Massachusetts Institute of Technology

ANYSCALE LEARNING FOR ALL

CSAIL

Lipizzaner:

Gradient-based Coevolution GAN Training

JAMAL TOUTOUH

jamal@lcc.uma.es

jamal.es

@jamtou

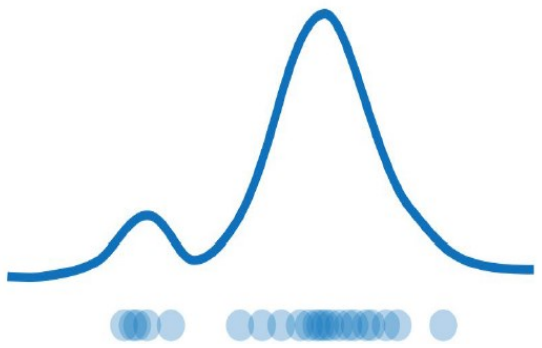
UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Generative Models

Generative Modeling

Goal: Given a distribution of data, take input training samples from it and learn a model that represents that distribution

- **Density estimation**

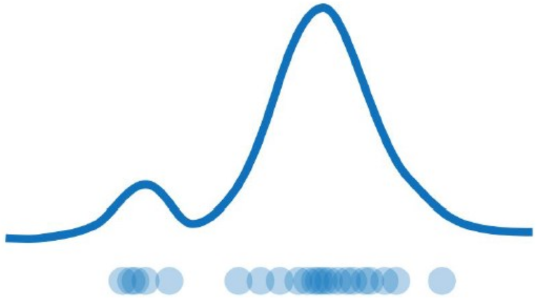


  - Understand better the data distribution
  - Compress the data representation
  - Generate samples

Generative Modeling

Goal: Given a distribution of data, take input training samples from it and learn a model that represents that distribution

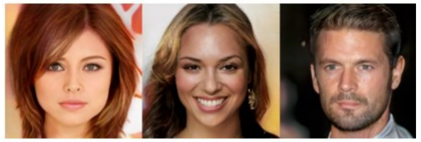
- **Density estimation**



- **Synthetic samples generation**

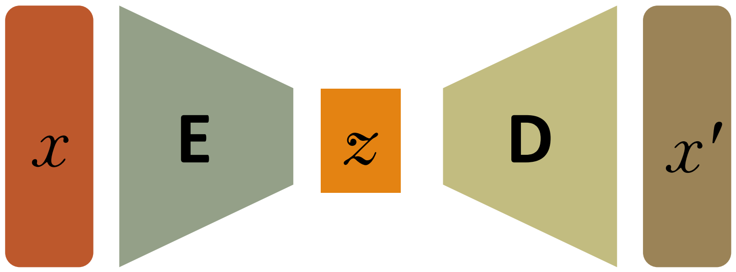
Training samples

Synthetic samples

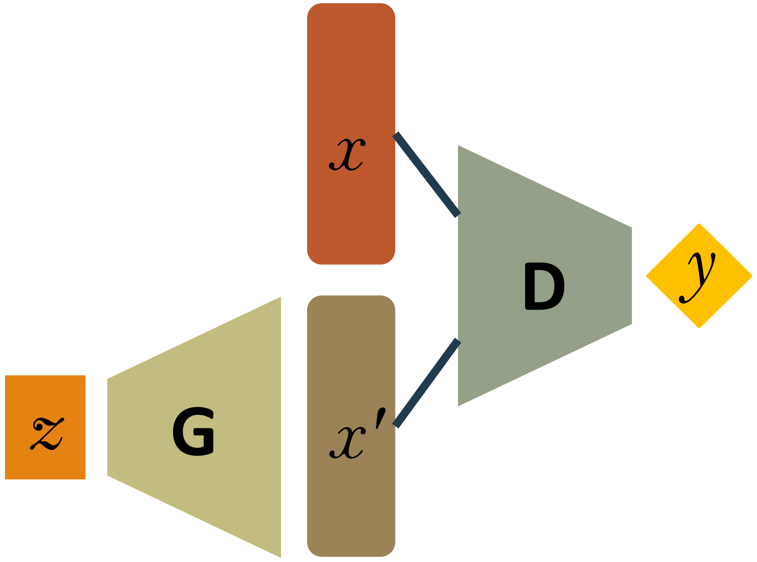


Models

Autoencoders and Variational Autoencoders (VAEs)

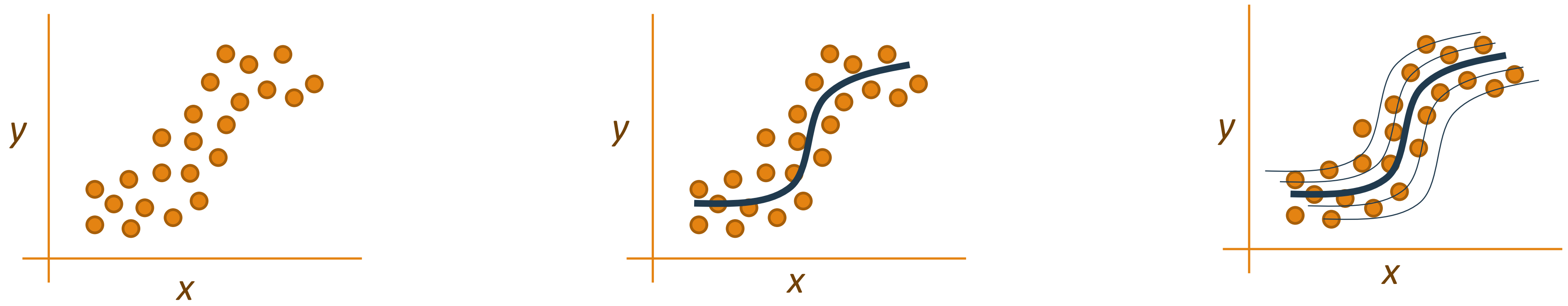


Generative Adversarial Networks (GANs)



Generative Adversarial Networks

Generating synthetic samples



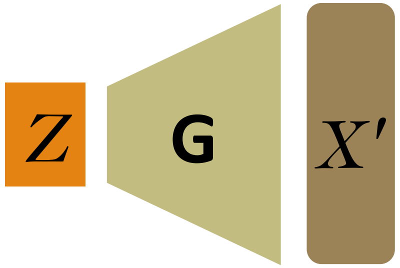
Generating synthetic samples

Global idea: Generating new synthetic samples without modeling the density estimation (for “complex” distributions)

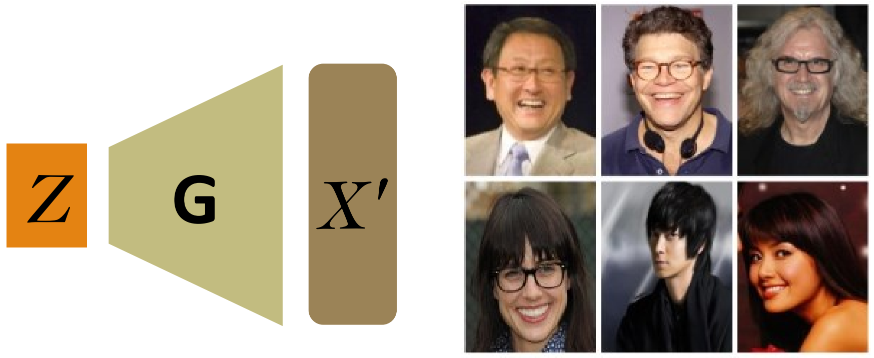
Solution: Sampling from something simple (noise) and learning a transformation to the real (training) distribution

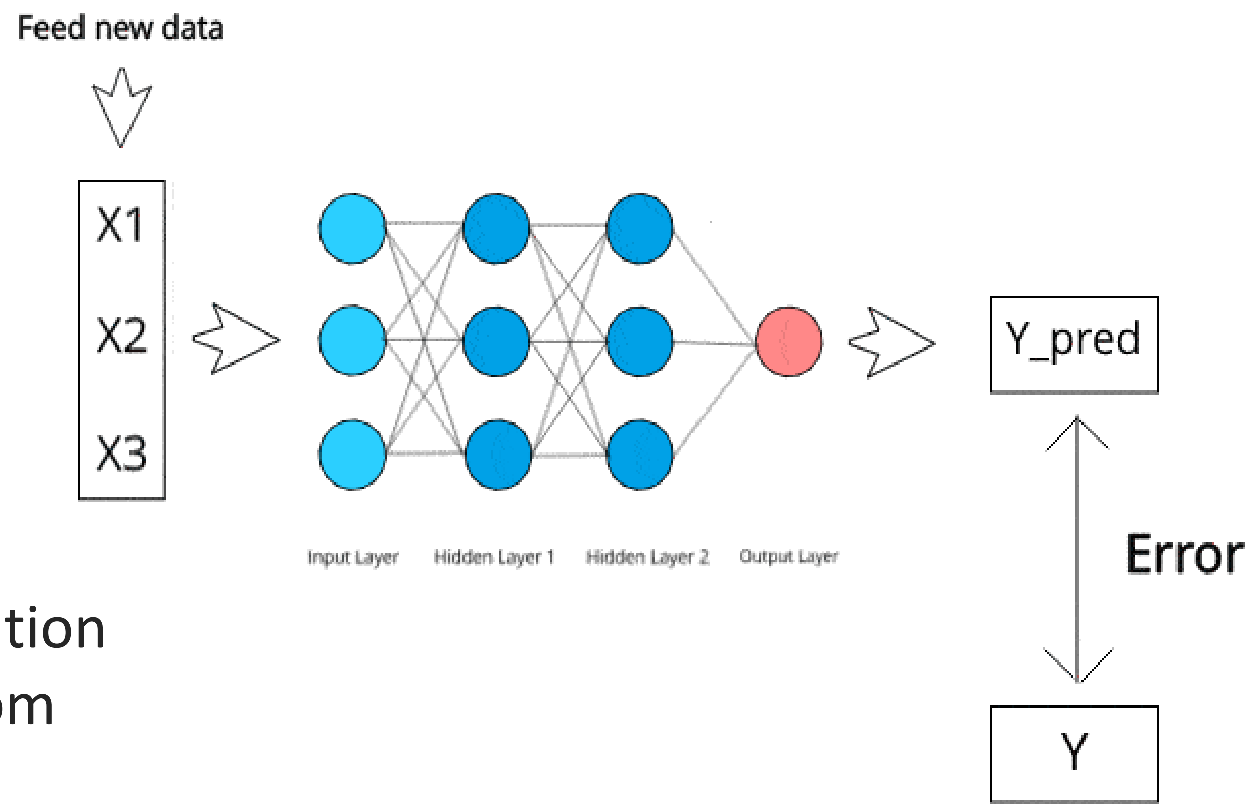
Main components of the **Generative Model:**

- **Generator** Neural Network → **G**
- **Noise** (latent space) → Z
- **Fake sample** from the training distribution → X'



How Generator Learn?





