Deep generative neural networks Fundamentals & problem solving

Class 2

JAMAL TOUTOUH

jamal@uma.es

jamal.es

@jamtou

Artificial Neurons. Implementation

Language: python

- High-level, general-purpose, interpreted programming language.
- Dynamically-typed and garbage-collected.
- Includes a comprehensive standard library and many auxiliary libraries.
- Supports multiple programming paradigms: structured (procedural), object-oriented, and functional programming.
- Tensors: particular data structures (extend vectors and matrices). Represented using n-dimensional arrays.

ANNs/Deep Learning

ANN results get better with more/better **data,** bigger **models,** more **computation effort**

- PyTorch library:
  - Deep learning framework/library for python, developed by Facebook.
  - Has own data structures that provide automatic operations on tensors.
- Tensorflow library:
  - Deep learning framework/library for python, developed by Google.
  - Particularly focused on training and inference of deep ANNs.

Implementation. Google colab

- Provides a workspace for machine learning on the cloud, with an environment based on Jupyter Notebooks + Python.
  - Jupyter Notebooks: web-based interactive computational environment for open source software development.
- Provides free computing resources (virtual machine with GPU), a significant improvement over local development/execution environment.
- Easy integration with Google Drive storage and github.
- Available at colab.research.google.com

Artificial Neural Networks. DNN

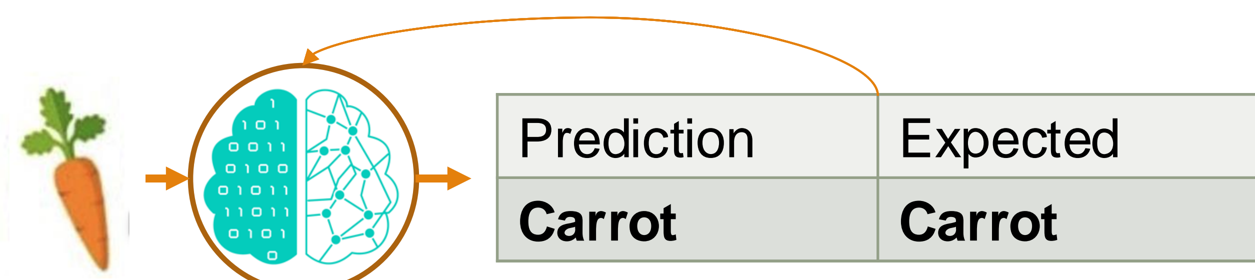
- Deep neural networks employ deep architectures in neural networks.
- “Deep” refers to functions with higher complexity in the number of layers and units in a single layer.

- Three following types of deep neural networks are popularly used today:
  - **Multi-Layer Perceptrons (MLP)**
  - **Convolutional Neural Networks (CNN)**
  - **Recurrent Neural Networks (RNN)**

Generative Models

Learning Paradigms

Labeled data



Prediction	Expected
Carrot	**Carrot**

Carrot

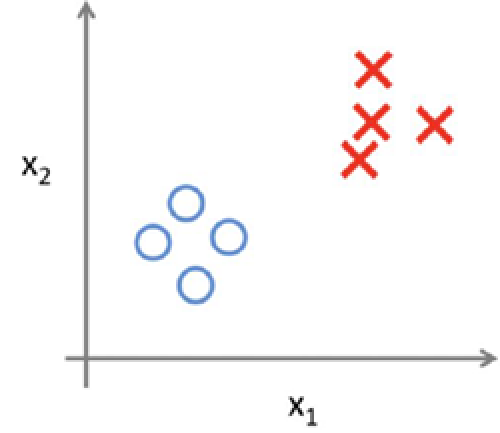
Tomato

Pepper

Supervised vs Unsupervised Learningx

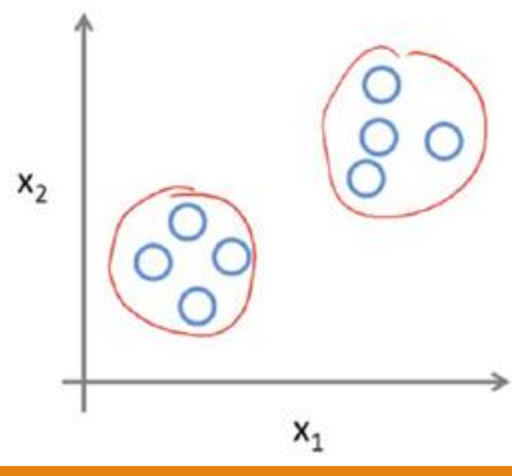
Supervised Learning

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



Unsupervised Learning

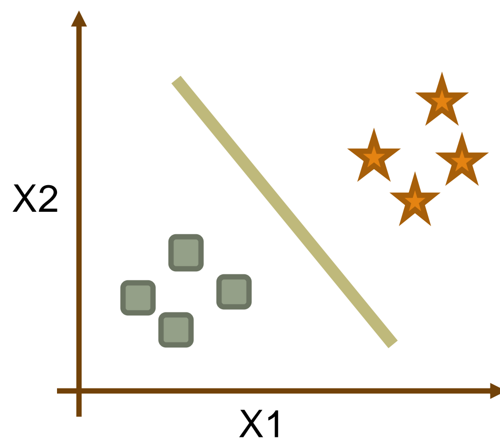
- Given data *x*
- Goal: Learn the *hidden or underlying structure* of the data
- Requires **data (no labels)**
- Methods: Clustering/Density, Compression



Learning Paradigms

Supervised Learning

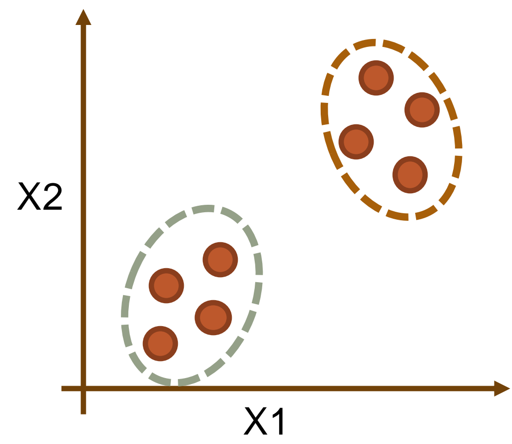
- The process of learning a function that **maps** input data (x) to output (y) using labeled examples
- Requires **labeled data:** Each input (x) is paired with a known output (y)
- Goal: Minimize error in predicting y for new, unseen data
- Applications & techniques:
    - Classification, Regression, Object Detection, Segmentation



Learning Paradigms

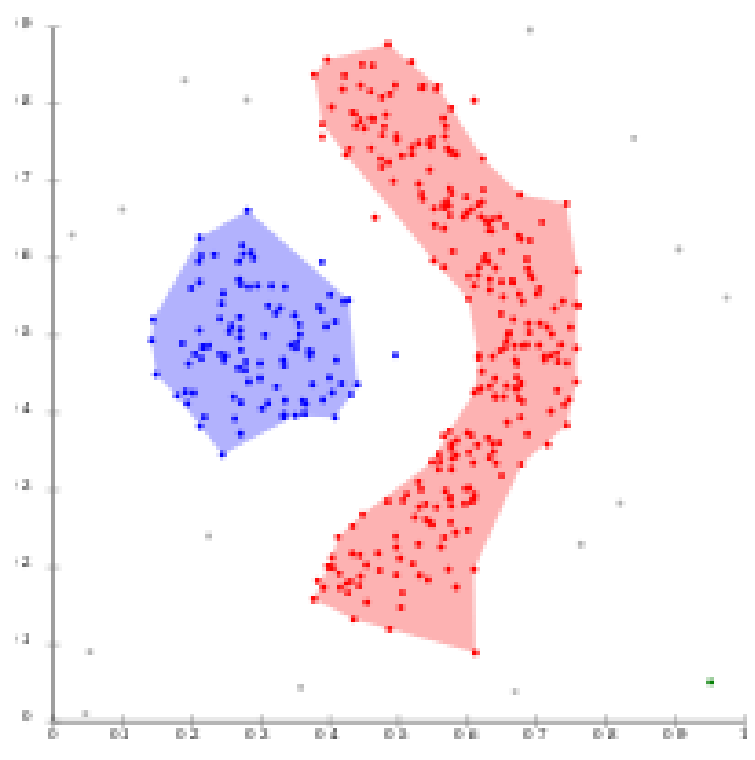
Unsupervised Learning

- A learning approach where the model explores unlabeled data (x) to **identify patterns or structures**
- Requires (**unlabeled**) **data:** No explicit outputs (y) are provided
- Goal: Discover hidden patterns, relationships, or structures in the data
- Applications & techniques:
  - Clustering, Dimensionality Reduction, Anomaly Detection

