international Evolutionary Computation Conference, GECCO, in 2005.

About You

- EA and CoEA experience?
- ML and DL experience?
- Programming? algorithms?
- Native English speakers?

Learning Outcomes

- describe generative modeling and generative adversarial networks (GANs)
- identify the intellectual intersection between GANs and coevolutionary algorithms
- describe the design principles of a GAN and be familiar with simple code for one
- use Python code to run a demonstration framework

Course Agenda

- Generative Machine Learning
- Generative Adversarial Networks
- Spatial Co-evolutionary GAN Training
- Lipizzaner
- Questions & Discussion (1)
- Hands-on experiments
- Questions & Discussion (2)

Course Agenda

- **Generative Machine Learning**
- Generative Adversarial Networks
- Spatial Co-evolutionary GAN Training
- Lipizzaner
- Questions & Discussion (1)
- Hands-on experiments
- Questions & Discussion (2)

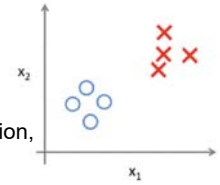
Generative Machine Learning

Coevolutionary Computation for Adversarial Deep Learning

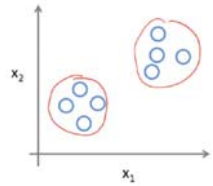
Jamal Toutouh & Una-May O'Reilly

Supervised vs Unsupervised Learning

- **Supervised Learning**
  - Given data x, predict output y
  - Goal: Learn a **function to map x → y**
  - Requires labeled data
  - Methods: Classification, Regression, Detection, Segmentation



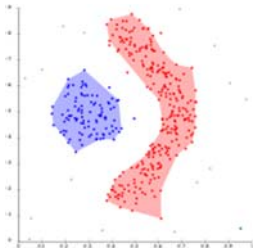
- **Unsupervised Learning**
  - Given data x
  - Goal: Learn the hidden or **underlying structure of the data**
  - Requires data (**no labels**)
  - Methods: Clustering/Density, Compression



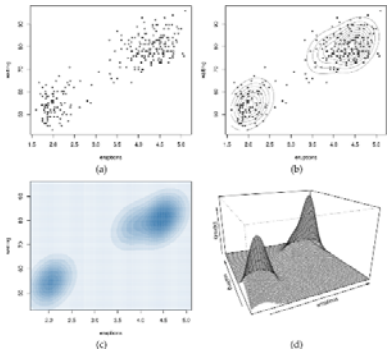
Supervised vs Unsupervised Learning

- **Unsupervised Learning examples**

Clustering



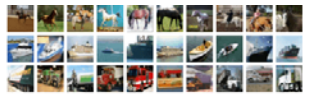
Density estimation



Generative Modeling

- **Overview:** Given training a training dataset, generate new samples from same distribution → Addressing ***density estimation***

Training dataset ~ $p_{data}(x)$

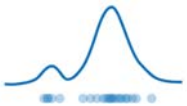


Generated samples ~ $p_{model}(x)$

- **Density estimation:** estimate the probability density function $p_{model}(x)$ of a random variable x, given a bunch of observations from the training dataset $p_{data}(x)$
  - Understand better the data distribution
  - Compress the data representation
  - Generate samples

Generative Modeling

- Flavors:
  - **Explicit** density estimation: estimates the true Probability Density Functions (PDF) or Cumulative Distribution Functions (CDFs) over the sample space, i.e., defines and solves for $p_{model}(x)$



  - **Implicit** density estimation: generates a function that can draw samples from the true distribution, i.e., learn model that can sample from $p_{model}(x)$ without explicitly defining it

Training samples

Synthetic samples

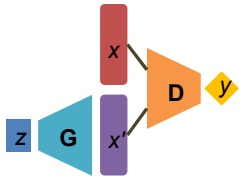
Generative Modeling

Explicit density estimation

Variational Autoencoders (VAEs)

Implicit density estimation

Generative Adversarial Networks (GAN)



Course Agenda

- Generative Machine Learning
- **Generative Adversarial Networks**
- Spatial Co-evolutionary GAN Training
- Lipizzaner
- Questions & Discussion (1)
- Hands-on experiments
- Questions & Discussion (2)

Generative Adversarial Networks

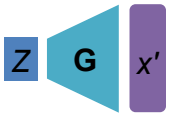
Coevolutionary Computation for Adversarial Deep Learning

Jamal Toutouh & Una-May O'Reilly

Generating Synthetic Samples

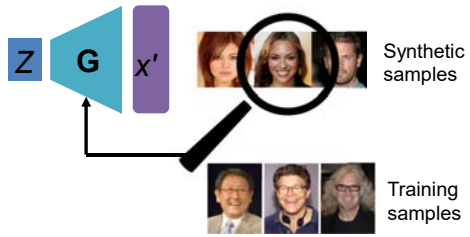
- **Global idea:** Generating new synthetic samples without modeling the density estimation, i.e., implicit density estimation
- **Solution:** Sampling from something simple (random distribution) and learning a transformation to the real (training) distribution

- Main components of the **Generative Model:**
  - Generator Neural Network: G
  - Random (latent space): Z
  - Synthetic sample from the training distribution: x'



How do the Generator Learn?