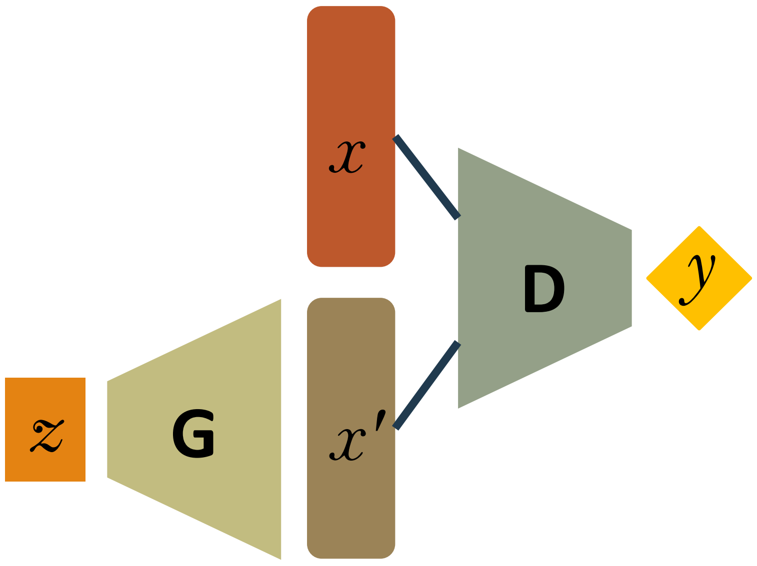
Using **another model** that gives information about how close/far are the samples from real data → **Discriminator**

Generative Adversarial Networks

Generative Adversarial Networks: Construct a generative model by raising an arms race between two neural networks, a **generator** and a **discriminator**

- Discriminator (**D**) tries to distinguish between real data (X) from the real data 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**)



GAN Training. Mathematical Model

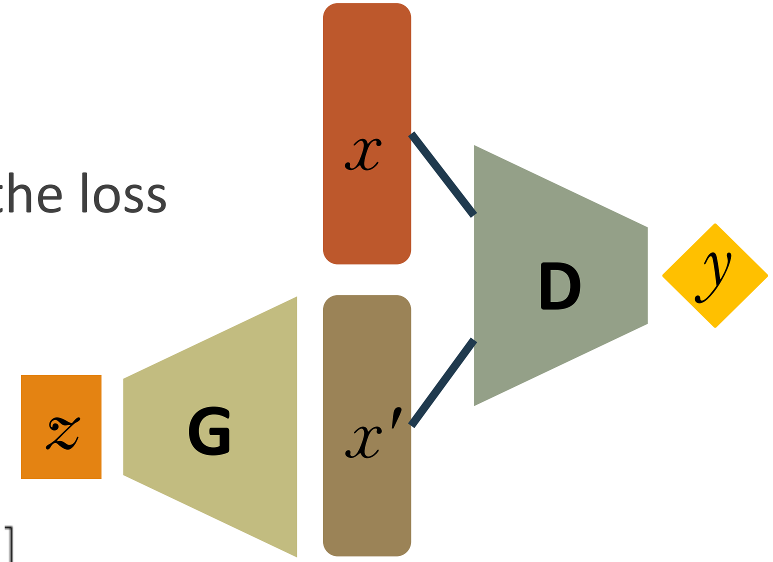
Discriminator is trained to correctly classify the input data as either real or fake

- **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 by generating realistic data

- **maximize** the probability that any fake sample is classified as real
  → **maximize D(G(z))**

In practice, the logarithm of the probability (e.g. log D(…)) is used in the loss functions

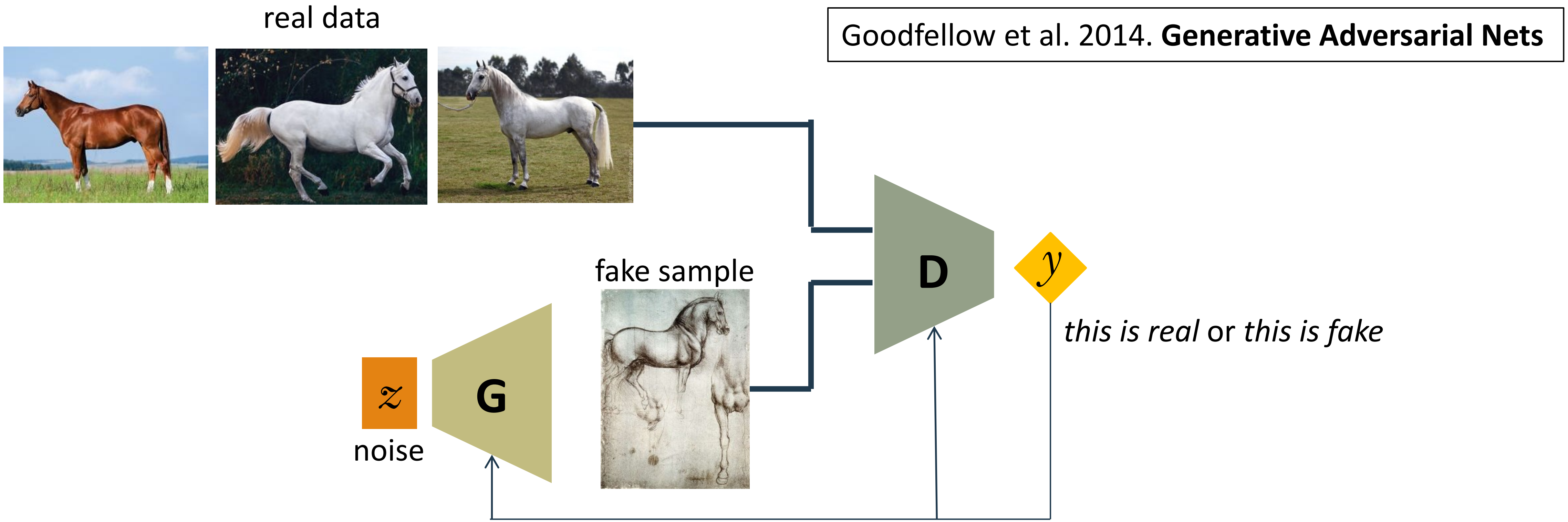


GAN training as a minmax optimization problem

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

Generative Adversarial Networks

Generative Adversarial Networks: Construct a generative model by raising an arms race between two neural networks, a **generator** and a **discriminator**



GAN Training. General Algorithm

Steps of the main training loop:

1. Train discriminator

1.1. Train discriminator on real data

1.1.1 Sample a batch of data from real dataset (x)

1.1.2 Get loss from the discriminator output with input x

1.2 Train the discriminator on data produced by the generator

1.2.1 Sample a batch of data from random latent space (z)

1.2.2 Get samples (x') from the generator with input z

1.2.3 Get loss from the discriminator output with input x'

1.3 Update discriminator weights according to the losses

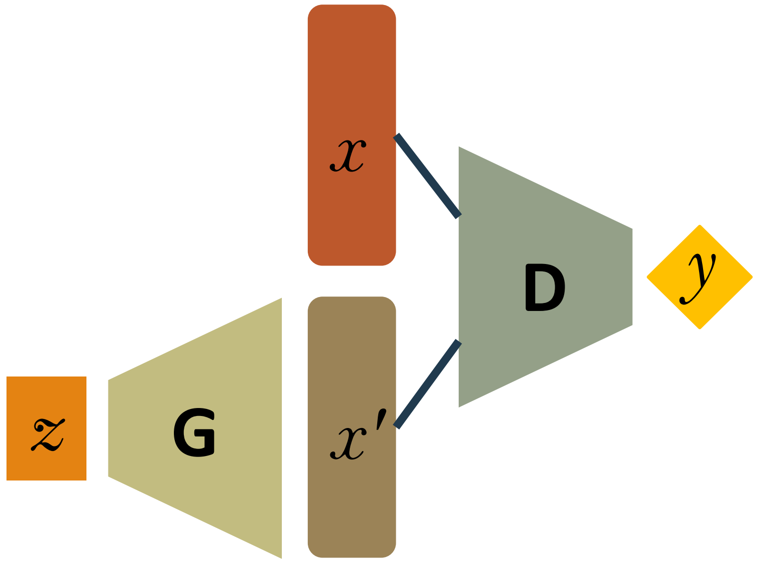
2. Train the generator

2.1 Sample a batch of data from random latent space (z)

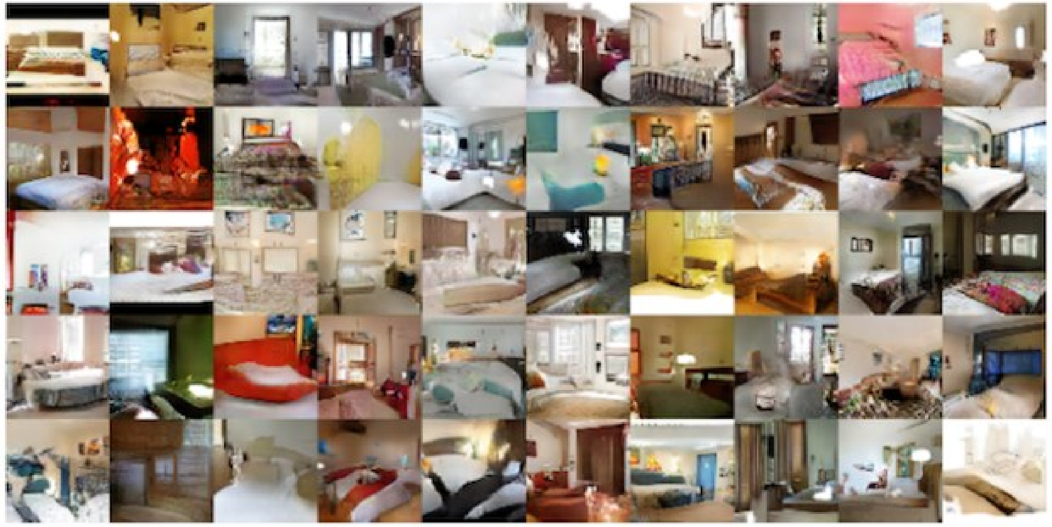
2.2 Get samples (x') from the generator with input z

2.3 Get loss from the discriminator output with input x'

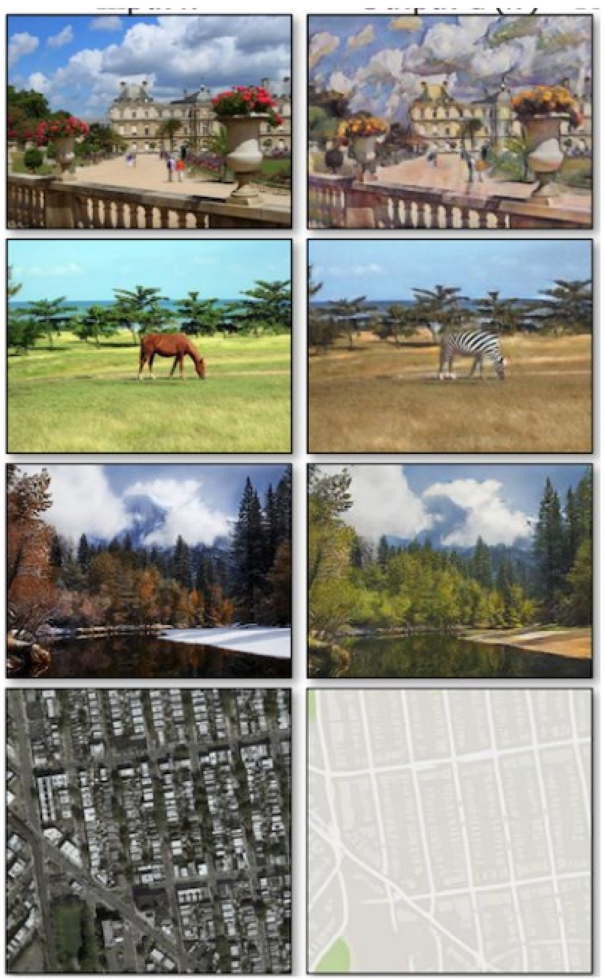
2.4 Update generator weights according to the losses