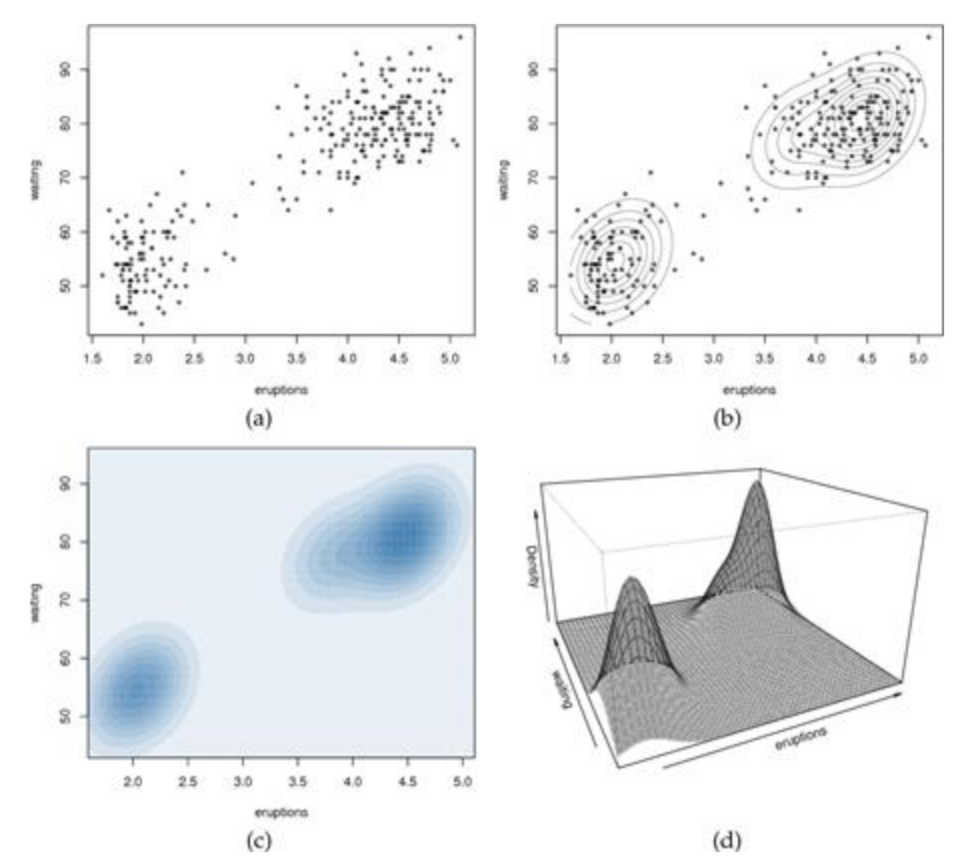


Unsupervised Learning

Clustering



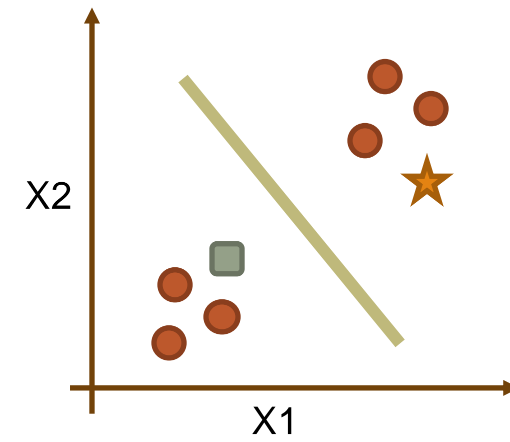
Density estimation



Learning Paradigms

Semi-supervised Learning

- The process of learning a function that **maps** input data (x) to output (y) combining **labeled and unlabeled** samples
- Requires **labeled & unlabeled data:** Some explicit outputs (y) are provided
- Goal: Improve learning accuracy while reducing the need for labeled data
- Applications & techniques:
  - Clustering, Dimensionality Reduction, Anomaly Detection

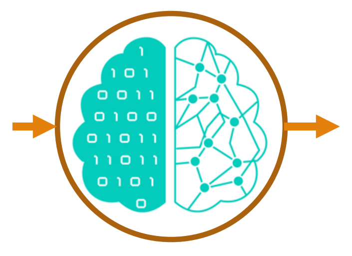


Learning Paradigms

In generative adversarial networks…

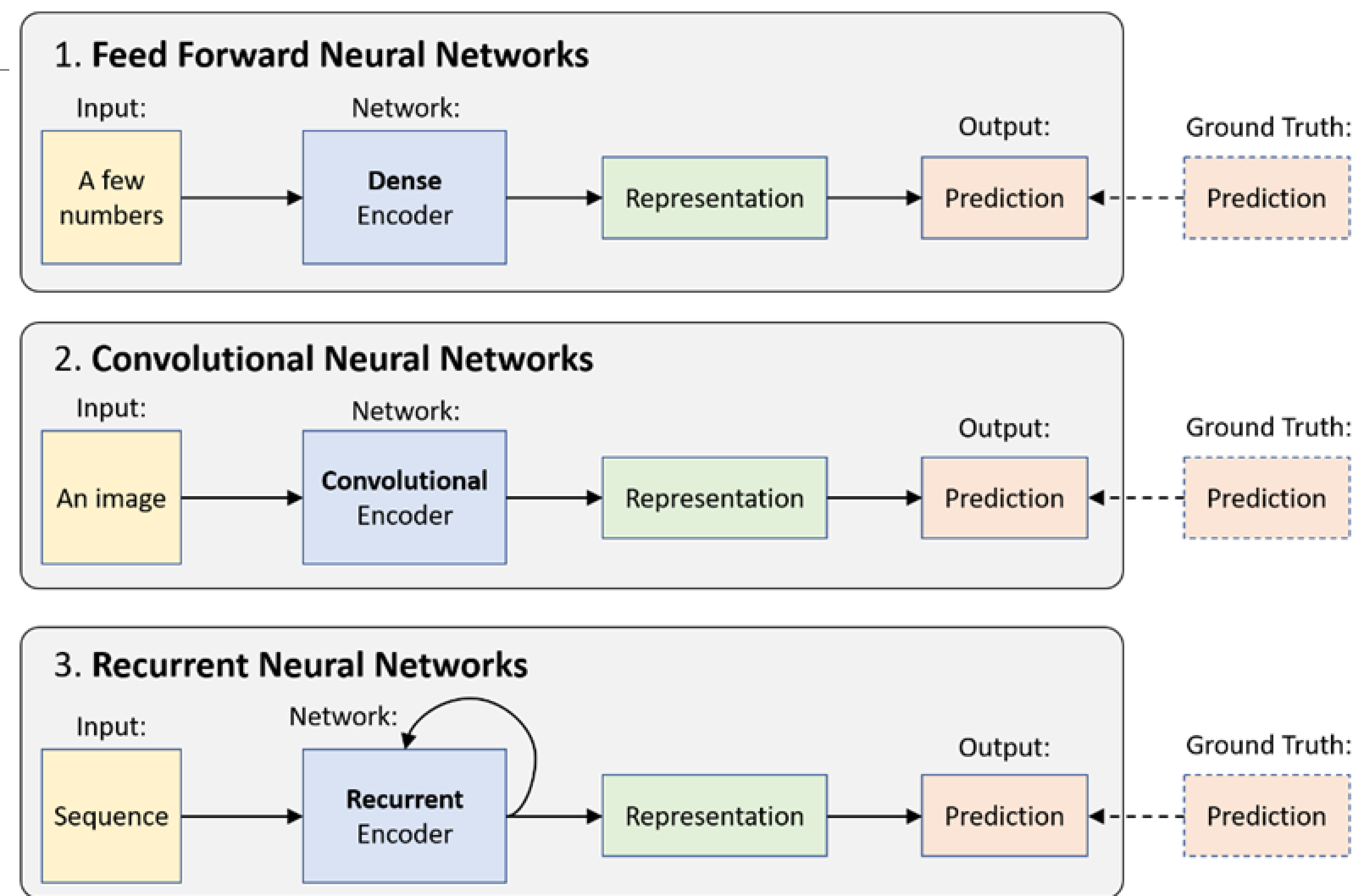
- What learning paradigm can be used?