- Using another model that gives *feedback* about how close/far are the samples from the distribution that represents the real training dataset → **Discriminator**

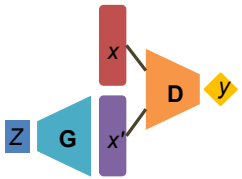


Generative Adversarial Networks

- Generative Adversarial Networks (GAN) construct a generative model by raising an arms race between two neural networks, a **generator** and a **discriminator** [6]

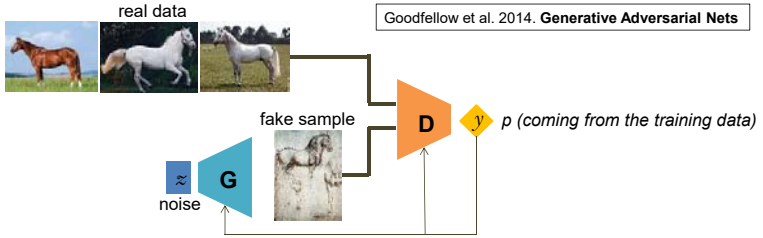
Goodfellow et al. 2014. **Generative Adversarial Nets**

- **Discriminator** (D) tries to distinguish between real data (x) from the training dataset distribution and fake data (x') from the generator (G)
- **Generator** (G) learns how to create synthetic/fake data samples (x') by sampling random noise (Z) to fool the discriminator (D)



Generative Adversarial Networks

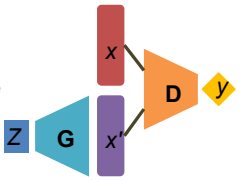
- **Discriminator** attributes a probability *p* of confidence of a sample being real (i.e. coming from the training data)
- **Generator** learns the real data distribution to generate fake samples



- Generator slightly changes the generated data based on Discriminator's feedback

Generative Adversarial Networks

- Generator and Discriminator are trained together (**minimax game**)
- **Discriminator** is trained to correctly classify the input data as either real or fake
  - maximize the probability that any real data input x is classified as real → maximize D(x)
  - minimize the probability that any fake sample x' is classified as real → minimize D(G(z))
- **Generator** is trained to fool the Discriminator
  - maximize the probability that any fake sample x' is classified as real → maximize D(G(z))



$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

- The training ideally converges in a scenario in which the Generator produces such a realistic samples that the Discriminator cannot distinguish between real and fake samples, i.e., p=0.5

Generative Adversarial Networks

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**
  for k steps **do**
    - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
    - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
    - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for
  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

GAN Applications

- Generate new samples of image datasets




GAN Applications

- Image-to-Image translation

GAN Applications

- Text-to-Image translation

GAN Applications

- Semantic-Image-to-Photo translation

http://nvidia-research-mingyuliu.com/gaugan

GAN Applications

- Deepfake videos

https://www.youtube.com/watch?v=cQ54GDm1eL0&feature=youtu.be&t=22

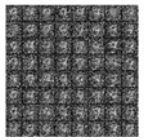
GAN Applications

- Many many others…

Training Pathologies

- **Non-convergence:** the model parameters oscillate, destabilize, and never converge
- **Mode collapse:** the generator collapses which produces limited varieties of samples

- **Diminished gradient:** the discriminator gets too successful that the generator gradient vanishes and learns nothing



Learning Variants

- **Some variants**

Name	Discriminator Loss	Generator Loss
Minimax GAN	$\mathcal{L}_D^{GAN} = -\mathbb{E}_{\mathbf{x}} \log D(\mathbf{x}) - \mathbb{E}_{\mathbf{z}} \log (1 - D(G(\mathbf{z})))$	$\mathcal{L}_G^{GAN} = \mathbb{E}_{\mathbf{z}} \log(1 - D(G(\mathbf{z})))$
Non-Saturating GAN	$\mathcal{L}_D^{NSGAN} = \mathcal{L}_D^{GAN}$	$\mathcal{L}_G^{NSGAN} = -\mathbb{E}_{\mathbf{z}} \log D(G(\mathbf{z}))$
Least-Squares GAN	$\mathcal{L}_D^{LSGAN} = \mathbb{E}_{\mathbf{x}}(D(\mathbf{x}) - 1)^2 + \mathbb{E}_{\mathbf{z}} D(G(\mathbf{z}))^2$	$\mathcal{L}_G^{LSGAN} = \mathbb{E}_{\mathbf{z}}(D(G(\mathbf{z})) - 1)^2$
Wasserstein GAN	$\mathcal{L}_D^{WGAN} = -\mathbb{E}_{\mathbf{x}} D(\mathbf{x}) + \mathbb{E}_{\mathbf{z}} D(G(\mathbf{z}))$	$\mathcal{L}_G^{WGAN} = -\mathbb{E}_{\mathbf{z}} D(G(\mathbf{z}))$
WGAN-GP	$\mathcal{L}_D^{WGANGP} = \mathcal{L}_D^{WGAN} + \lambda \mathbb{E}_{\mathbf{x},\mathbf{z}}(\lVert \nabla D(\alpha \mathbf{x} + (1-\alpha) G(\mathbf{z})) \rVert_2 - 1)^2$	$\mathcal{L}_G^{WGANGP} = \mathcal{L}_G^{WGAN}$
DRAGAN	$\mathcal{L}_D^{DRAGAN} = \mathcal{L}_D^{GAN} + \lambda \mathbb{E}_{\mathbf{x} \sim p_{data} + \mathcal{N}(0,c)}(\lVert \nabla D(\mathbf{x}) \rVert_2 - 1)^2$	$\mathcal{L}_G^{DRAGAN} = \mathcal{L}_G^{GAN}$
BEGAN	$\mathcal{L}_D^{BEGAN} = \mathbb{E}_{\mathbf{x}} \lVert \mathbf{x} - AE(\mathbf{x}) \rVert_1 - k_t \mathbb{E}_{\mathbf{z}} \lVert G(\mathbf{z}) - AE(G(\mathbf{z})) \rVert_1$	$\mathcal{L}_G^{BEGAN} = \mathbb{E}_{\mathbf{z}} \lVert G(\mathbf{z}) - AE(G(\mathbf{z})) \rVert_1$