Applications: Generate New Samples of Image Datasets

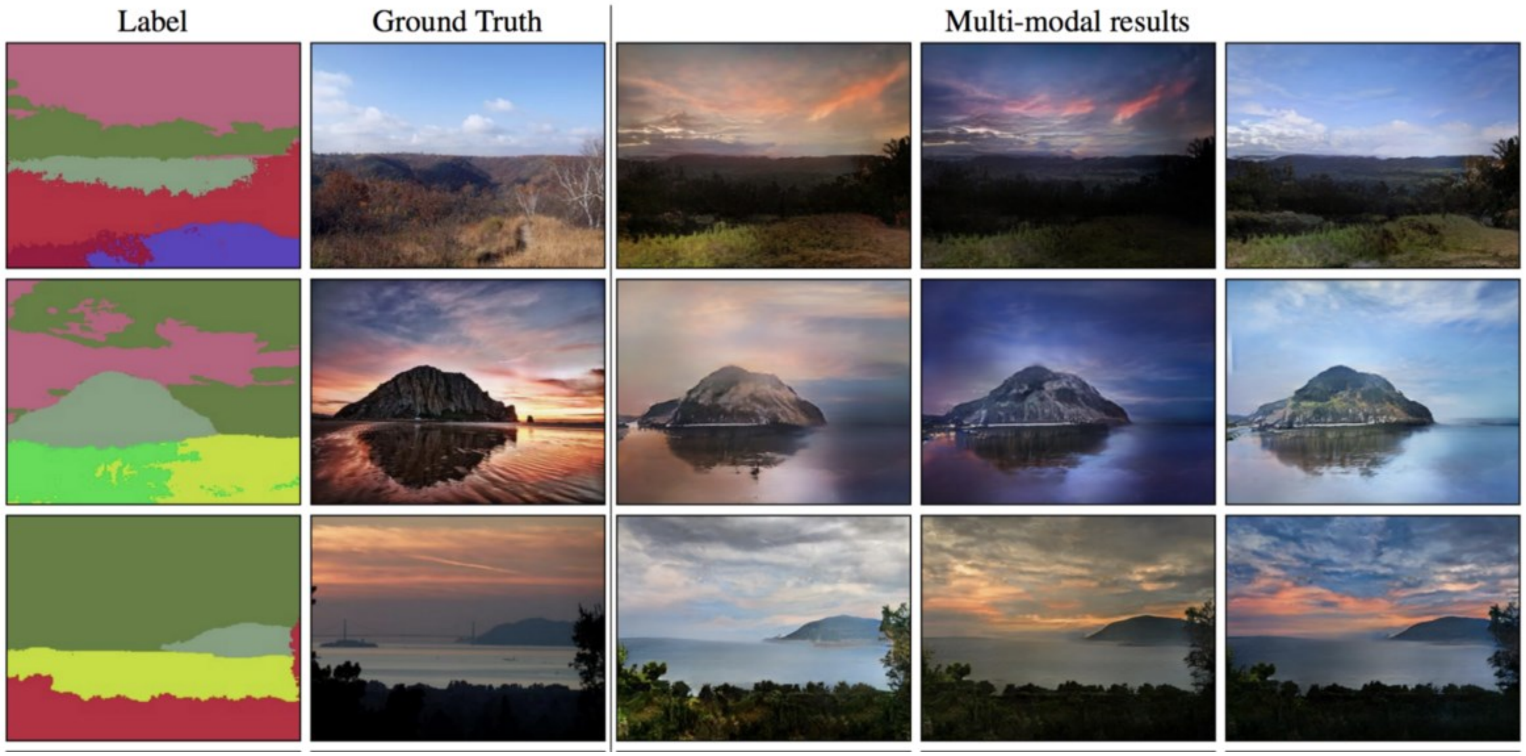


Applications: Image-to-Image Translation



Applications: Text-to-Image Translation

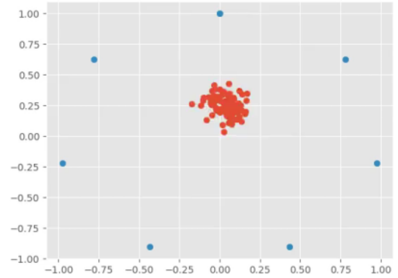
Applications: Semantic-Image-to-Photo Translation



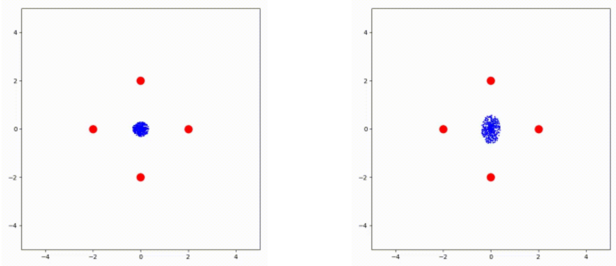
http://nvidia-research-mingyuliu.com/gaugan

Not all is good news

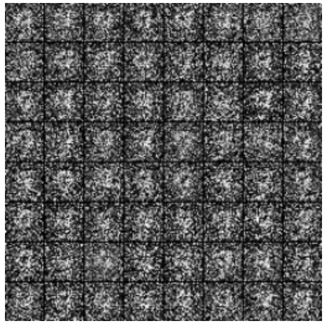
- **Non-convergence:** the model parameters oscillate, destabilize and never converge



- **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



Lipizzaner

From GANs to Deep Neuroevolution

- GAN training can be seen as a **two-player minmax game** (generators $G(z)$ vs discriminators $D(x)$)

- **Evolutionary computing** community has already addressed similar issues in two-player minmax optimization
  - Focusing, relativism or loss of gradient
  - **Competitive Coevolution**, two different populations

    (adds diversity → robustness)

Introduction to Evolutionary Computing

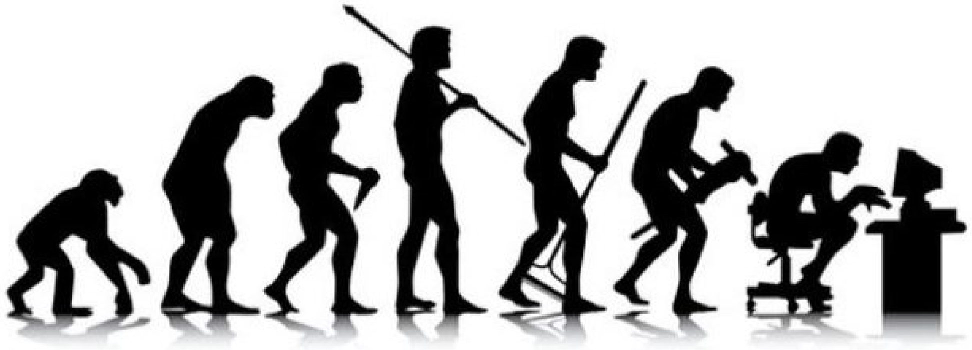
Darwinism:

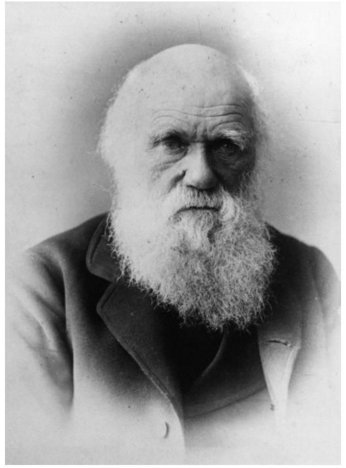
*"I have called this principle, by which, each **slight variation, if useful, is preserved**, by the term of **Natural Selection**. … The expression often used by Mr. Herbert Spencer of the **Survival of the Fittest** is more accurate, and is sometimes equally convenient.“*

Charles Darwin

On the Origin of Species by means of Natural Selection, 1859

Evolution of species through a gradual process of **natural selection**

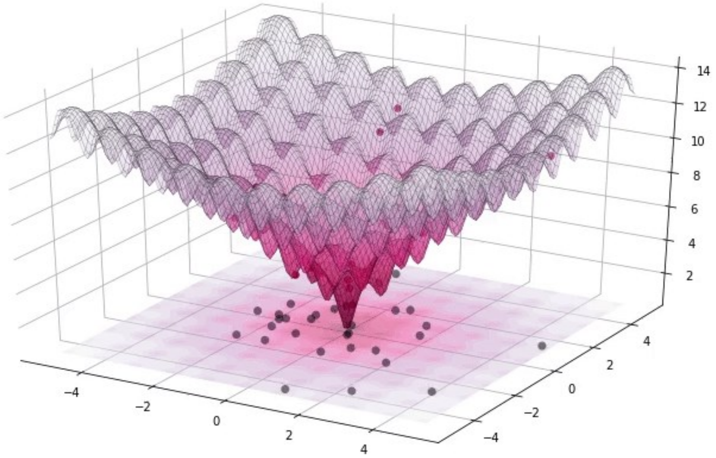




Introduction to Evolutionary Computing

Evolutionary Computing comprises a set of computational methods (*metaheuristics*) that **mimics biological evolution**

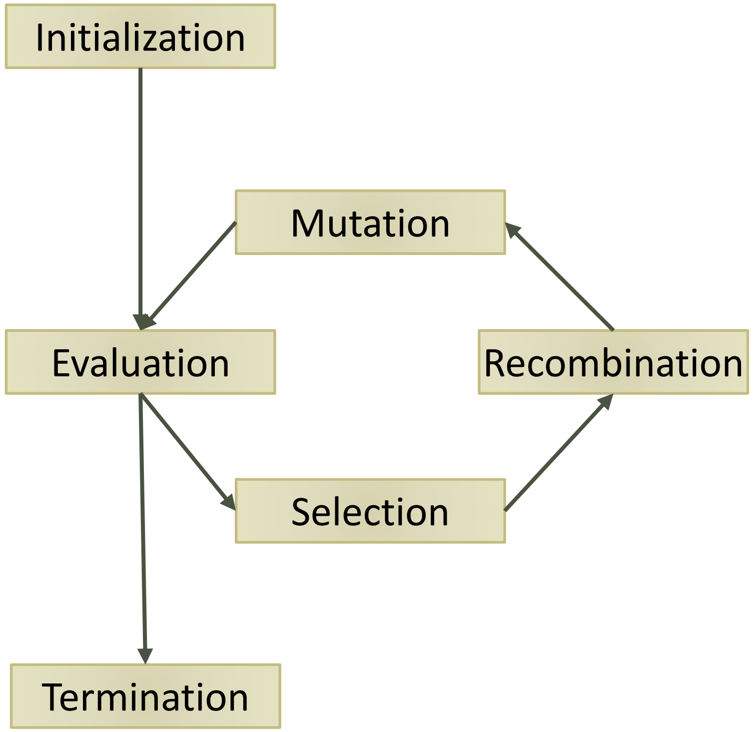
- They apply a mechanism analogous to natural evolutionary processes, to solve **search and optimization** problems
- They work with a **population** (of **representations**) of solutions
- Principles: natural **selection** (**fitness**), **reproduction** (**recombination** and **mutation**) and genetic **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**



Introduction to Evolutionary Computing

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, recombination_probability)
    4. offspring = ***mutate***(parents, mutation_probability)
    5. new_population = ***replace***(offspring, population(generation))
    6. generation++
    7. population(generation) = new_population



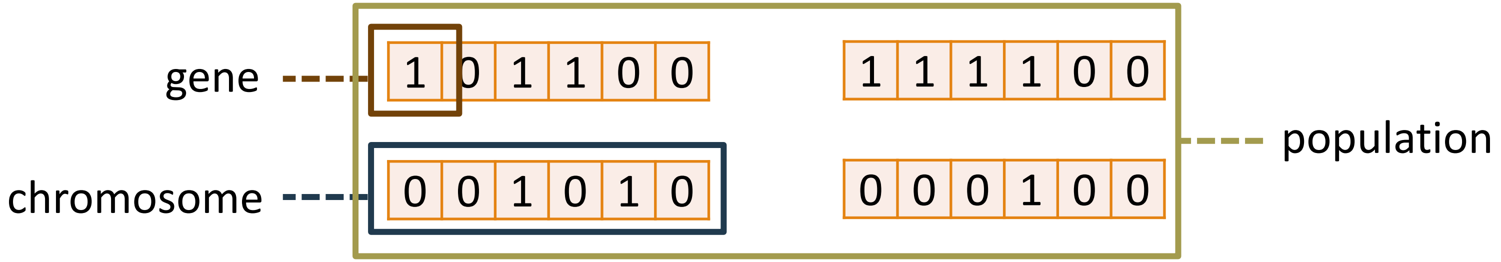
Introduction to Evolutionary Computing

Example: One-Max problem

- Maximizing the number of 1s of a bitstring of length *n* (i.e., composed by 1s and 0s)

Optimum | 1 | 1 | 1 | 1 | 1 | 1 |

- They work with a **population** (of representations) of solutions



- Fitness

$$fitness(x) = \sum_{i=0}^{n-1} x_i$$

Introduction to Evolutionary Computing

Example: One-Max problem

- Maximizing the number of 1s of a bitstring of length *n* (i.e., composed by 1s and 0s)
- They work with a **population** (of representations) of solutions

1	0	1	1	0	0

0	0	1	0	1	0

1	1	1	1	0	0

0	0	0	1	0	0

- Recombination

p1 = 1 0 | 1 1 0 0
p2 = 0 0 | 1 0 1 0
→
o1 = 1 0 | 1 0 1 0
o2 = 0 0 | 1 1 0 0

One-point crossover

p1 = 1 0 | 1 1 | 0 0
p2 = 0 0 | 1 0 | 1 0
→
o1 = 1 0 | 1 0 | 0 0
o2 = 0 0 | 1 1 | 1 0