Training data

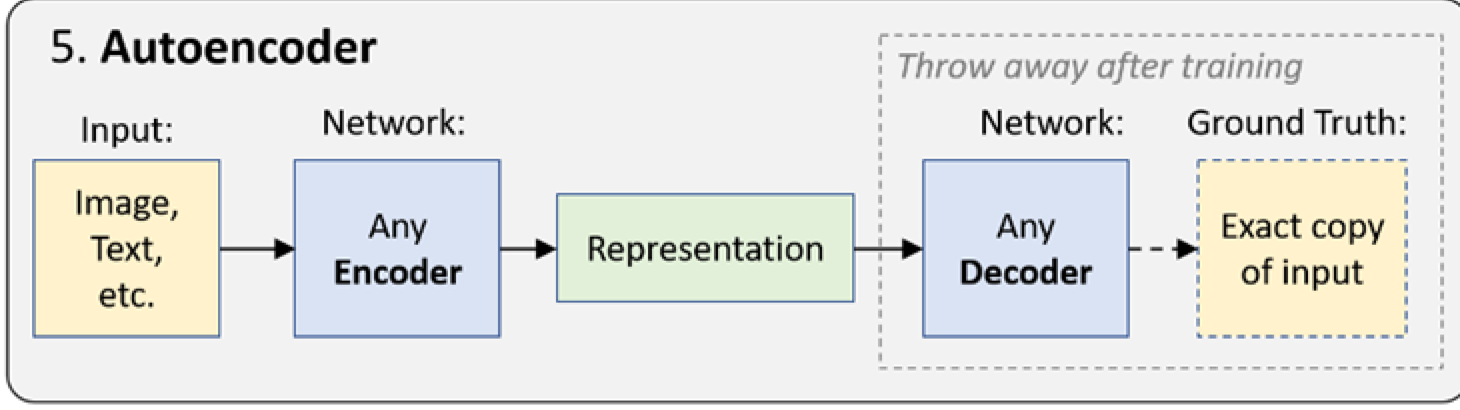


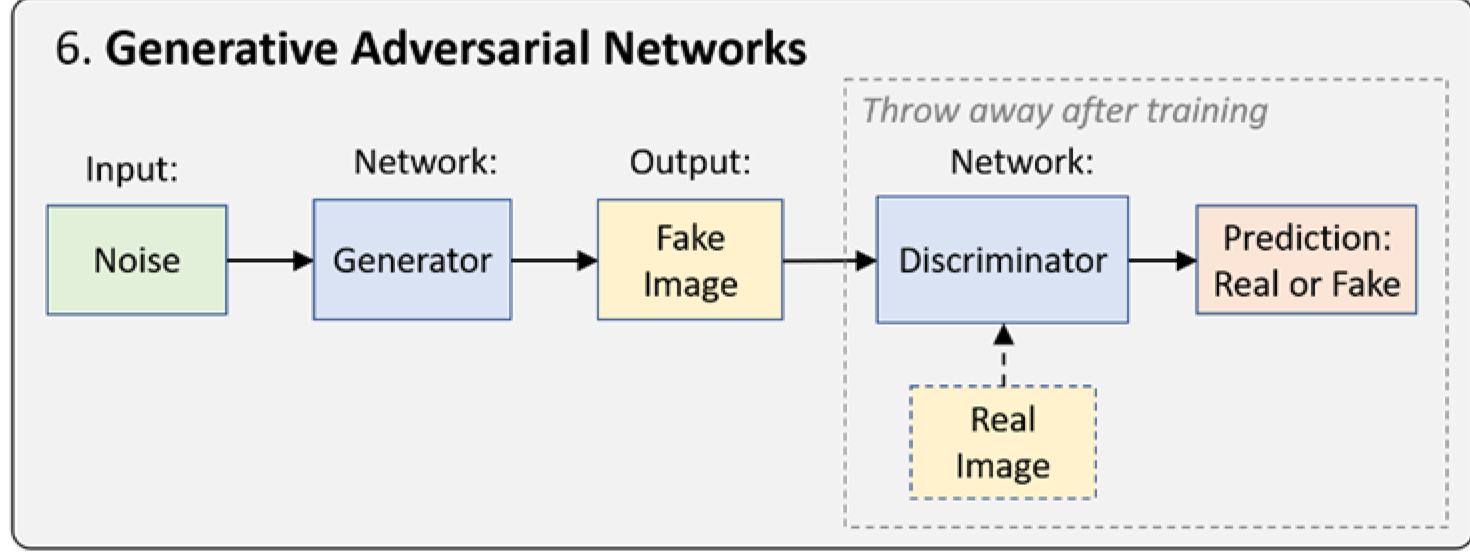
Generated samples

Supervised Learning



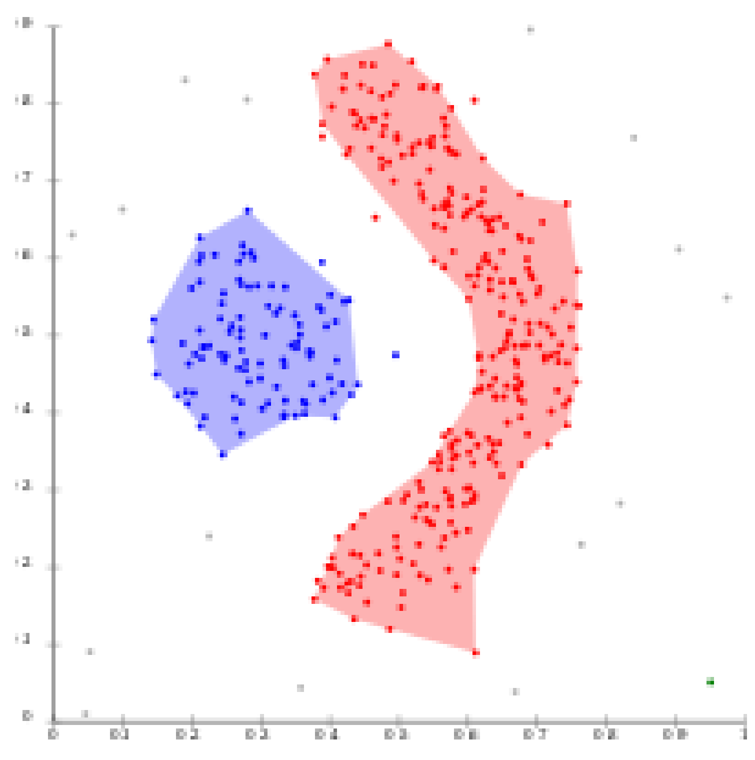
Unsupervised Learning



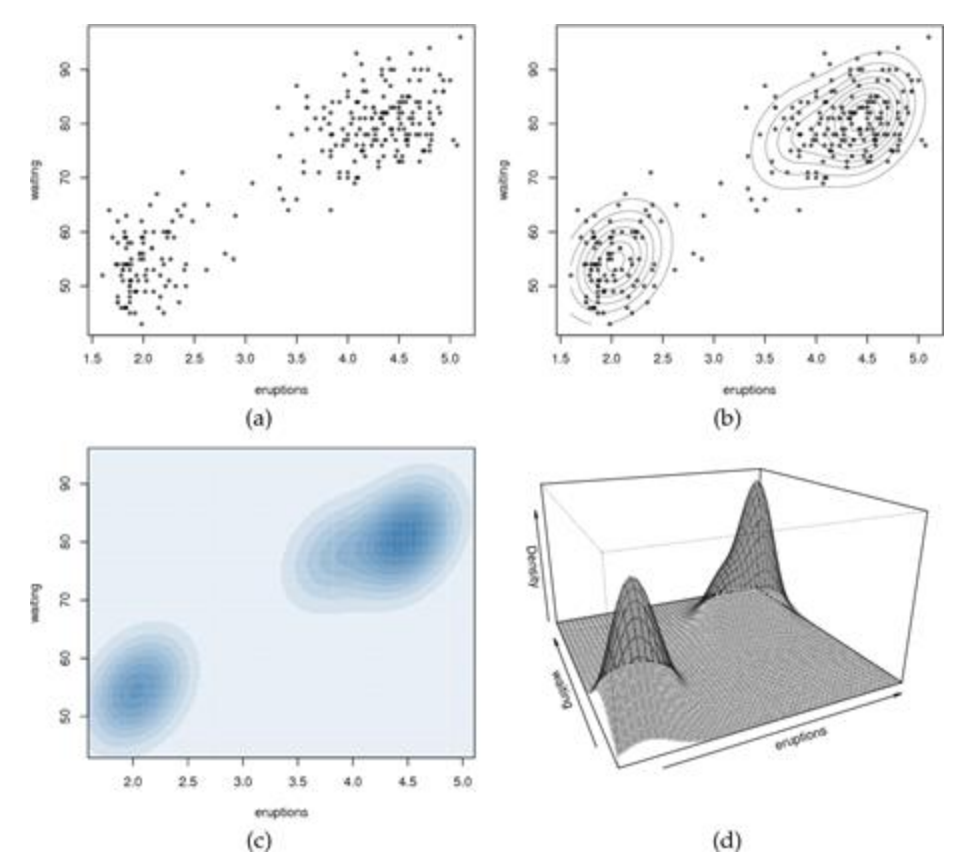


Unsupervised Learning

Clustering



Density estimation



Generative Modeling

Overview: Given a training dataset, generate new samples from same distribution

→ Addressing density estimation

Density estimation: given a bunch of observations from the training dataset p_{data} estimate the probability density function p_{model}

p_{data}

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

Generative Modeling