Evaluation Metrics

- How to evaluate the generated samples?
  - We cannot rely on the **models' loss**
  - **A human** cannot qualitatively evaluate the whole distribution of generated data or an extensive representation of it
  - **A pre-trained model** can be used to assess the quality of a high number of samples
- There are a number of evaluation metrics [20]
  - Inception Score (IS)
  - Fréchlet Inception Distance (FID)

Inception Score

- Well correlated with human perception
- Use pre-trained model: Inception v3 trained on ImageNet-1K
- KL-Divergence between conditional and marginal label distributions over generated data [14]
  - If the image x is recognized as y label, p(y|x) should have low entropy → p(y|x) is evaluated with inception model
  - If model outputs diverse images, p(y) should have high entropy → p(y) is calculated with marginal

$$\mathrm{IS}(G) = \exp\left(\mathbb{E}_{\mathbf{x}\sim p_g} D_{KL}(p(y|\mathbf{x}) \parallel p(y)) \right),$$

- Higher IS represents better generative model
- It does not use the real training dataset

Fréchlet Inception Distance

- The Fréchlet Inception Distance (FID) score was proposed to overcome the various shortcomings of the IS [8]
- Inception network is used to extract the feature maps from an intermediate layer
- A multivariate Gaussian distribution learns the distribution of the feature maps
  - Mean and covariance for layer with real data *r* and generated data *g*

$$FID = \|\mu_r - \mu_g\|^2 + T_r(\Sigma_r + \Sigma_g - 2(\Sigma_r\Sigma_g)^{1/2})$$

  - Wasserstein-2 distance between multi-variate Gaussians fitted to data embedded into a feature space
- Lowe FID represents better generative model
- Consistent with human perception, more robust to noise than IS, and can detect intra-class mode dropping

Course Agenda

- Generative Machine Learning
- Generative Adversarial Networks
- **Spatial Co-evolutionary GAN Training**
- Lipizzaner
- Questions & Discussion (1)
- Hands-on experiments
- Questions & Discussion (2)

ALFA ANYSCALE LEARNING FOR ALL NEO

Spatial Co-evolutionary GAN Training

Coevolutionary Computation for Adversarial Deep Learning

Jamal Toutouh & Una-May O'Reilly

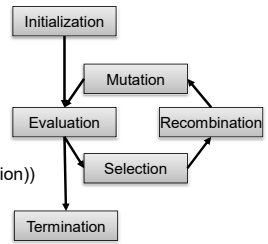
Coevolution: Evolutionary Algorithms

- **Evolutionary computation** comprises a set of computational methods (metaheuristics) that mimics **biological evolution** [5]
  - They apply a mechanism analogous to natural evolutionary processes, **to solve search and optimization** problems
  - They work with a **population** (of representations) of solutions
  - Principles:
    - natural selection (**fitness**)
    - reproduction (**recombination** and **mutation**)
    - genetic **diversity**
  - They follow the idea of **survival of the fittest** individuals, evaluating the fitness according to the problem to be solved, through a **fitness function**

Coevolution: Evolutionary Algorithms

- **Evolutionary Algorithm**

1. generation = 0
2. population(0) = Create **initial** population
3. while not stop criteria do
   1. ***evaluate***(population(generation))
   2. parents = ***selection***(population(generation))
   3. offspring = ***recombine***(parents, rec_probability)
   4. offspring = ***mutate***(parents, mut_probability)
   5. new_population = ***replace***(offspring, population(generation))
   6. generation++
   7. population(generation) = new_population



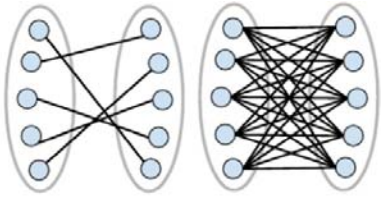
Coevolution

- Coevolutionary algorithms were proposed to address problems with complex structure [14], i.e., **fitness depends on the context:**
  - An evolved individual does not represent a complete solution of the problem
  - The evaluation of an individual is dependent on other individuals (i.e., individuals are explicitly part of the environment)

- In **biology**, coevolution occurs when individuals of one or more species reciprocally affect each other's evolution through the process of natural selection
  - **Positive effect** (mutualism/symbiosis), e.g. plants and insects
    - Cooperative coevolution
  - **Negative effect** (predation/parasitism), e.g. foxes and rabbits
    - Competitive coevolution