Two-point crossover

- Mutation

o1 = 0 0 1 0 1 0 → o2 = 0 0 0 0 1 0

Bit flip mutation

o1 = 0 0 1 0 1 0 → o2 = 0 0 0 0 1 1

Swap mutation

Introduction to Evolutionary Computing

1. Initialization

Population

2. Evaluation

fit $(s_0)=3$	1 0 1 1 0 0
fit $(s_1)=1$	0 0 1 0 0 0
fit $(s_2)=4$	1 1 1 1 0 0
fit $(s_3)=1$	0 0 0 1 0 0
fit $(s_4)=2$	0 0 1 0 1 0

average fitness=2.2

3. Selection (tournament selection)

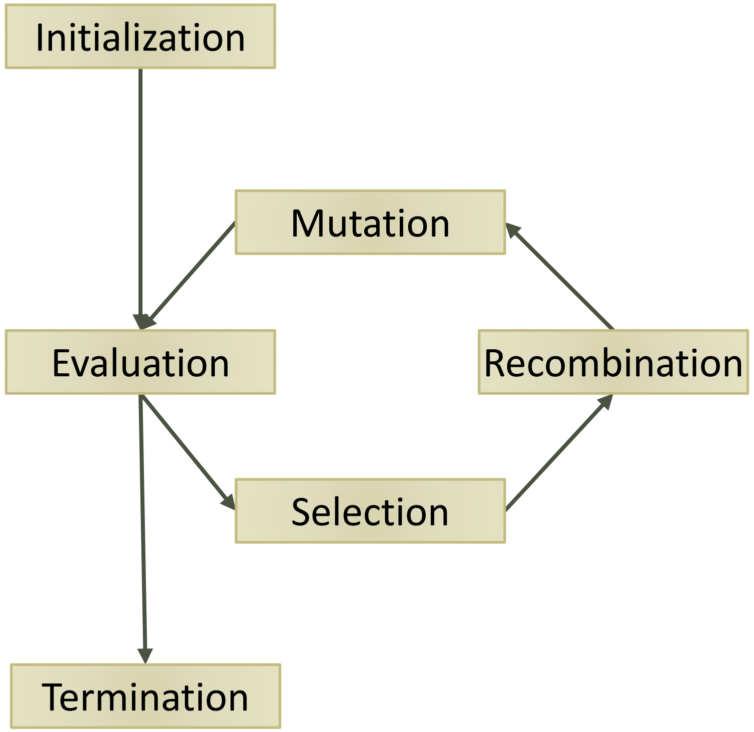
1 0 1 1 0 0 vs 0 0 0 1 0 0 →

0 0 1 0 0 0 vs 0 0 1 0 1 0 →

Parents

p1 = 1 0 1 1 0 0

p2 = 0 0 1 0 1 0



Introduction to Evolutionary Computing

4. Recombination (two-point cx)

Parents

p1 = | 1 | 0 | 1 | 1 | 0 | 0 |

p2 = | 0 | 0 | 1 | 0 | 1 | 0 |

p1 = | 1 | 0 ¦ 1 | 1 ¦ 0 | 0 |

p2 = | 0 | 0 ¦ 1 | 0 ¦ 1 | 0 |

o1 = | 1 | 0 ¦ 1 | 0 ¦ 0 | 0 |

o2 = | 0 | 0 ¦ 1 | 1 ¦ 1 | 0 |

5. Mutation (bit-flip)

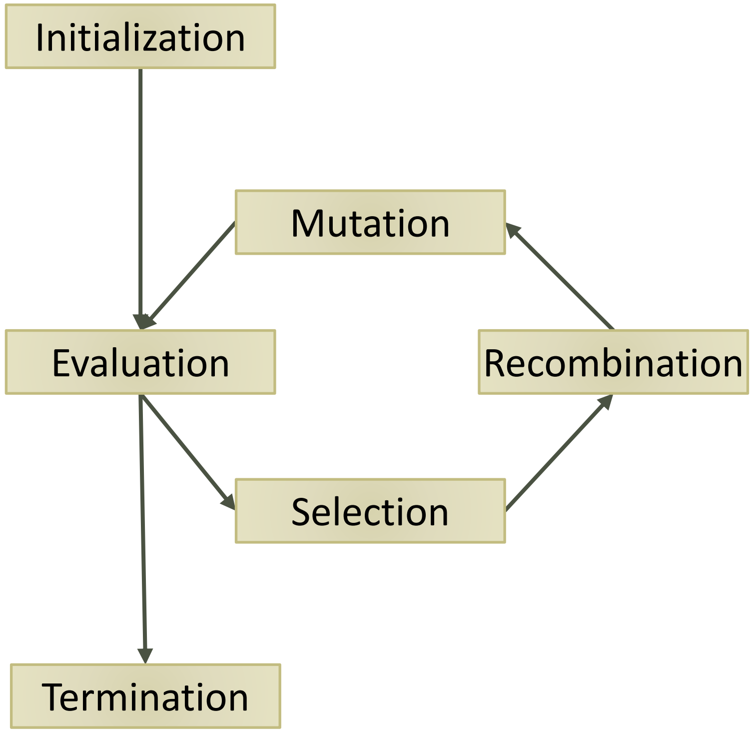
o1 = | 1 | 0 | 1 | 0 | 0 | 0 |

o2 = | 0 | 0 | 1 | 1 | 1 | 0 |

Offspring

o1 = | 1 | 0 | 0 | 0 | 0 | 0 |

o2 = | 0 | 0 | 1 | 1 | 1 | 1 |



Introduction to Evolutionary Computing

6. Replacement (*remove the less fit*)

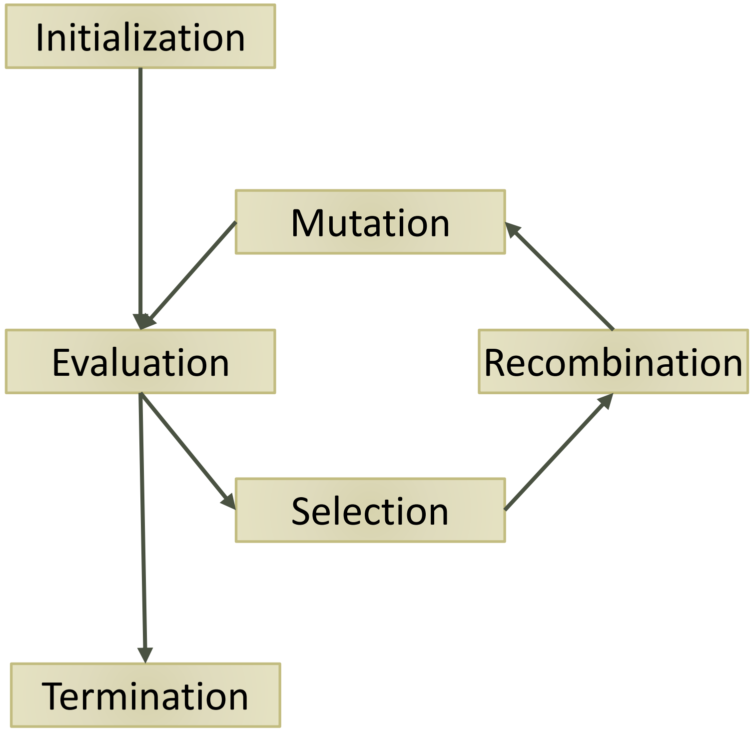
Population

1	0	1	1	0	0
0	0	1	0	0	0
1	1	1	1	0	0
0	0	0	1	0	0
0	0	1	0	1	0

Offspring

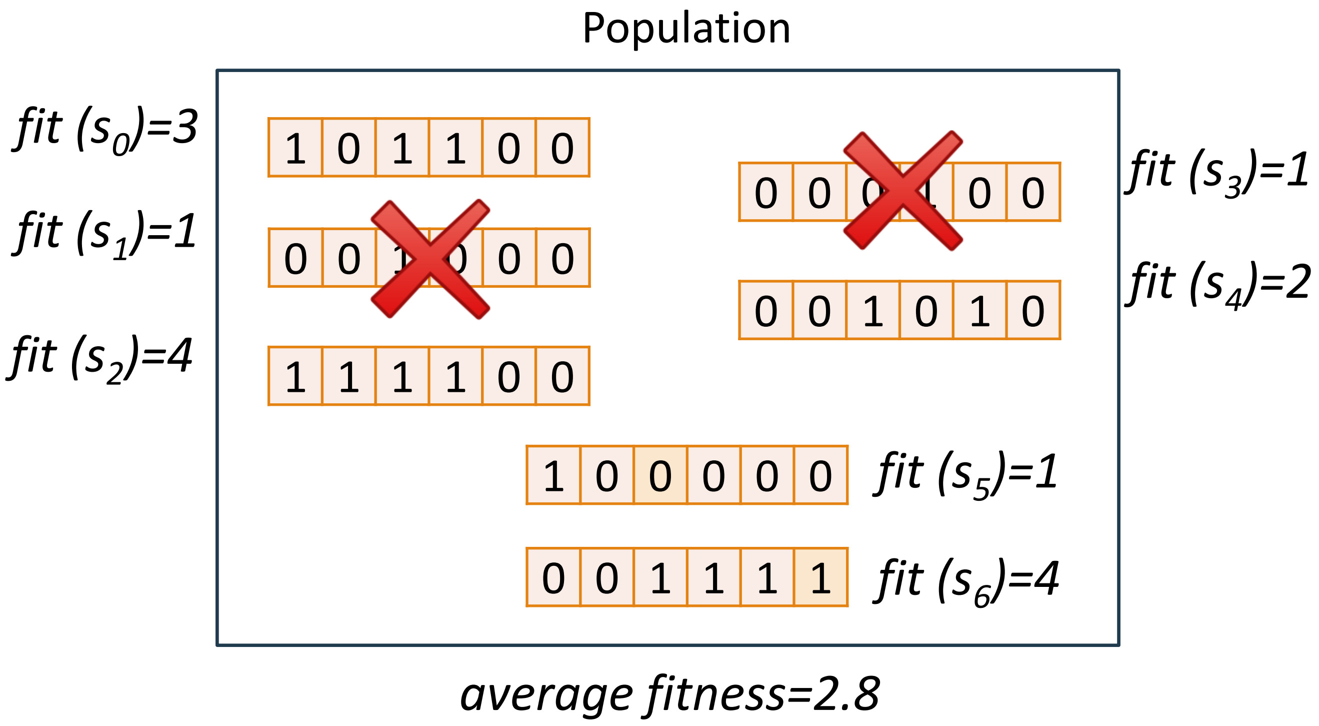
o1 = 1 0 0 0 0 0

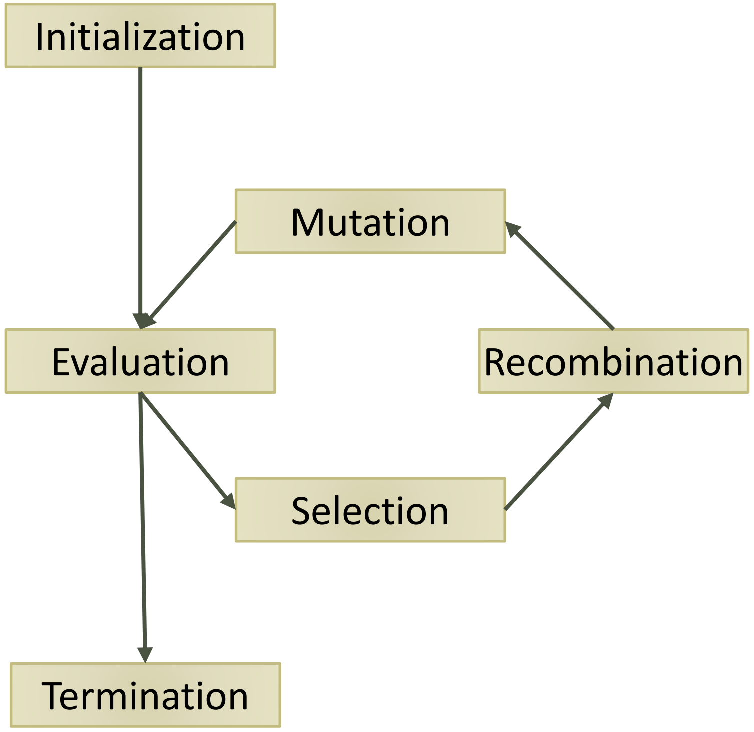
o2 = 0 0 1 1 1 1



Introduction to Evolutionary Computing

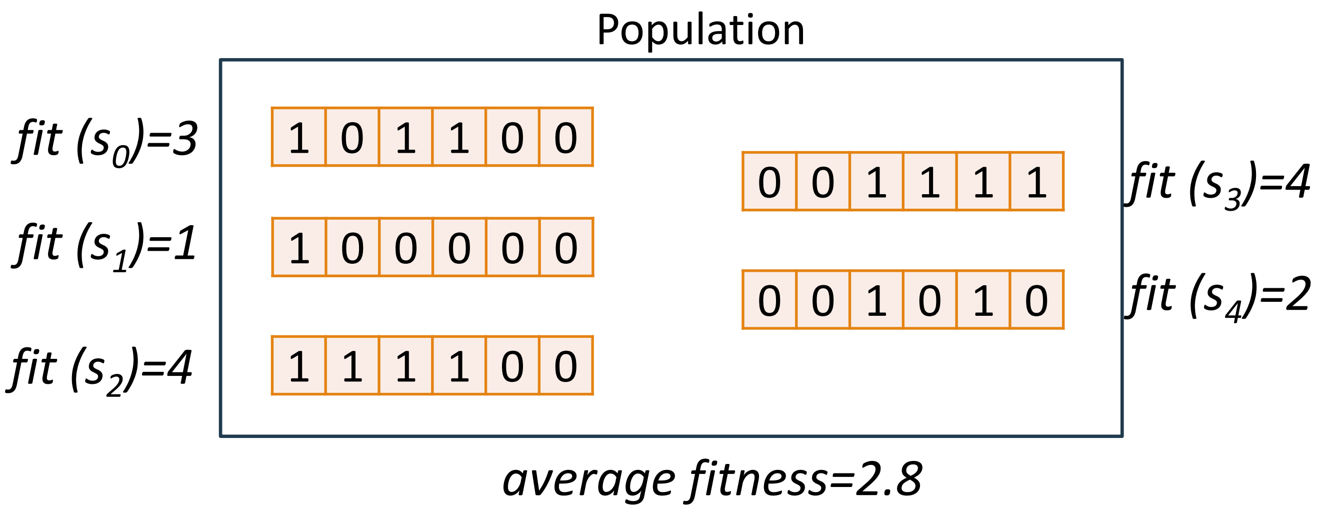
6. Replacement (*remove the less fit*)