Overview: Given a training dataset, generate new samples from same distribution

→ Addressing density estimation

Density estimation: given a bunch of observations from the training dataset p_{data} estimate the probability density function p_{model}

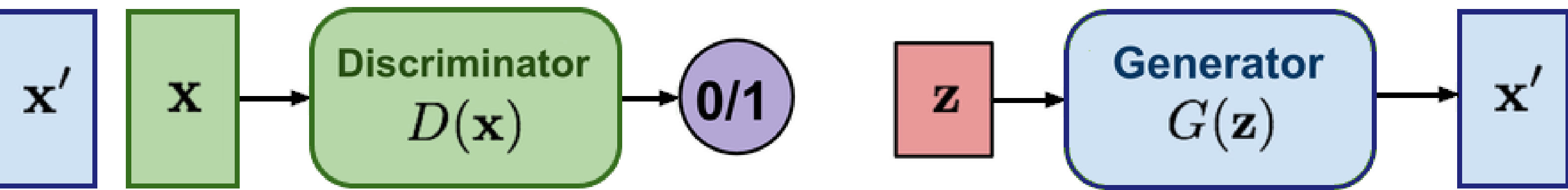
Training dataset ~ p_{data}

Generated samples ~ p_{model}

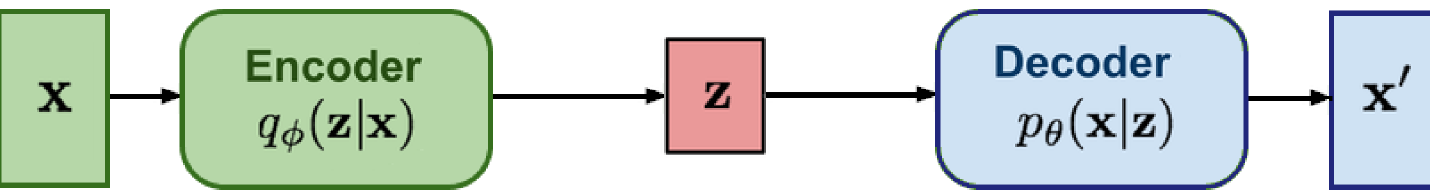
- Understand better the data distribution
- Compress the data representation
- **Generate samples**

Generative Modeling. Models

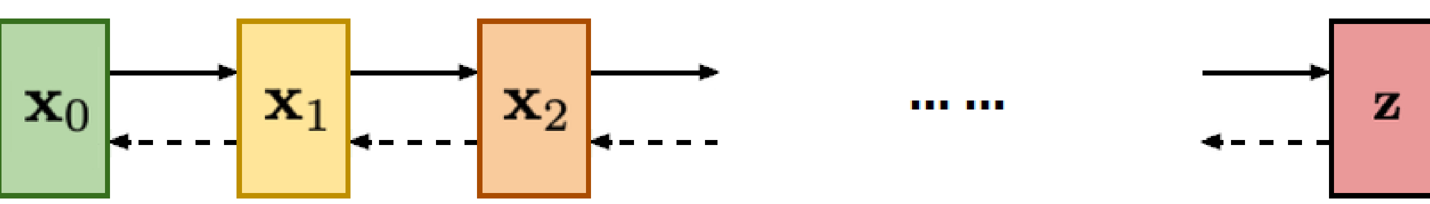
GAN: Adversarial training



VAE: maximize variational lower bound



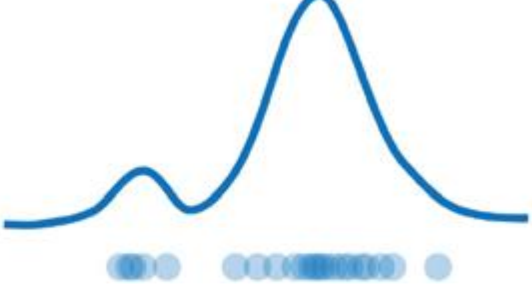
Diffusion models: Gradually add Gaussian noise and then reverse