Coevolution

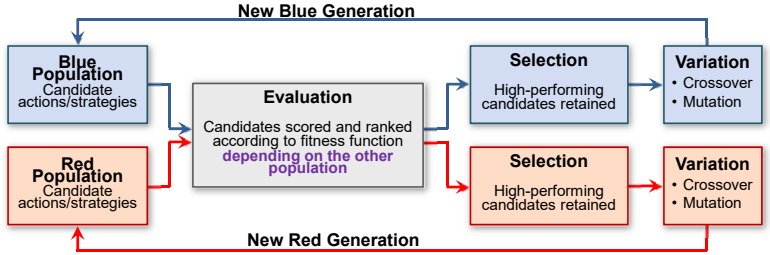
- In **competitive coevolution**, a number of different species, each representing part of a problem, cooperate in order to solve a larger problem
- In **competitive coevolution**, individuals evolve and are evaluated based on direct competition against individuals of a different species, which in turn evolve separately



One-vs-one All-vs-all

Competitive Coevolution

- Competitive coevolutionary algorithms have **adversarial populations** (usually two) that simultaneously evolve solutions against each other with members engaging in **two-player games** [11]
- 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**



- In seminal work, more efficient sorting problems were produced by engaging two populations, one of programs and one of tests, in a competition [9]

Coevolution Limitations

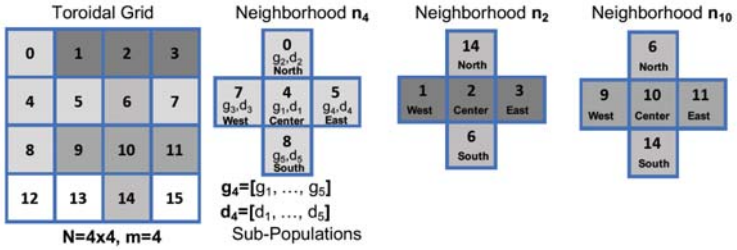
- **Focusing:** The ability to focus on an opponent's weakness can provide an easy way to win. This may produce degenerate players that over-specialize on opponents weaknesses, and fail to learn a task in a general way → *GAN mode collapse*
- **Loss of gradient**: One population comes to severely dominate the others, thus creating an impossible situation in which the other participants are not able to learn → *GAN diminished gradient*
- **Cyclic:** One population loses the genetic knowledge of how to defeat an earlier generation adversary and that adversary re-evolves → *GAN non-convergence*
- As these limitations have already been broadly studied, literature proposes a set of **general remedies** [4]

Coevolution Limitations

- 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
- **Solution diversity** has been explicitly improved with competitive fitness sharing, **separation**, e.g. a spatial topology, or a spatial topology and temporal segregation

Spatial Coevolution

- **A spatial (2D toroidal) topology** is an effective means of controlling the mixing of adversarial populations in coevolutionary algorithms [10]
- The members of populations are divided up on a grid of cells and overlapping neighborhoods (e.g., von Neuman) for each cell are identified. 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
  - It reduces the cost of interaction from O(N^2) to O(Ns), where N is pop. size
  - Each neighborhood can evolve in semi-isolation



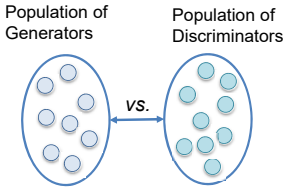
Coevolution and GANs, Reminding

- GAN training can be seen as a **two-player minimax game/problem**

  generator $G(z)$ vs discriminator $D(x)$
- Coevolutionary algorithms addressed **similar issues** in two-player minimax optimization as GAN training

  focusing, relativism or loss of gradient

- Main idea: using **competitive coevolution** as a main framework to train GANs [1,2,3,7,12,15,16]



Course Agenda

- Generative Machine Learning
- Generative Adversarial Networks
- Spatial Co-evolutionary GAN Training
- **Lipizzaner**
- Questions & Discussion (1)
- Hands-on experiments
- Questions & Discussion (2)

Lipizzaner

Coevolutionary Computation for Adversarial Deep Learning

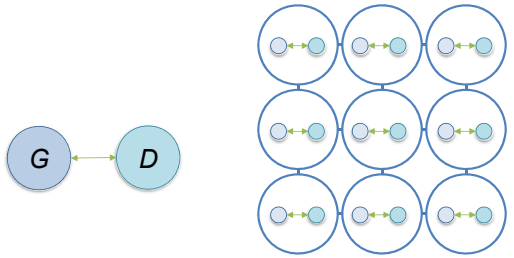
Jamal Toutouh & Una-May O'Reilly

Lipizzaner

- A **spatially distributed**, **coevolutionary** framework to train GANs with **gradient-based** optimizers [1,7,15]
- Main features:
  - It trains/evolves **two populations**, one of generators and one of discriminators, by using a competitive coevolutionary algorithm
  - The populations are distributed upon a **spatial grid**
  - **Asynchronous parallel** training in each cell
  - The **learning rate** parameter is **evolved** through the training process
  - At the end of the training process each cell contains a generative model defined by an **ensemble of the generators** in the sub-population
  - The generative models are evolved (as well) to optimize the quality of the synthesized samples, i.e., **ensemble weights evolution**

Spatial Coevolution GAN Training

- Instead of training a single GAN, **Lipizzaner trains two populations**, a population of generators against a population of generators
- Each pair generator-discriminator is located in a cell of a toroidal grid, i.e., **GANs are spatially distributed**