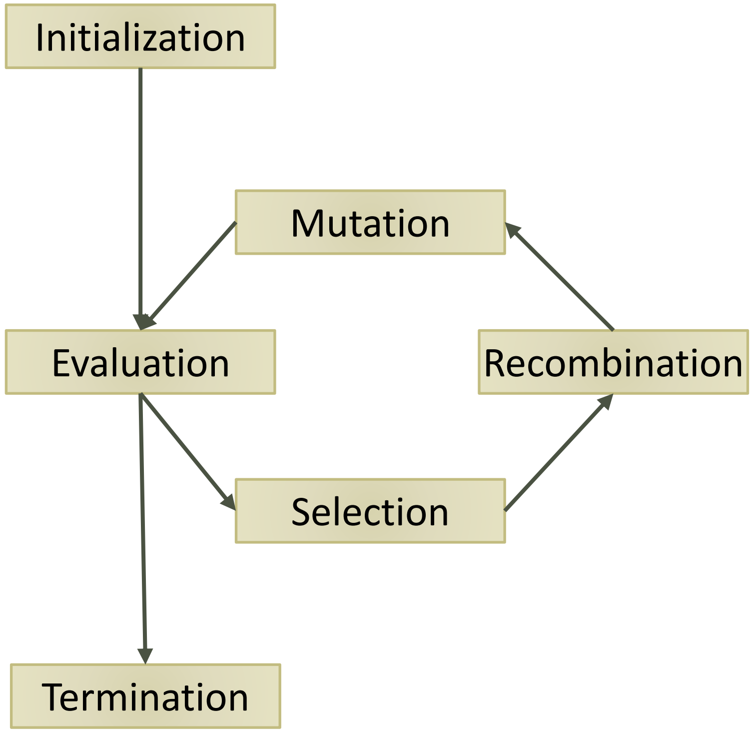




Introduction to Evolutionary Computing

2. Evaluation

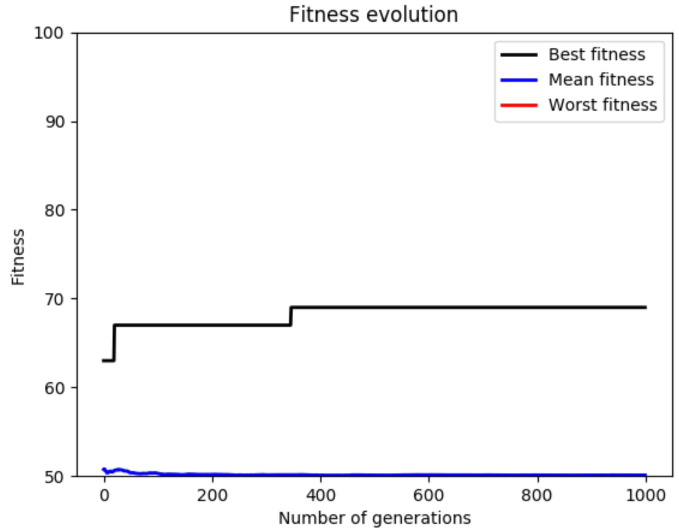




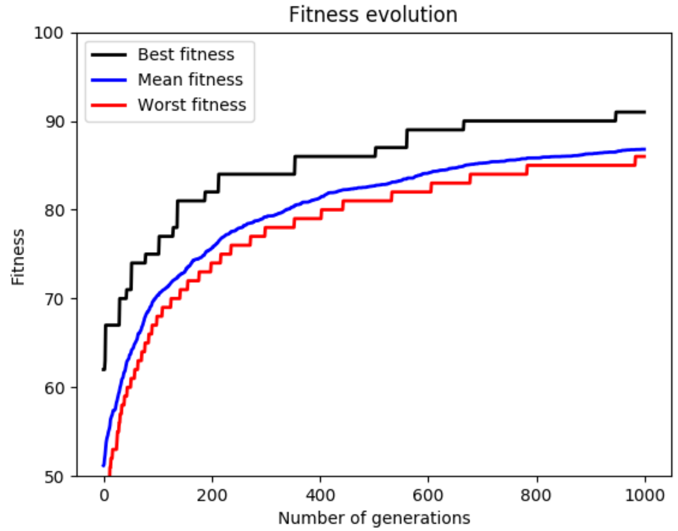
Introduction to Evolutionary Computing

Example: One-Max problem (n=100, 1000 generations, 10 offspring)

2^{100} possible solutions = 1.23×10^{30}



Random search



Evolutionary Algorithm

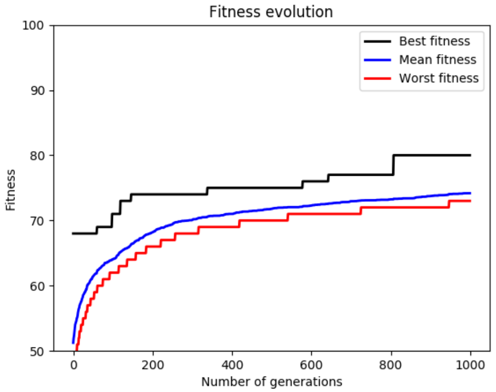
Population size 100
1000 generations
Offspring size 10
Tournament selection
Two-point cx (0.5)
Bit-flip mutation (0.1)

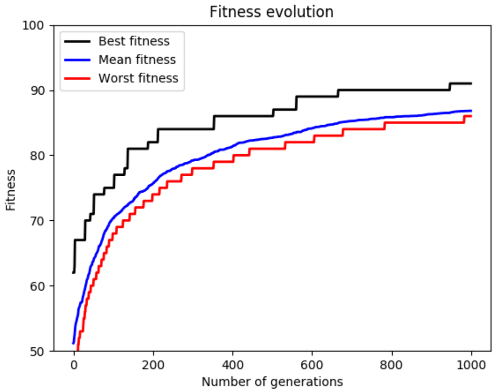
Introduction to Evolutionary Computing

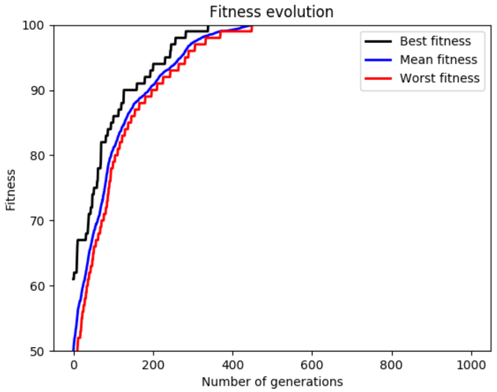
Example: One-Max problem (n=100, 1000 generations, 10 offspring)

mut probability = 0.2 | mut probability = 0.1 | mut probability = 0.01

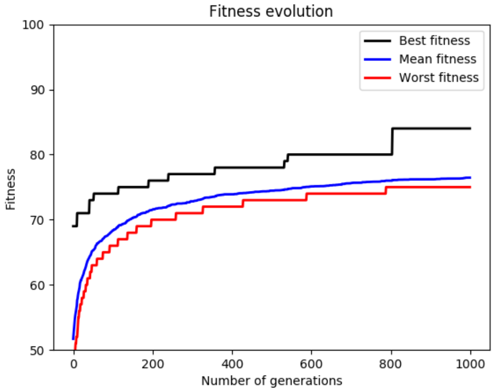
cx probability = 0.5

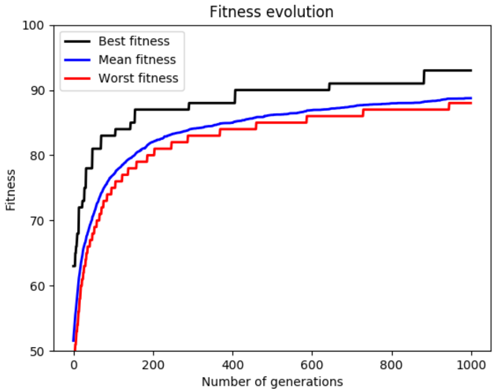


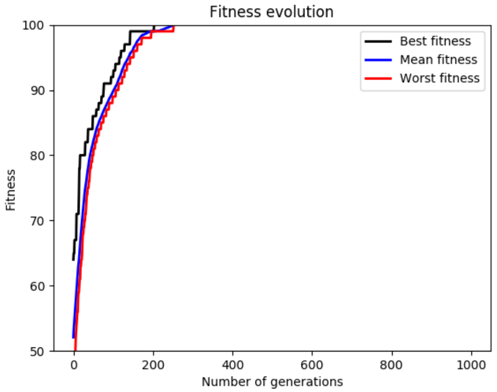




cx probability = 0.9



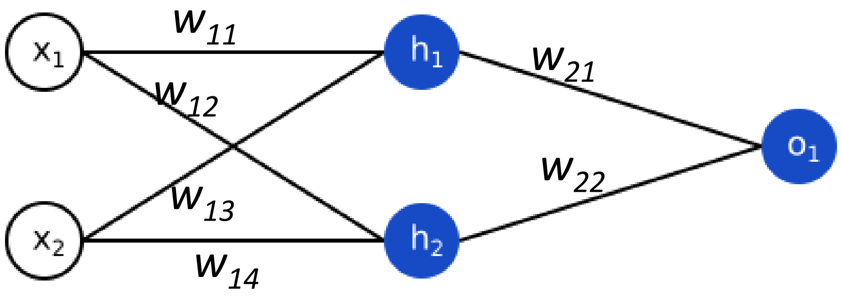




Introduction to Evolutionary Computing

Example: Neuroevolution → Train networks using EA

- Compute the weights to minimize the error (*loss*) of the network



- Representation:

w_{11}	w_{12}	w_{13}	w_{14}	w_{21}	w_{22}

0.21	4.13	-1.57	8.62	-9.02	6.84

$s_i \in [min\_value, max\_value]$

- Fitness: *fitness(s) = loss(s, inputs, outputs)*

Introduction to Evolutionary Computing

Example: Neuroevolution → Train networks using EA

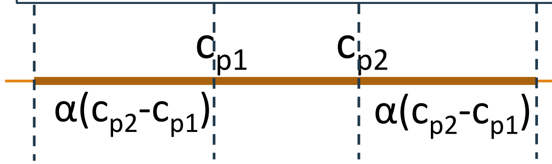
- Compute the weights to minimize the error (*loss*) of the network
- They work with a **population** (of representations) of solutions
- Recombination:

p1 =	*0.21*	*4.13*	*-1.57*	*8.62*	*-9.02*	*6.84*
p2 =	*2.14*	*-1.03*	*7.68*	*-0.98*	*-4.12*	*8.75*

o1 =	*0.21*	*4.13*	*-1.57*	*-0.98*	*-4.12*	*8.75*
o2 =	*2.14*	*-1.03*	*7.68*	*8.62*	*-9.02*	*6.84*