Generative Modeling

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

- **Density estimation**

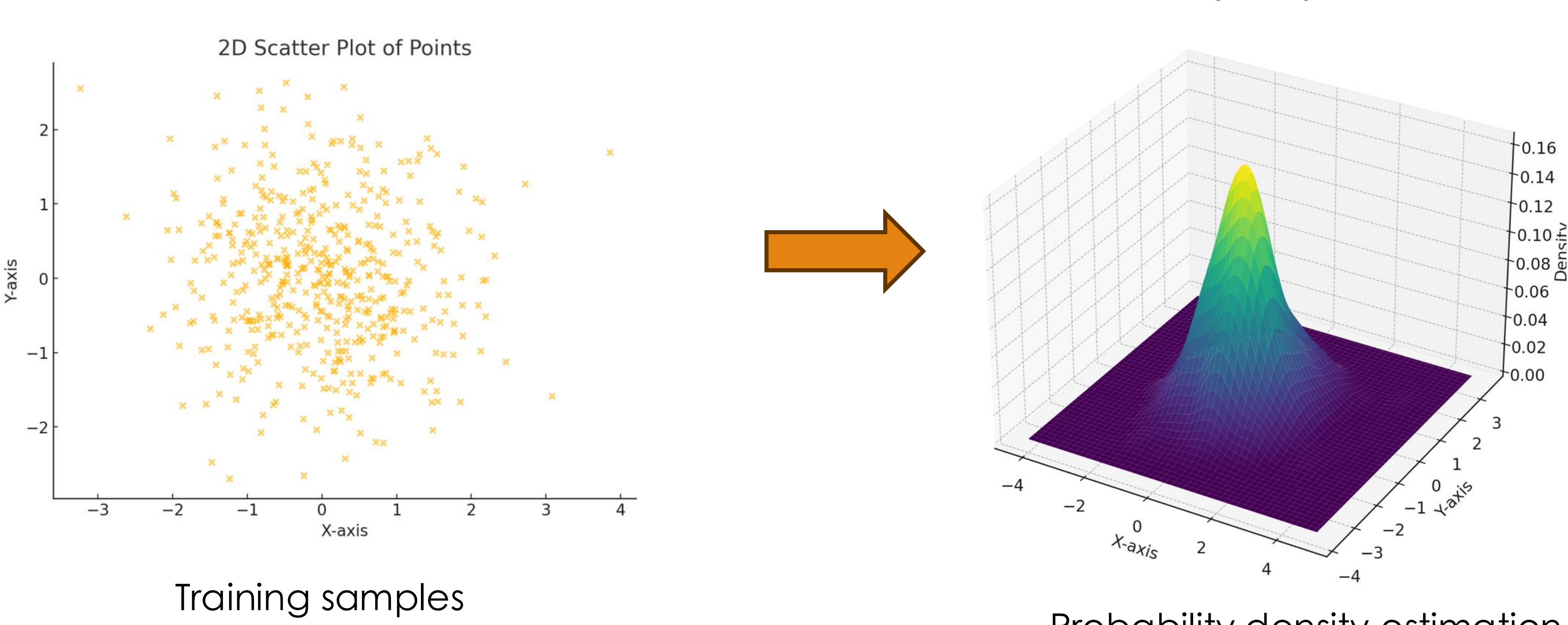


- **Synthetic samples generation**

Training samples

Synthetic samples

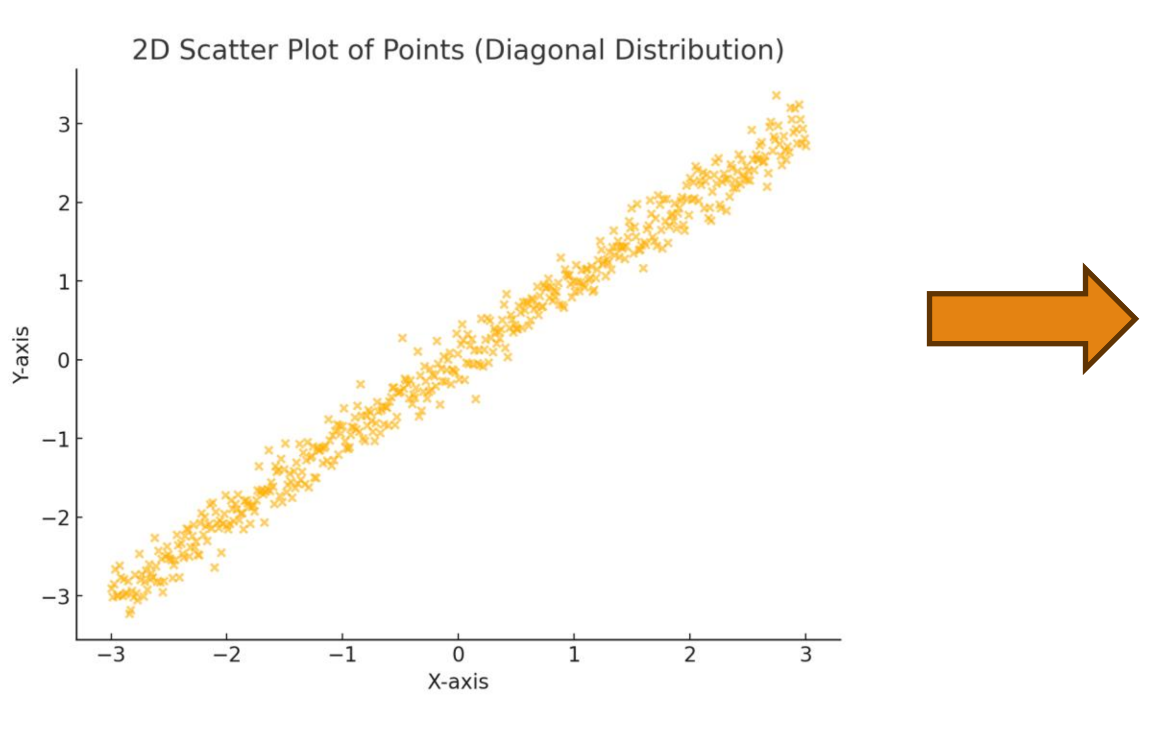
Generating synthetic samples



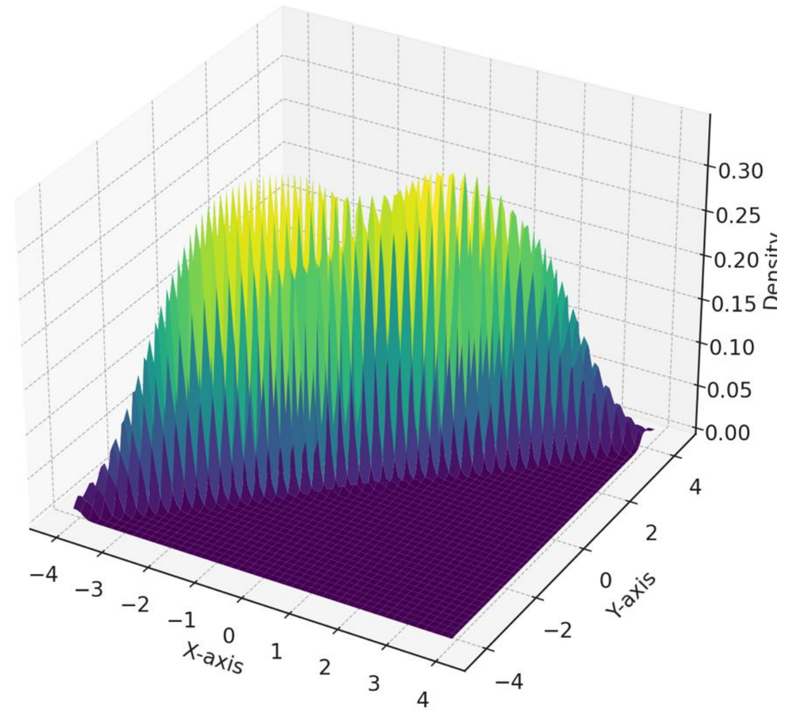
Training samples

Probability density estimation

Generating synthetic samples



Training samples



Probability density estimation

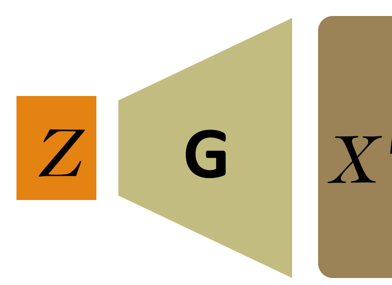
Generating synthetic samples

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

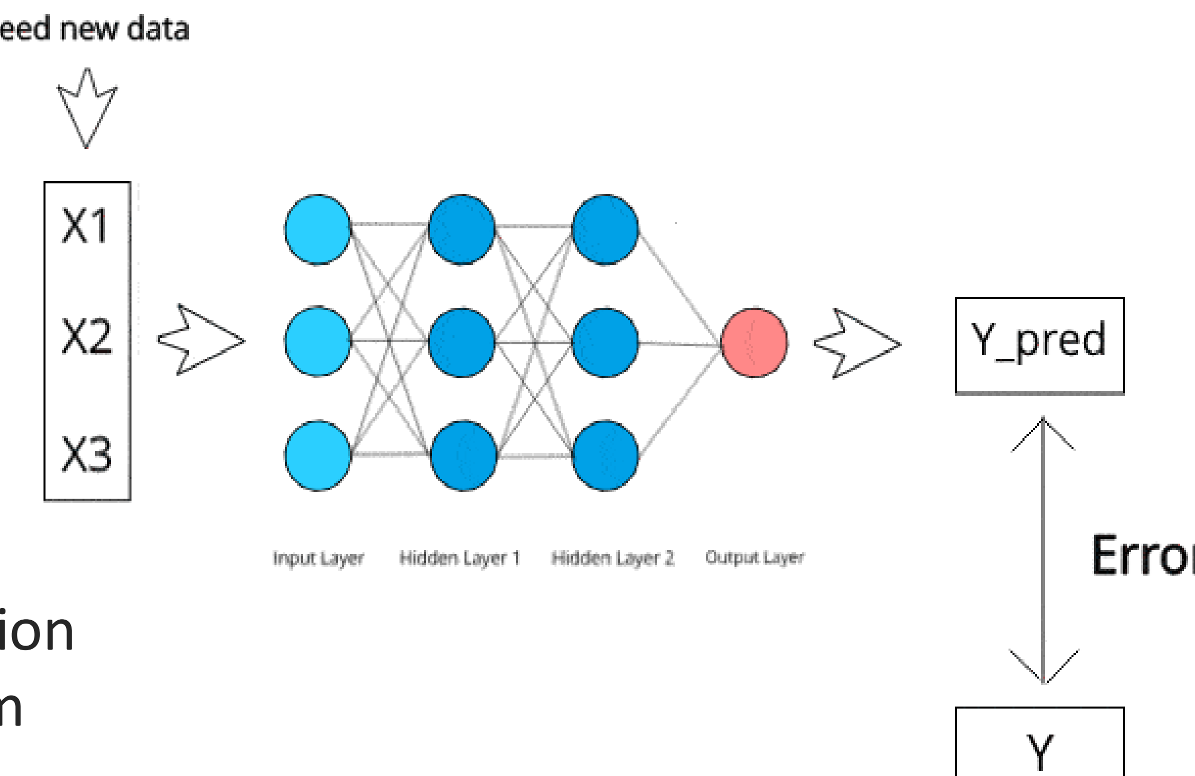
Solution: Sampling from something simple (noise) and learning a transformation to the real (training) distribution

Main components of the **Generative Model:**

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



How Generator Learn?