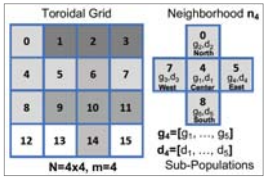
Single-GAN training framework

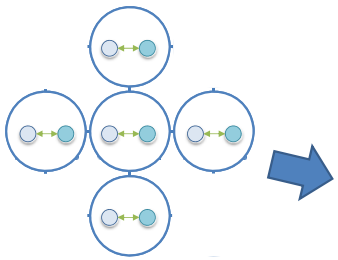
Spatially Distributed COEA GAN training framework

Spatial Coevolution GAN Training

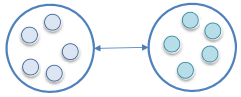
- For each cell, two **sub-populations** are defined by gathering the individuals (neural networks) from the overlapping neighborhoods (e.g., von Neuman or von Neuman)

Overlapping von Neuman neighborhood of cell *i* to define sub-populations



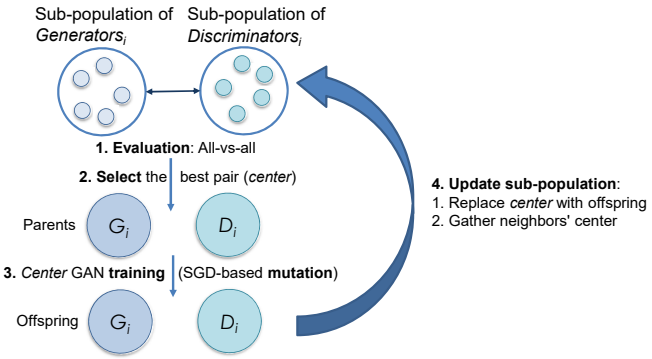


Sub-population of *Generators*$_i$ Sub-population of *Discriminators*$_i$



Spatial Coevolution GAN Training

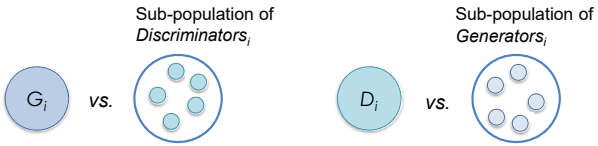
- Each cell performs in **parallel the coevolutionary GAN training** loop for a given number of iterations (training epochs)



- After each training epoch, the **sub-populations are updated** by gathering the *center* GAN of the neighborhood cells

Spatial Coevolution GAN Training

- The *center* GAN training consists of **updating the weights** of the selected center generator and discriminator by using a number of training data mini-batches
  - The *center* generator is trained against a randomly chosen discriminator from the sub-population
  - The *center* discriminator is trained against a randomly chosen from the sub-population



- **Gaussian-based mutation** is applied to update the **learning rate** after each training epoch

Spatial Coevolution GAN Training

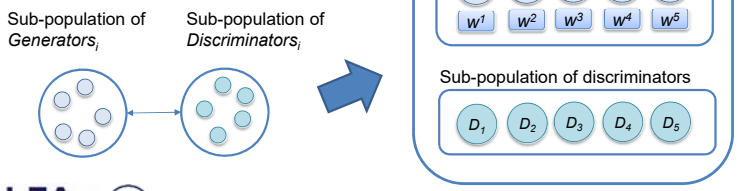
- Coevolution and training algorithm is performed in each cell, a.k.a., *CoevolveAndTrainModels*

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

```
B ← getMiniBatches(X)                                   ▷ Load minibatches
φ ← EvaluateGANPairs(B, n)                              ▷ Evaluate all updated GAN pairs, Alg. 4
g, d ← tournamentSelect(n, φ, τ)                        ▷ Select using loss(ℒ) as fitness
for B ∈ B do                                            ▷ Loop over batches
    n_δ ← mutateLearningRate(n_δ, β)                    ▷ Update neighborhood learning rate with with Gaussian mutation
    d ← getRandomOpponent(d)                            ▷ Get uniform random discriminator
    for g ∈ g do                                        ▷ Evaluate generators and train with SGD
        g, d ← pickRandom(M)                            ▷ Random draw of loss function for each generator and discriminator
        ∇_g ← computeGradient(g, d, B)                  ▷ Compute gradient for neighborhood center
        g ← updateNN(g, ∇_g, B)                         ▷ Update with gradient
    g ← getRandomOpponent(g)                            ▷ Get uniform random generator
    for d ∈ d do                                        ▷ Evaluate discriminator and train with SGD
        g, d ← pickRandom(M)                            ▷ Random draw of loss function for each generator and discriminator
        ∇_d ← computeGradient(d, g,)                    ▷ Compute gradient for neighborhood center
        d ← updateNN(d, ∇_d, B)                         ▷ Update with gradient
φ ← EvaluateGANPairs(B, n)                              ▷ Evaluate all updated GAN pairs, Alg. 4
n ← replaceCenterIndividuals(n, φ)                      ▷ Best generator and discriminator are placed in the center based on loss
return n
```