One-point crossover

α is a constant between (0, 1), generally α=0.5
For two genes of two parents c_{p1} and c_{p2} ($c_{p1} < c_{p2}$)
BLX-α selects a random value for chromosomes of offspring c_{o1} and c_{o2} in the range $[c_{p1} - \alpha(c_{p2}-c_{p1}), c_{p1} + \alpha(c_{p2}-c_{p1})]$



1.13	*-2. 11*	*1.26*	*2.81*	*-6.62*	*5.14*

BLX-α crossover

- Mutation:

  Swap mutation swaps to gens in the chromosome
  Random mutation changes a chromosome by a random value between Є [min_value, max_value]
  Gaussian mutation adds a value given by a gaussian distribution

Introduction to Evolutionary Computing

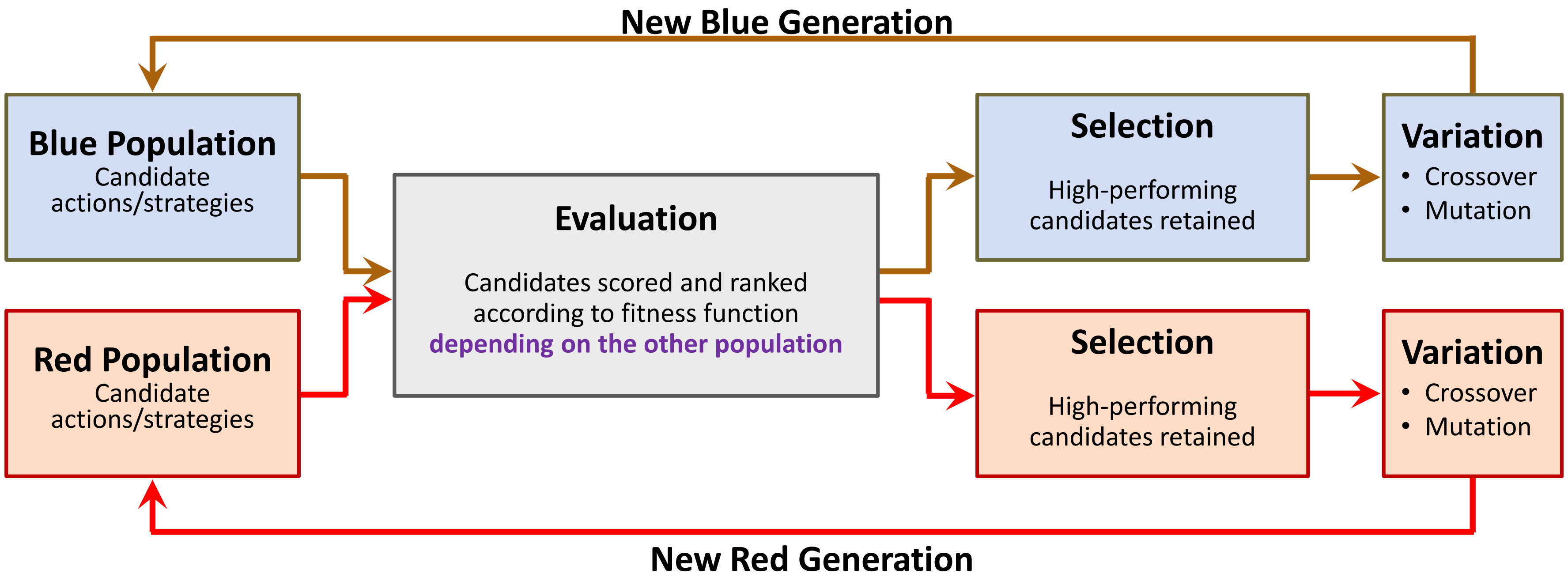
Example: Neuroevolution → Find the best network architecture

A. Camero, J. Toutouh, D.H. Stolfi, E. Alba **Evolutionary Deep Learning for Car Park Occupancy Prediction in Smart Cities** *International Conference on Learning and Intelligent Optimization, LION 12*, pp. 1-15, 2018.

A. Camero, J. Toutouh, J. Ferrer, E. Alba. **Waste generation prediction under uncertainty in smart cities through deep neuroevolution.** *Revista de Ingeniería, Universidad de Antioquia*, No.93, pp. 128-138, 2019.

Competitive Coevolution

In biology, **coevolution** occurs when two or more species **reciprocally affect** each other's evolution through the process of natural selection.



Competitive Coevolution

In biology, **coevolution** occurs when two or more species **reciprocally affect** each other's evolution through the process of natural selection.

- Video games AI
- Cybersecurity

U. O'Reilly, J. Toutouh, M. Pertierra, D. Prado-Sanchez, D. Garcia, A. Erb-Luogo, J. Kelly, E Hemberg (2019). **Adversarial Genetic Programming for Cyber Security: A Rising Application Domain Where GP Matter**s. *Genetic Programming and Evolvable Machines* (In Press).

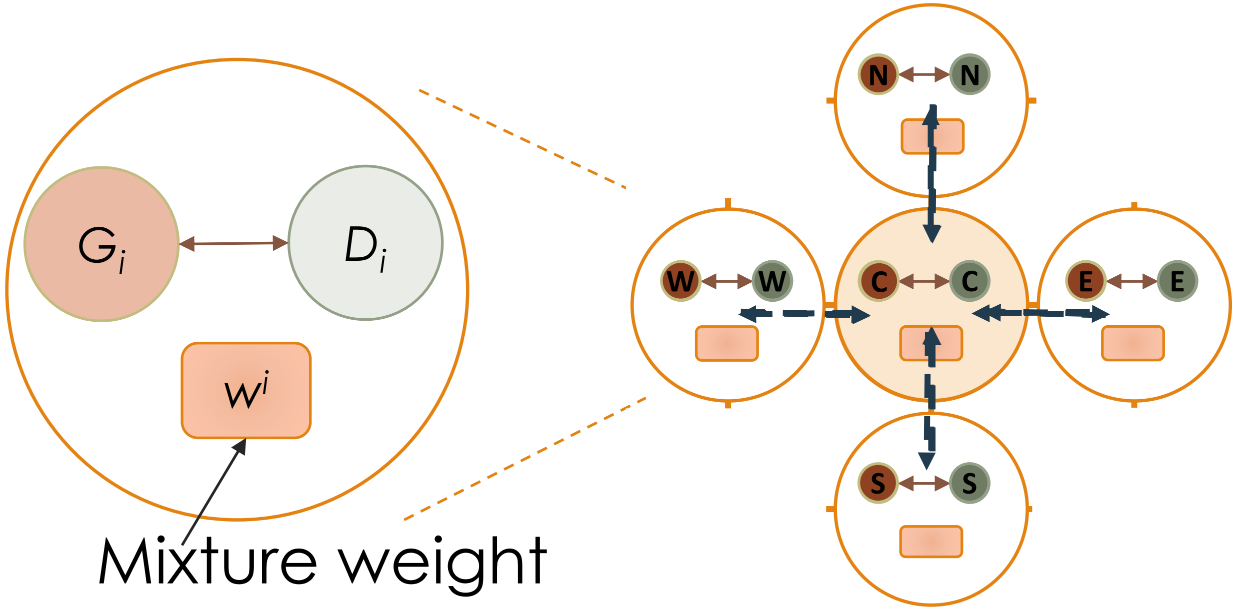
Lipizzaner: Gradient-based Coevolution

- **Non-convergence**
- **Mode collapse**
- **Diminished gradient**

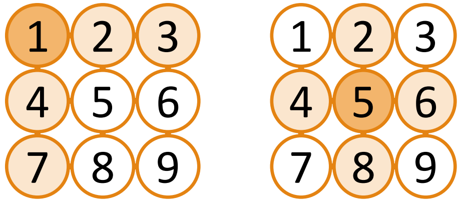
Lipizzaner

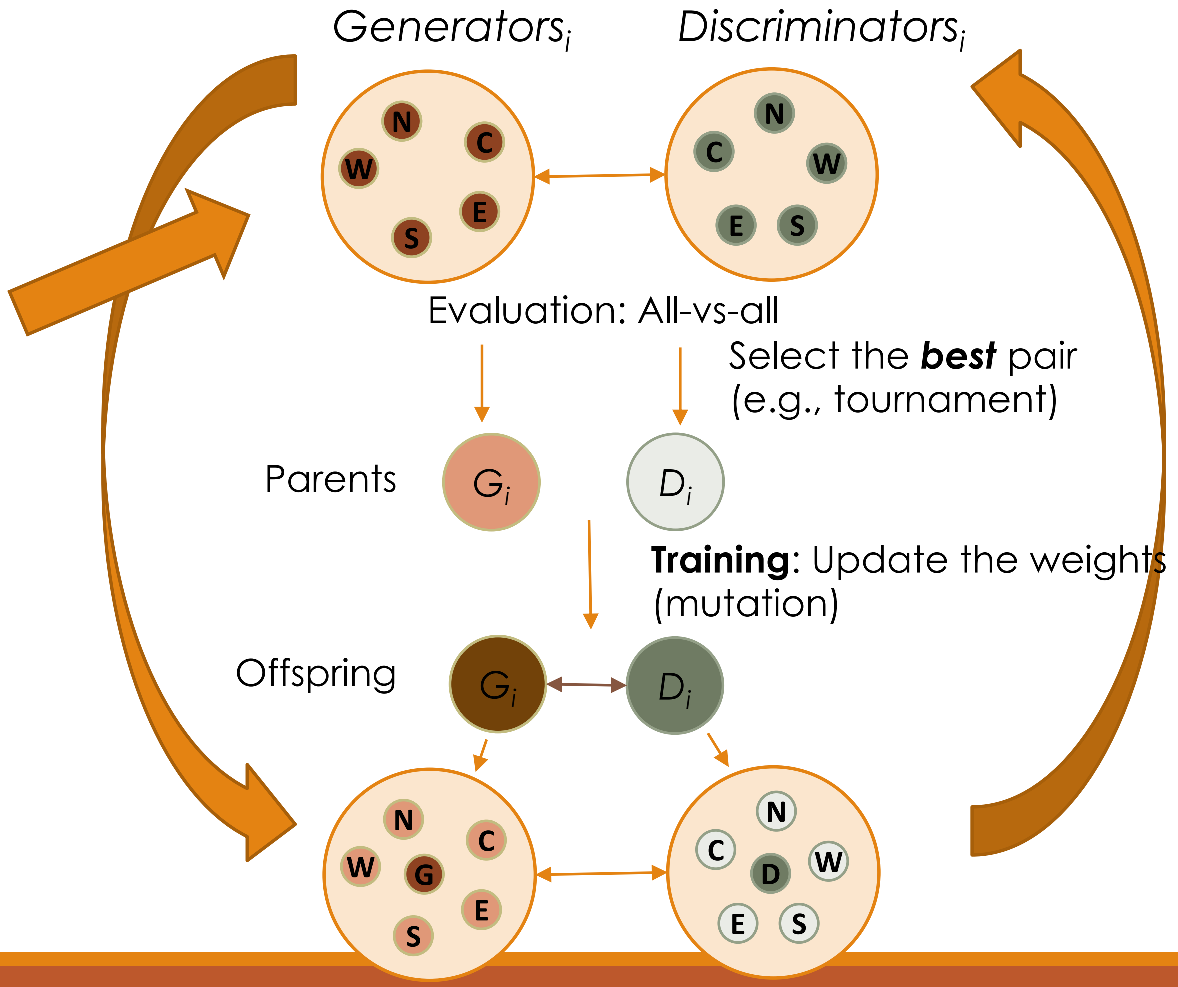
Lipizzaner: Gradient-based Coevolution

A **distributed**, **coevolutionary** framework to train GANs with **gradient-based** optimizers



Neighborhood examples





ALFA
ANYSCALE LEARNING FOR ALL

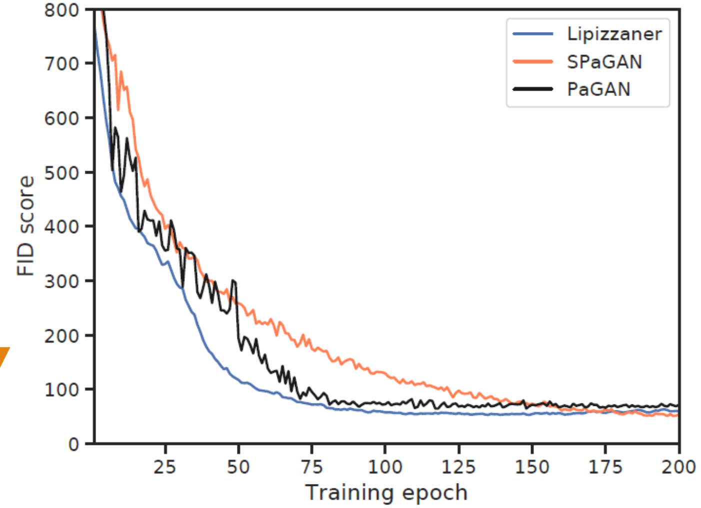
Lipizzaner: Gradient-based Coevolution

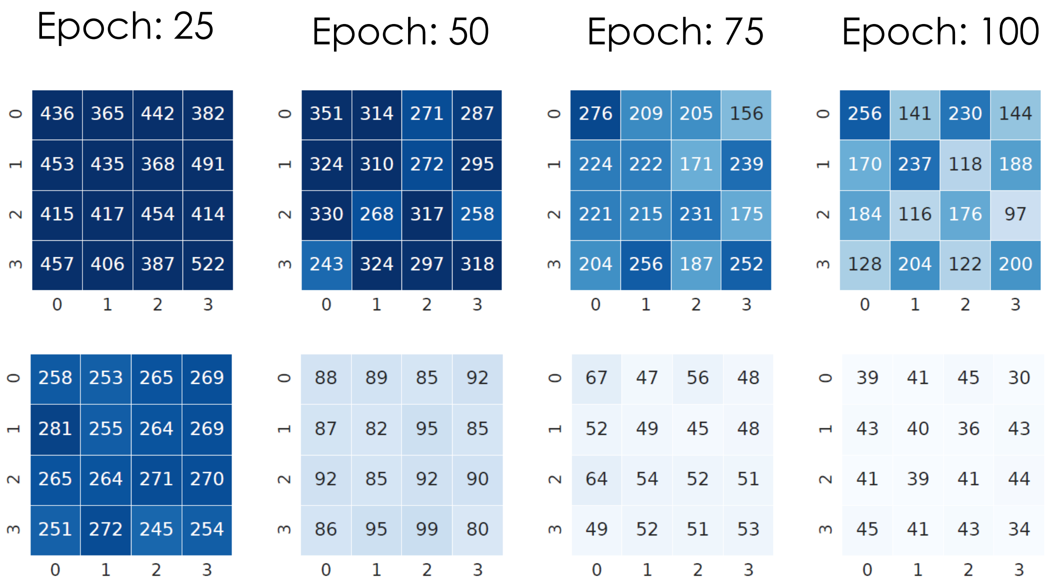
- Lipizzaner is a **distributed**, **coevolutionary** framework to train GANs with **gradient-based** optimizers
  - **Fast convergence** due to gradient-based steps
  - **Improved convergence** due to hyperparameter evolution
  - **Robustness** and **resilience** due to coevolution
  - **Diverse solutions** due to mixture evolution
  - **Scalability** due to spatial distribution topology and asynchronous parallelism

Fast and improved convergence

- **Lipizzaner → communication** and **performance-based selection** pressure converge faster and avoids continuous fluctuations
- **SPaGAN – no selection/replacement**
- **PaGAN – no communication**

FID→Accuracy
Lower is better



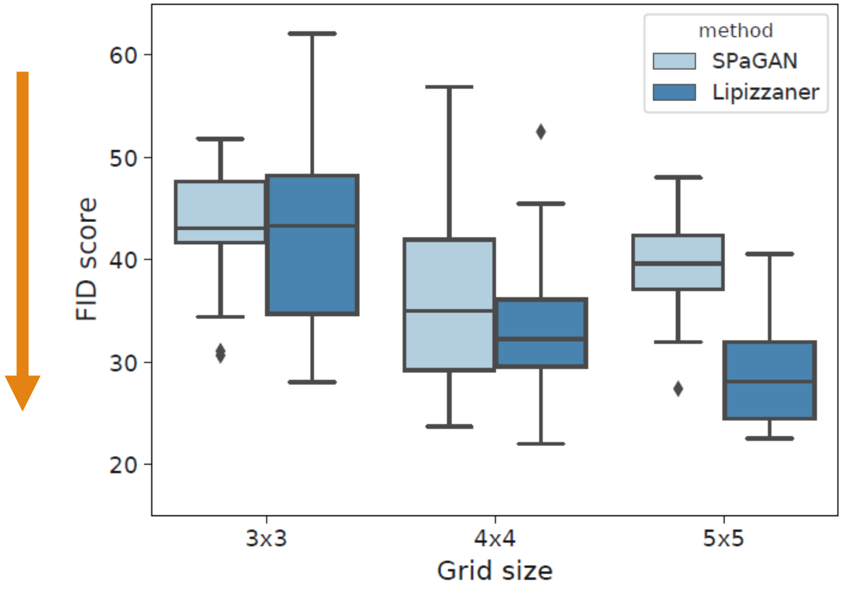


Fast and improved convergence

- As the grid size increase (larger populations), the Lipizzaner converges to better generative models

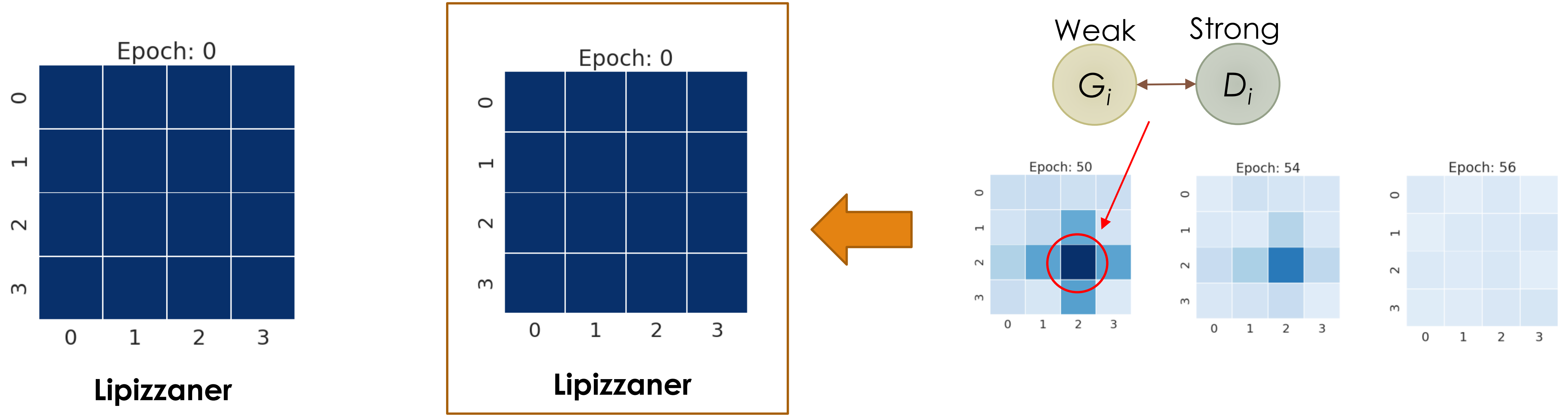
Final results (FID)

FID→Accuracy
Lower is better



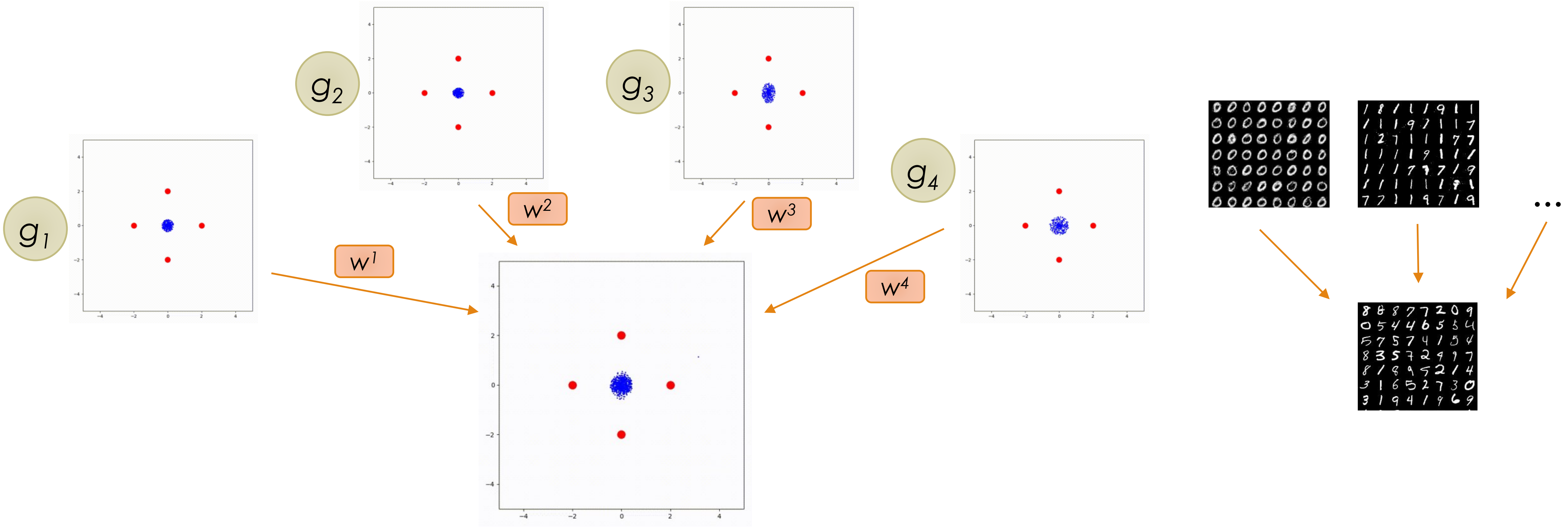
Robustness and resilience

- **Competitive coevolution** allows the cells to escape from local optima and addresses **vanishing gradient issues**



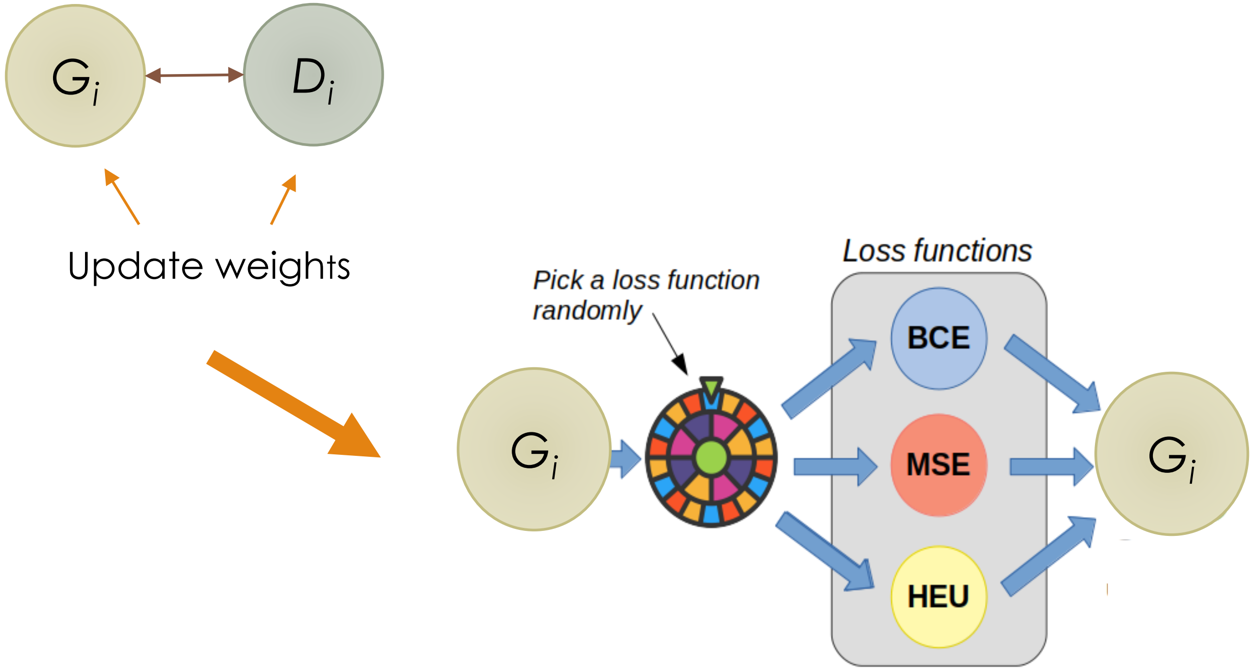
Ensembles: Robustness

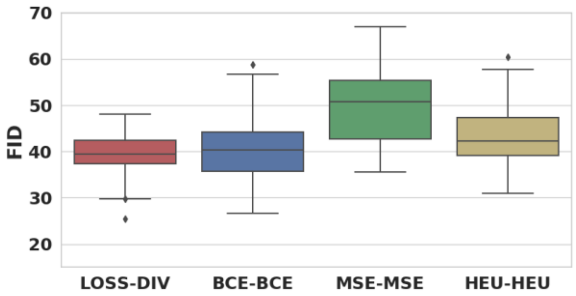
- **Mixture** of generators overcomes **mode collapse**