Ensembles evolution

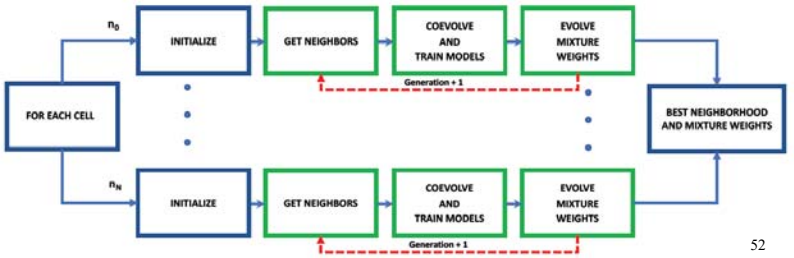
- Lipizzaner at the highest level searches for and returns **a mixture of generators** e composed from a sub-population
- It evolves a mixture **weight vector *w*** for each neighborhood n using an ES-(1+1) algorithm which optimizes for generator ensemble performance (e.g., inception score or FID)
- The **weight vector *w*** is updated after each training epoch



Lipizzaner global method

Input: T : Total generations, N : Population on grid cells, s : Neighborhood size, θ_{EA} : Parameters for `MixtureEA`, θ_{COEV} : Parameters for `CoevolveAndTrainModels`
Return: **e** : ensemble of generators and mixture weights

```
parfor k ∈ [1, ..., |N|] do                             ▷ Asynchronous parallel execution of all cells in grid
    n_k, w, f_w ← initializeNeighborhoodAndMixtureWeights(s)   ▷ Uniformly random initialization of settings
    for dot ∈ [0, ..., T] do                            ▷ Iterate over generations
        n'_k ← copyNeighbors(k, s, N)                   ▷ collect neighbour cells individuals for the sub-populations
        n'_k ← CoevolveAndTrainModels(n'_k, θ_COEV)     ▷ Coevolve GANs using Algorithm 3
        n_k ← updateNeighborhood(n'_k, k, N)            ▷ Update neighborhood (population), i.e. Y
        w, f_w ← MixtureEA(w, f_w, n_k, θ_EA)           ▷ Evolve mixture weights, Algorithm 2
end parfor
return e*                                               ▷ Best generator mixture
```



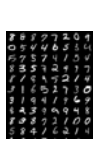
52

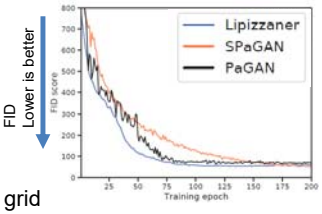
Main Advantages

- **Fast convergence** due to gradient-based steps
- **Improved convergence** due to hyperparameter evolution
- **Resilience** due to coevolution and communication
- **Robustness** due to the use of ensembles
- Improved samples **quality** and **diversity** due to mixture evolution
- **Scalability** due to spatial distribution topology and asynchronous parallelism

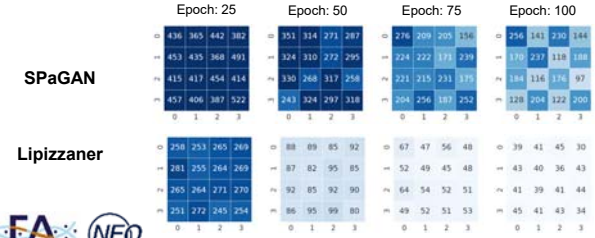
Faster Convergence

- Experiment on MNIST dataset with Lipizzaner variations (4x4 grid) [17]
  - **Lipizzaner**
  - Spatial Parallel GAN (**SPaGAN**): no selection/replacement
  - Parallel GAN (**PaGAN**): no communication



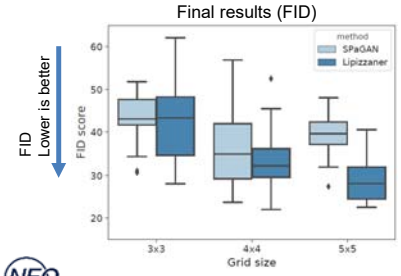


- FID score evolution in the cells of a 4x4 grid



Improved Convergence

- Experiment on MNIST dataset with Lipizzaner and SPaGAN on different grid sizes
- Lipizzaner provides better results than SPaGAN
- As the grid size increase (larger populations), Lipizzaner converges to better generative models



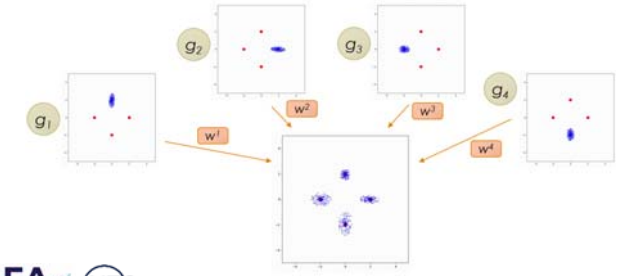
Resilience

- **Coevolution** and exchange of individuals through the grid allow cells to addresses **vanishing gradient issues**
- For example, in the training **epoch 50**, there is a **powerful discriminator** that does not allow a generator to learn in a cell of a 4x4 grid (the cell shows a dark color to represent a high FID score)

- In the successive epochs, with the exchange of the individuals plus the selection and replacement process, the weak **generator in the cell is replaced** by stronger ones. Thus, the **cell can escape** from that pathological situation, and it can generate better samples (lighter color, lower FID score)