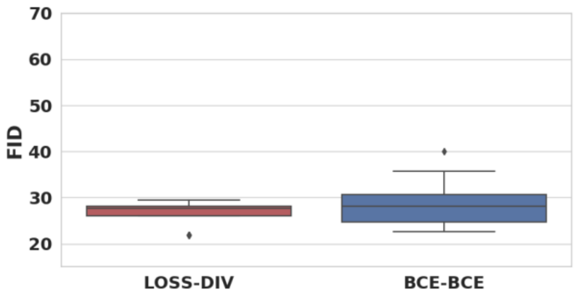
Loss Diversity: Robustness

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





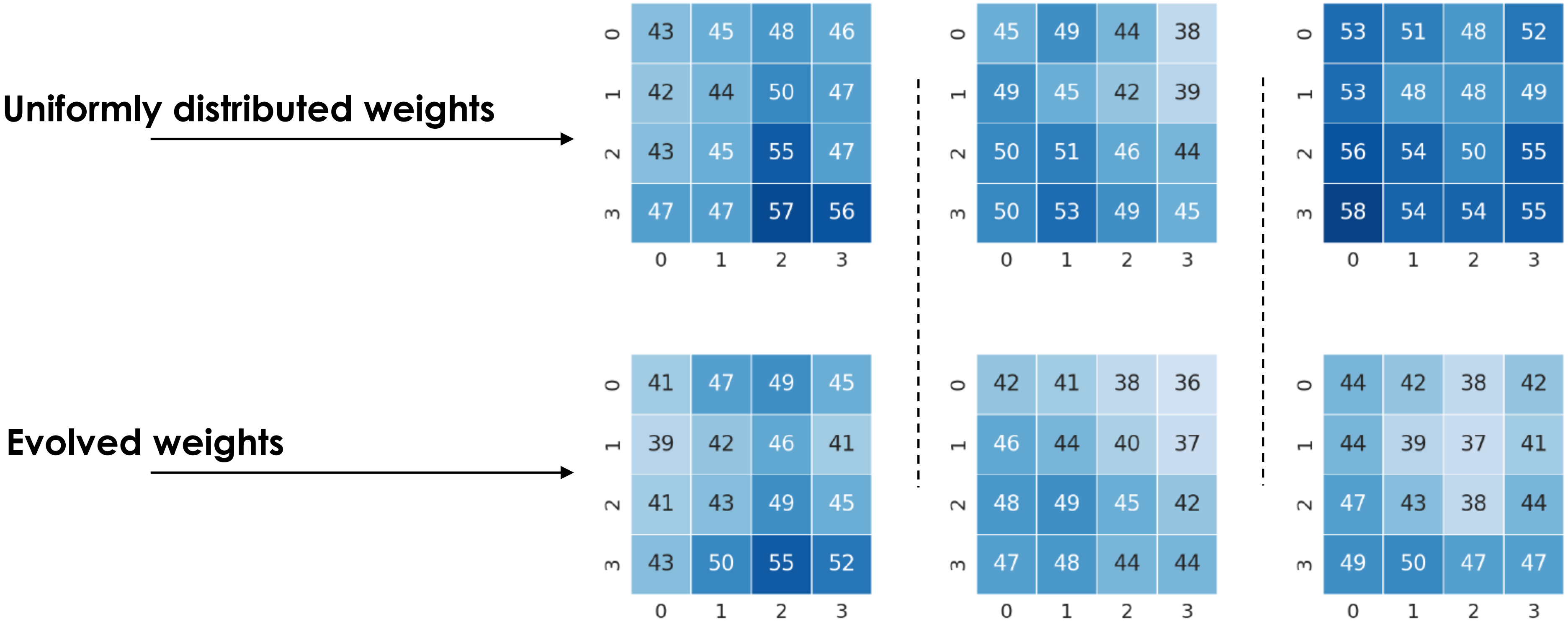
3x3 grid



5x5 grid

Ensembles: Better results

- The use of **evolved ensembles** improves **quality** of the samples



Ensembles: Re-purpose models

- Given a set of heterogeneous generators that were optimized for one objective (e.g., FID), create **ensembles for optimizing a different objective**

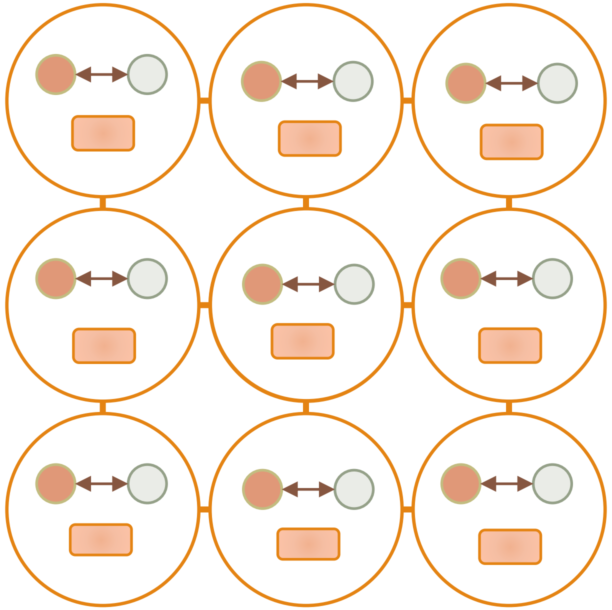
High quality
Low diversity

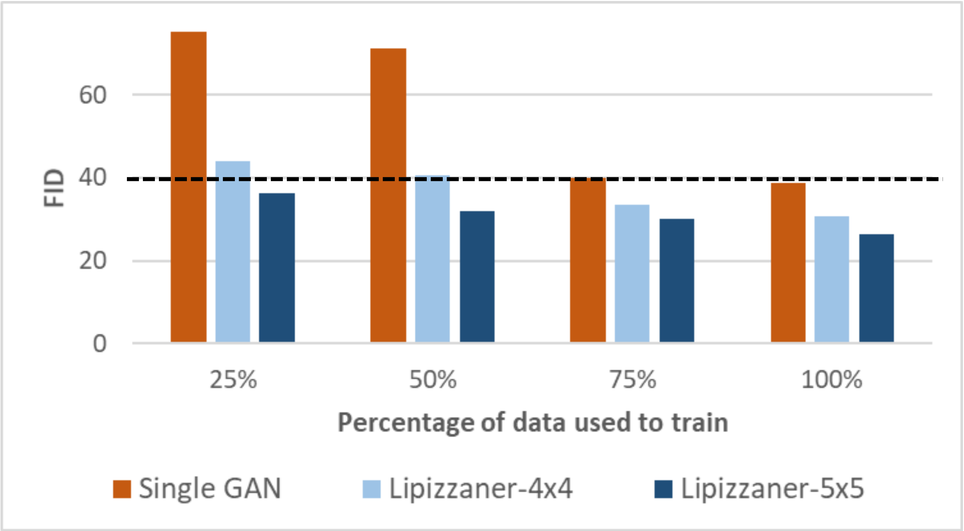
High quality
High diversity

Ensemble size	TVD - Diversity (new objective)	FID - Accuracy (old objective)
1	*0.113*	*36.393*
3	0.046	27.576
4	0.043	27.890
5	0.046	28.225
6	0.045	27.077
<8	**0.033**	**27.342**

Improvements: Data Dieting

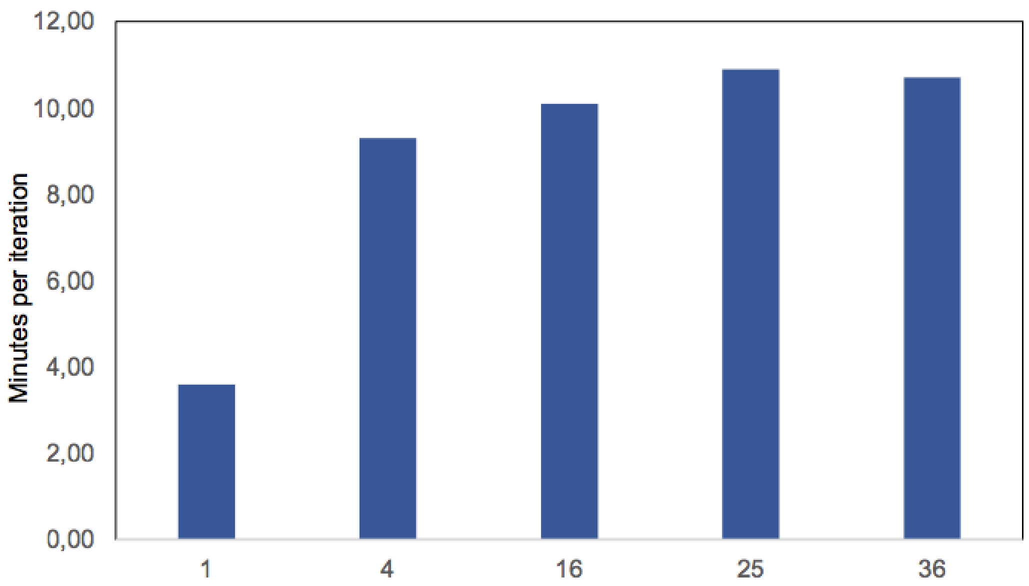
- As we have communication between cells, do we need to replicate whole data among all the cells? **Data diversity**





Improvements: Data Dieting

- **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.



Some implemented approaches

- Conditional GAN
- Wasserstein GAN
- Deep Convolutional GAN
- Semi-supervised learning
- Temporal (LSTM) based generative models

Publications

Lipizzaner: A System That Scales Robust Generative Adversarial Network Training. E. Hemberg, A. Al-Dujaili, T. Schmiedlechner, U. O'Reilly. *Systems for Machine Learning workshop@* ***NeurIPS 2018***.

An Artificial Coevolutionary Framework for Adversarial AI. Una-May O’Reilly, Erik Hemberg. ***AAAI Fall Symposia***, 2018.

Towards Distributed Coevolutionary GANs. Abdullah Al-Dujaili, Tom Schmiedlechner, Erik Hemberg, Una-May O'Reilly. ***AAAI Fall Symposia***, 2018.

Spatial Evolutionary Generative Adversarial Networks. Jamal Toutouh, Erik Hemberg, Una-May O'Reilly. **GECCO, 2019**.

Data Dieting in GAN Training. Jamal Toutouh, Erik Hemberg, Una-May O'Reilly. ***Deep Neural Evolution: Deep Learning with Evolutionary Computation*** (*2020*).

Re-purposing Heterogeneous Generative Ensembles with Evolutionary Computation. Jamal Toutouh, Erik Hemberg, Una-May O'Reilly. ***GECCO, 2020.***

Parallel/distributed implementation of cellular training for generative adversarial neural networks. E. Perez, S. Nesmachnow, J. Toutouh, E. Hemberg, U. O'Reilly. ***PDCO 2020***

Selection Pressure and Communication in Evolutionary GAN Training. Jamal Toutouh, Erik Hemberg, Una-May O'Reilly. ***PPSN 2020*** (*Under review*).

Spatial Coevolution for the Robust and Scalable Training of Generative Adversarial Networks. Erik Hemberg, Jamal Toutouh, Una-May O'Reilly. ***ACM Transactions on Evolutionary Computing*** (*Under review*).

Massachusetts Institute of Technology

Thanks! Comments?

JAMAL TOUTOUH
jamal@lcc.uma.es
jamal.es
@jamtou

UNIVERSIDAD DE LA REPÚBLICA URUGUAY