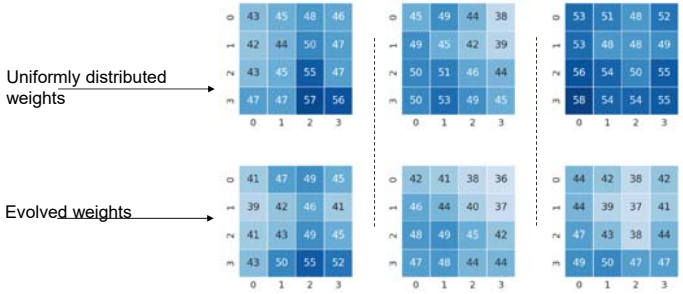
Robustness

- **Ensemble (mixture)** of generators overcomes **mode collapse**
- This extreme example shows four generators that collapse in each one of the four modes of the training dataset. A mixture of these four generators provides a generative model that can generate data samples of each mode



Improved Quality

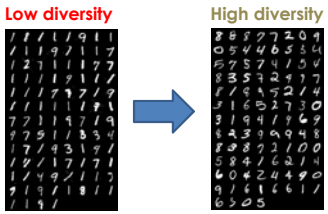
- The following example shows the quality of the generated samples in a 4x4 grid when **using evolved mixture weights** or **uniformly distributed mixture weights** (i.e., the samples are generated by mixtures created with the same generators but different weights)



- The use of **evolved ensembles** improves **quality** of the samples (provide lower FID score)

Improved Diversity

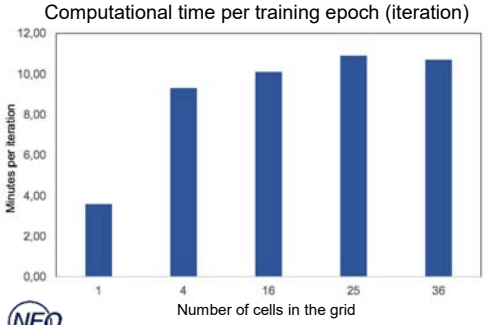
- The use of **evolved ensembles** improves the diversity of the generated samples [19]
- In this example, the diversity on MNIST dataset is evaluated in terms of the Total Variation Distance (lower distance more diverse generated data)



Ensemble	TVD - Diversity
No (single generator)	0.113
Yes (5 generators)	**0.046**

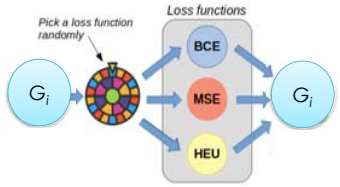
Scalability

- **Spatial distribution** in a 2D grid addressees the **quadratic computational complexity**
- **Asynchronous** communication
- Deployed over workstations, cloud based, and **HPC environments**
  - OpenStack, Google Cloud, AWS, Summit, MIT Satori, etc. [12]



Research Over Lipizzaner

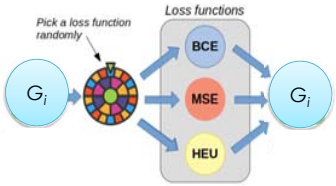
- **Mustangs:** For each training epoch, each cell randomly picks a loss function to optimize the networks' weights to increase the **genotype diversity**

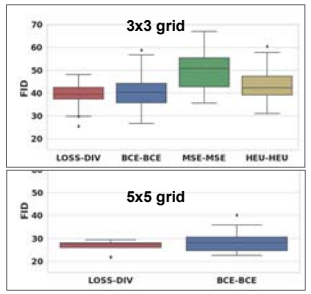


- **Data Dieting:** As we have communication between cells, do we need to replicate whole data among all the cells? **Data diversity**

Mustangs

- **Mustangs:** For each training epoch, each cell randomly picks a loss function to optimize the networks' weights to increase the **genotype diversity** [16]
- Lipizzaner when using one of the following loss functions: binary cross entropy (BCE-BCE), mean square error (MSE-MSE), and an heuristic one (HEU-HEU). And a variant of Lipizzaner that randomly picks one of these three loss functions (**LOSS-DIV**)

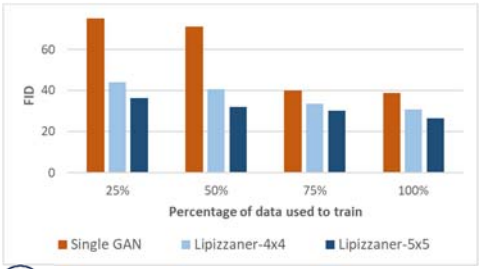




- **LOSS-DIV** increases the diversity and provides better generative models (lower FIDs)

Data Dieting

- **Data Dieting**: As we have communication between cells, do we need to replicate whole data among all the cells? **Data diversity** [18]
- **Signal propagation** (in the form of network exchange) through the grid allows the cells to exchange learning information
- Data diversity allows training the cells of the grid with **subsets of the training dataset**
- The experimental results show that, with Lipizzaner, the cells can be trained by using a lower amount of data



Course Agenda

- Generative Machine Learning
- Generative Adversarial Networks
- Spatial Co-evolutionary GAN Training
- Lipizzaner
- **Questions & Discussion (1)**
- Hands-on experiments
- Questions & Discussion (2)

ANYSCALE LEARNING FOR ALL

Course Agenda

- Generative Machine Learning
- Generative Adversarial Networks
- Spatial Co-evolutionary GAN Training
- Lipizzaner
- Questions & Discussion (1)
- **Hands-on experiments**
- Questions & Discussion (2)

65

Course Agenda

- Generative Machine Learning
- Generative Adversarial Networks
- Spatial Co-evolutionary GAN Training
- Lipizzaner
- Questions & Discussion (1)
- Hands-on experiments
- **Questions & Discussion (2)**

66

The End

Coevolutionary Computation for Adversarial Deep Learning

Jamal Toutouh & Una-May O'Reilly

Acknowledgements

- This research was partially funded by European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No 799078 and the Systems that Learn Initiative at MIT CSAIL.

68

References

[1] Al-Dujaili, A., Schmiedlechner, T., Hemberg, E., O'Reilly, U.M. "Towards distributed coevolutionary GANs." In *Association for the Advancement of Artificial Intelligence (AAAI) 2018 Fall Symposium*. 2018

[2] Costa, V., Lourenço, N., Correia, J., Machado, P. "COEGAN: evaluating the coevolution effect in generative adversarial networks." In *Proceedings of the Genetic and Evolutionary Computation Conference*. pp 374-382. 2019

[3] Costa, V., Lourenço, N., Machado, P. "Coevolution of generative adversarial networks." In *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)*. Springer, 473–487. 2019

[4] Ficici, S. G., & Pollack, J. B. "Challenges in coevolutionary learning: arms-race dynamics, open-endedness, and medicocre stable states." In *Proceedings of the sixth international conference on Artificial life*. pp. 238-247. 1998

[5] Fogel, D. B. "*Evolutionary computation: toward a new philosophy of machine intelligence*." John Wiley & Sons. 2006

[6] Goodfellow, I.J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y. "Generative Adversarial Nets." In *arXiv preprint arXiv:1406.2661*. 2014

[7] Hemberg, E., Toutouh, J., Schmiedlechner, T., O'Reilly, U.M. "Spatial Coevolution for Generative Adversarial Network Training". *ACM Transactions on Evolutionary Learning and Optimization*, pages 25, Springer. 2021

References

[8] Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., Hochreiter, S. "GANs trained by a two time-scale update rule converge to a local nash equilibrium." In *Proceedings of the 31st International Conference on Neural Information Processing Systems*. pp. 6629-6640. 2017

[9] Hillis, D.W. "Co-evolving parasites improve simulated evolution as an optimization procedure." *Physica D: Nonlinear Phenomena 42, 1* pp. 228 – 234. 1990

[10] Mitchell, M. "Coevolutionary Learning with Spatially Distributed Populations." *Computational Intelligence: Principles and Practice*. 2006

[11] O'Reilly, U. M., Toutouh, J., Pertierra, M., Sanchez, D. P., Garcia, D., Luogo, A. E., Kelly, J., Hemberg, E. "Adversarial genetic programming for cyber security: A rising application domain where GP matters." *Genetic Programming and Evolvable Machines, 21(1)*, 219-250. 2020

[12] Pérez, E., Nesmachnow, S., Toutouh, J., Hemberg, E., & O'Reily, U. M. "Parallel/distributed implementation of cellular training for generative adversarial neural networks." In *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. pp. 512-518. 2020

[13] Popovici E., Bucci A., Wiegand R.P., De Jong E.D. "Coevolutionary Principles." In *Rozenberg G., Bäck T., Kok J.N. (eds) Handbook of Natural Computing*. Springer, Berlin, Heidelberg. 2012

References

[14] Salimans, T., Goodfellow, I. J., Zaremba, W., Cheung, V., Radford, A., Chen, X. "Improved Techniques for Training GANs." In *Proceedings of the 30th International Conference on Neural Information Processing Systems*. 2016

[15] Schmiedlechner, T., Ng Zhi Yong, I., 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

[16] Toutouh, J., Hemberg, E., O'Reilly, U.M. "Spatial Evolutionary Generative Adversarial Networks." In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '19). ACM, New York, NY, USA, 472–480. 2019

[17] Toutouh, J., Hemberg, E., O'Reilly, U.M. "Analyzing the Components of Distributed Coevolutionary GAN Training." In *International Conference on Parallel Problem Solving from Nature*. Springer, Cham. pp. 552-566. 2020

[18] Toutouh, J., Hemberg, E., O'Reilly, U.M. "Data Dieting in GAN Training." In: *Iba H., Noman N. (eds) Deep Neural Evolution. Natural Computing Series*. Springer, Singapore. 2020

[19] Toutouh, J., Hemberg, E., O'Reilly, U.M. "Re-purposing heterogeneous generative ensembles with evolutionary computation." In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. pp 425-434. 2020

References

[20] Xu, Q., Huang, G., Yuan, Y., Guo, C., Sun, Y., Wu, F., Weinberger, K. "An empirical study on evaluation metrics of generative adversarial networks." *arXiv preprint arXiv:1806.07755*. 2018

Coevolutionary Computation for Adversarial Deep Learning

Jamal Toutouh & Una-May O'Reilly
ALFA: AnyScale Learning for All Research Group
MIT CSAIL
toutouh@mit.edu, unamay@csail.mit.edu

http://lipizzaner.csail.mit.edu/

GECCO '21 Companion, July 10–14, 2021, Lille, France

ACM ISBN 978-1-4503-8351-6/21/07.
https://doi.org/10.1145/3449726.3461419