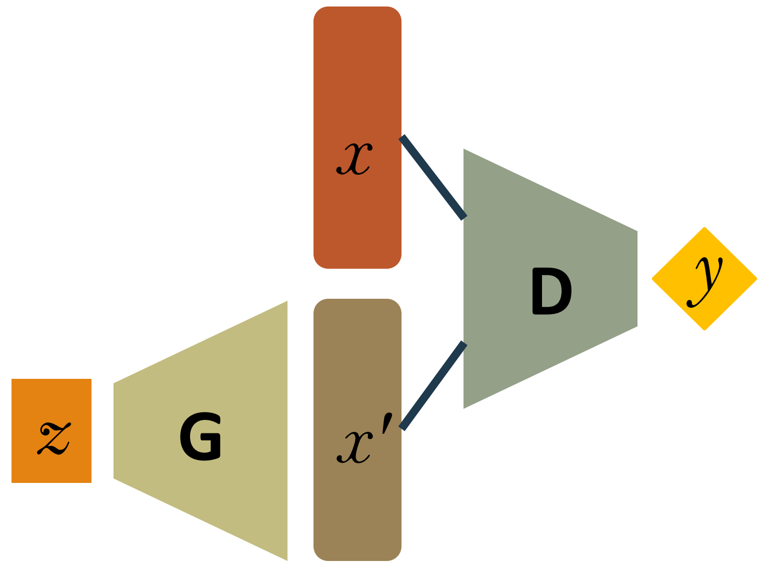
Using **another model** that gives information about how close/far are the samples from real data → **Discriminator**

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**)

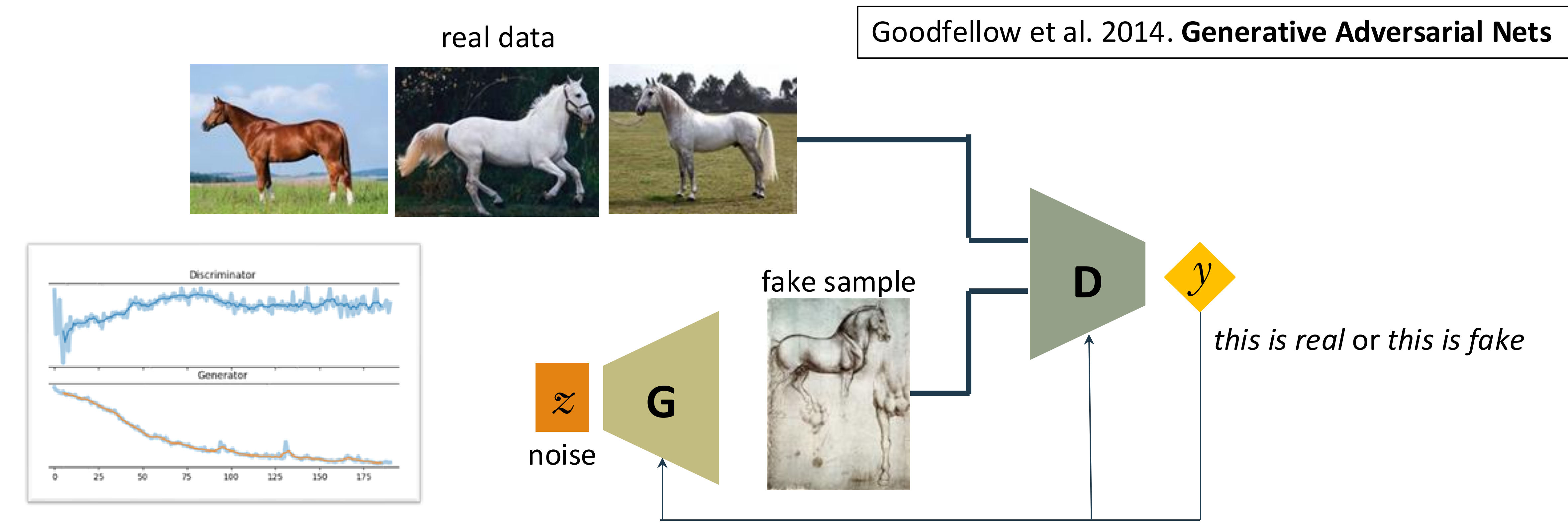
- Generator (**G**) learns how to create synthetic/fake data samples (X') by sampling random noise (Z) to fool the discriminator (**D**)



Generative Adversarial Networks

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

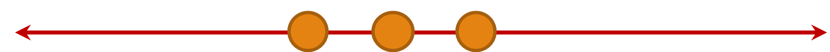
Goodfellow et al. 2014. **Generative Adversarial Nets**



GAN Training

GAN training intuition

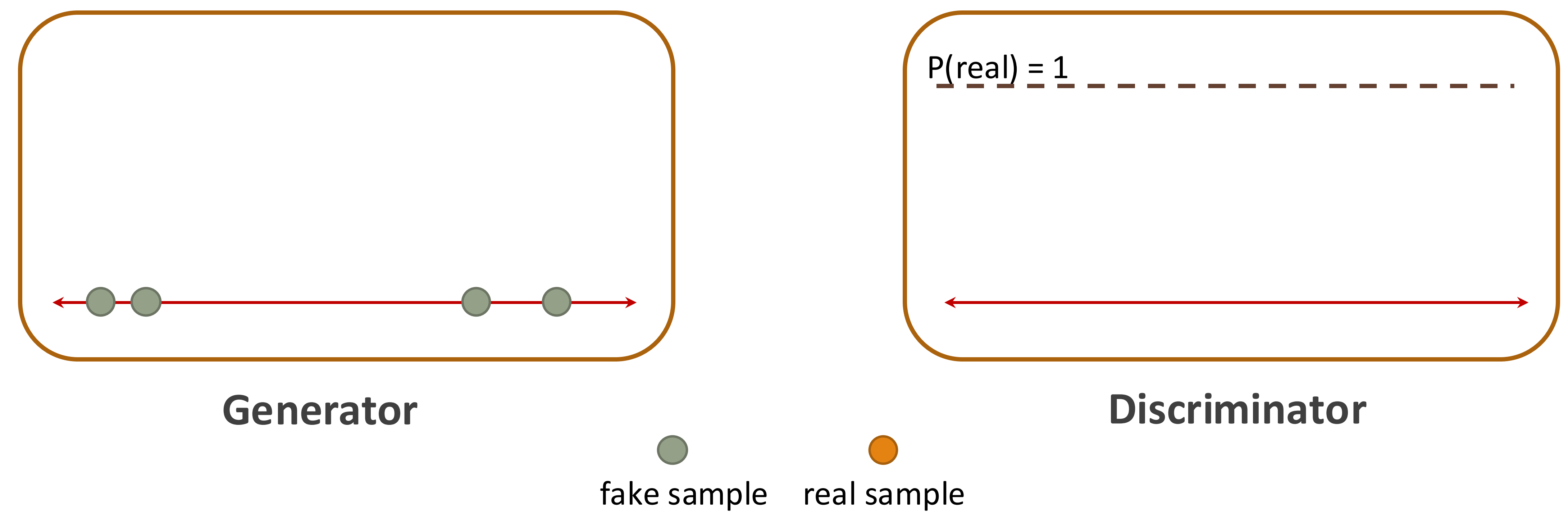
Real data distribution



GAN Training

Step 1:

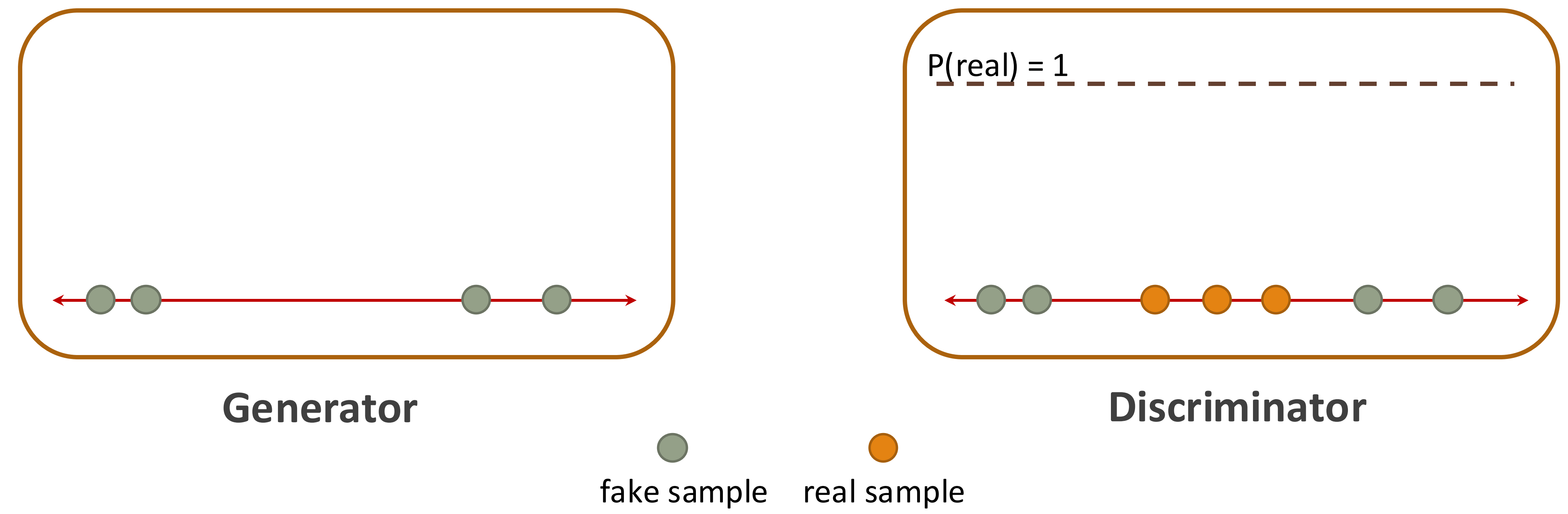
Generator samples from the noise to create data samples to imitate real data



GAN Training

Step 2:

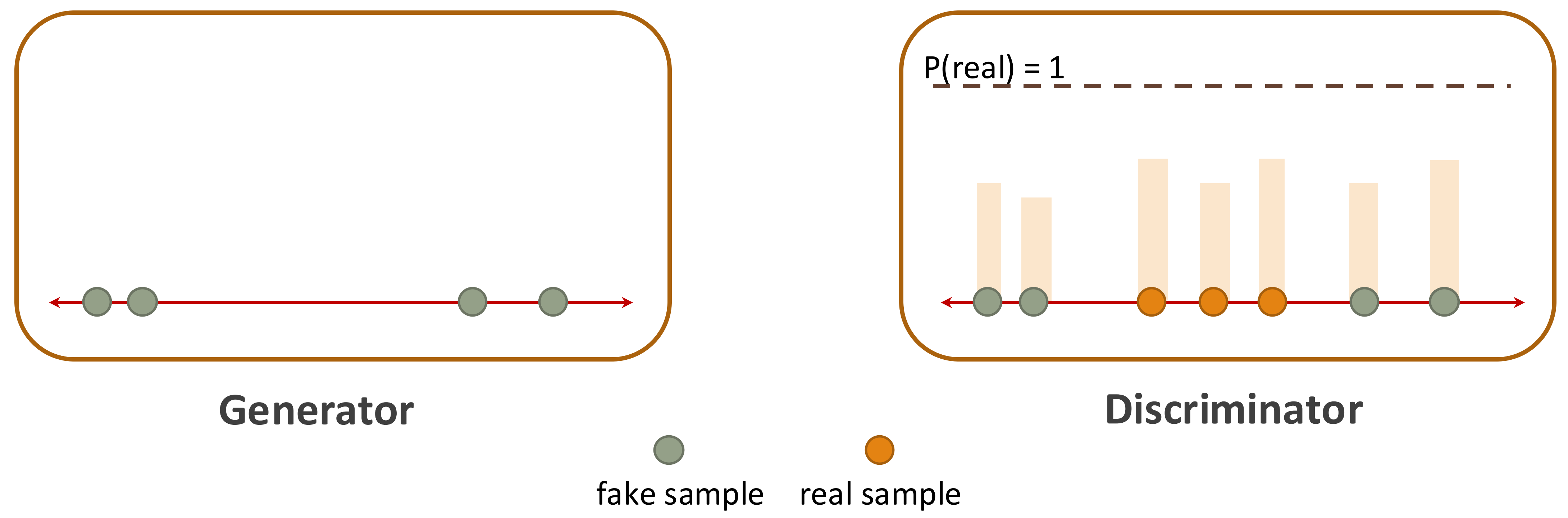
Discriminator gets fake samples from the generator and real samples from the real data distribution



GAN Training

Step 3:

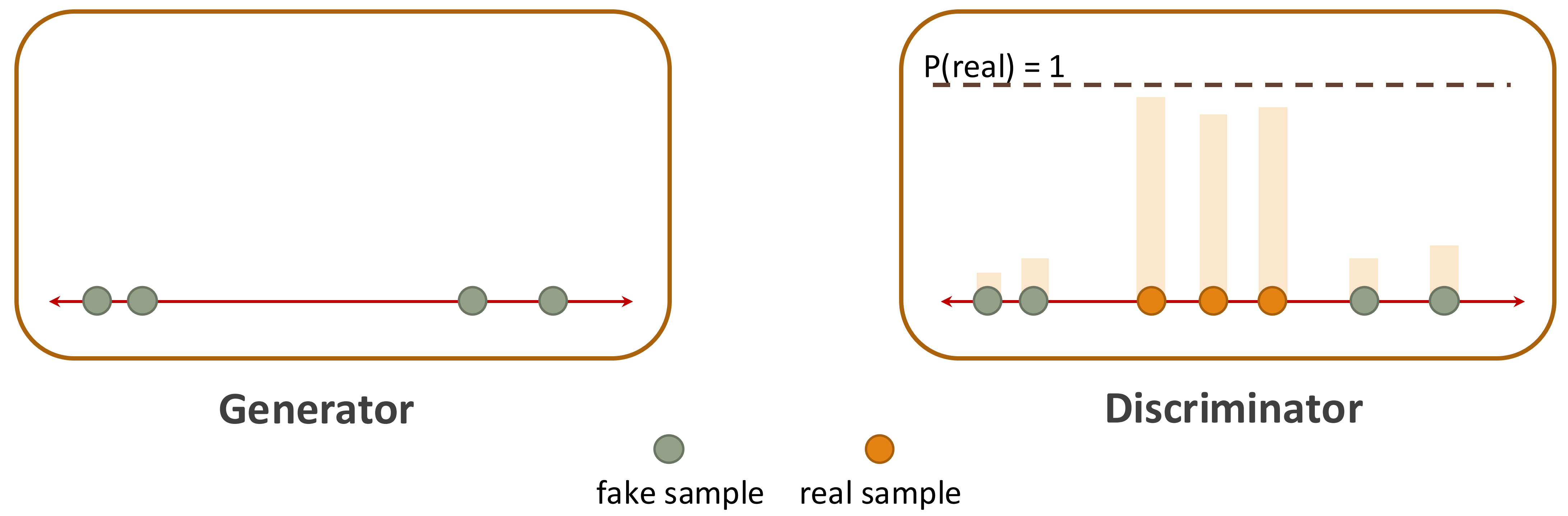
Discriminator learn how to distinguish between real and fake data (*supervised learning*)



GAN Training

Step 3:

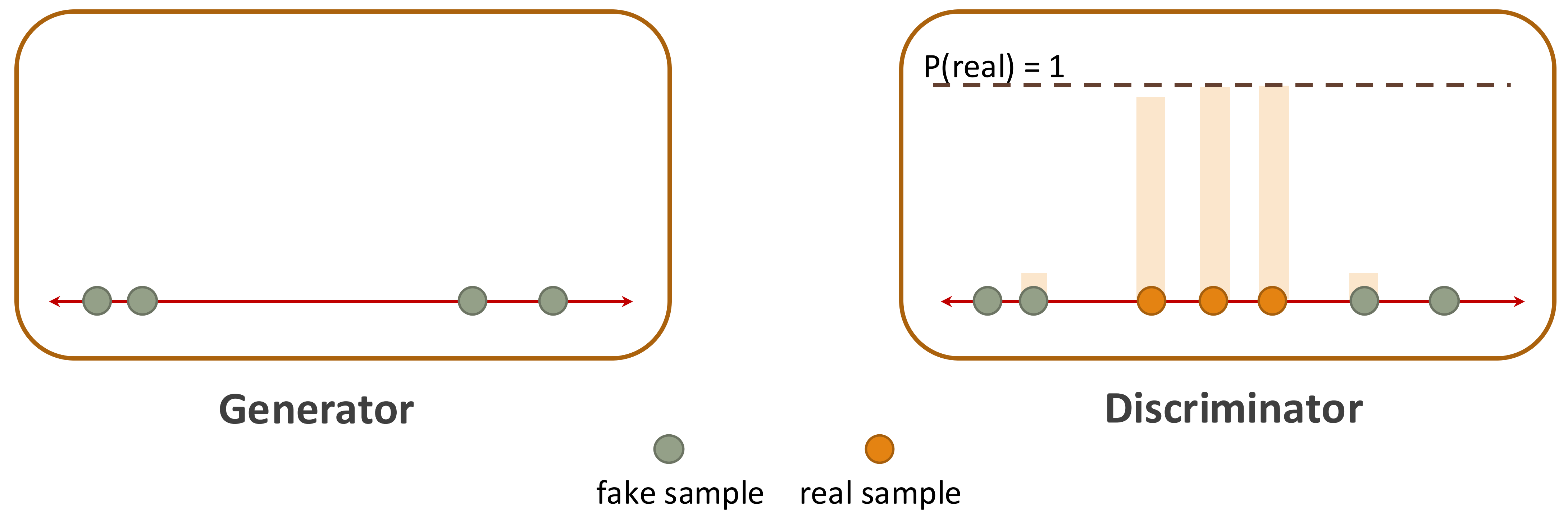
Discriminator learn how to distinguish between real and fake data (*supervised learning*)



GAN Training

Step 3:

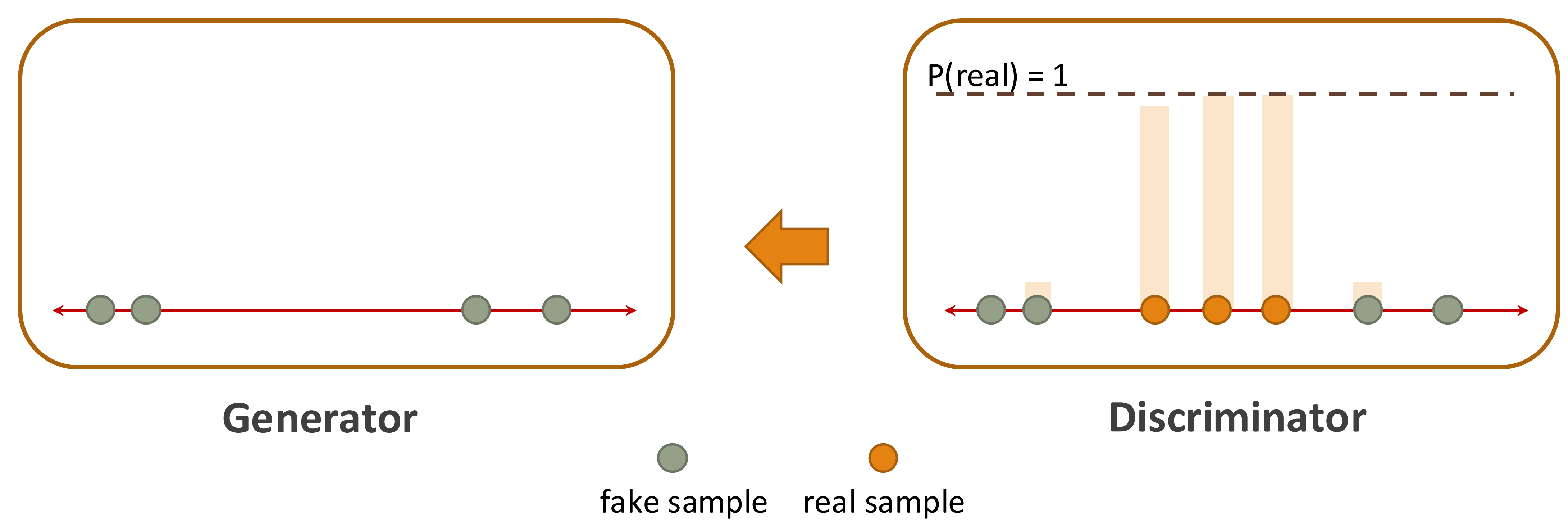
Discriminator learn how to distinguish between real and fake data (*supervised learning*)



GAN Training

Step 4:

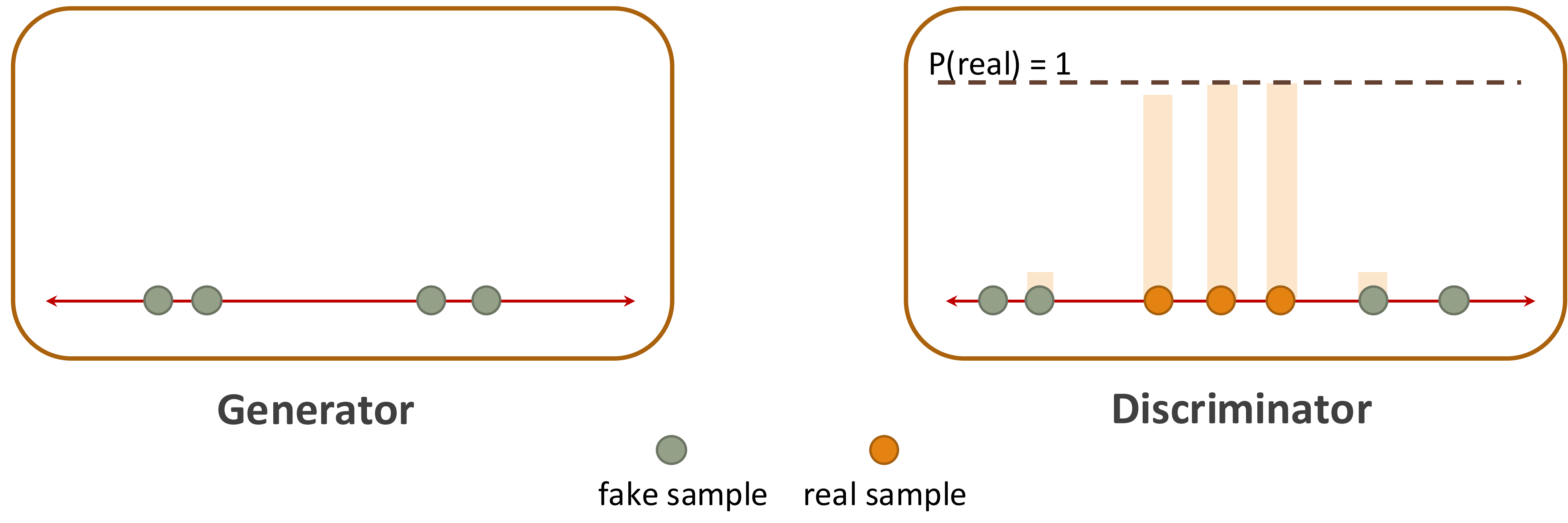
Generator get discriminator feedback and updates its parameters (weights) to improve the synthetic data



GAN Training

Step 4:

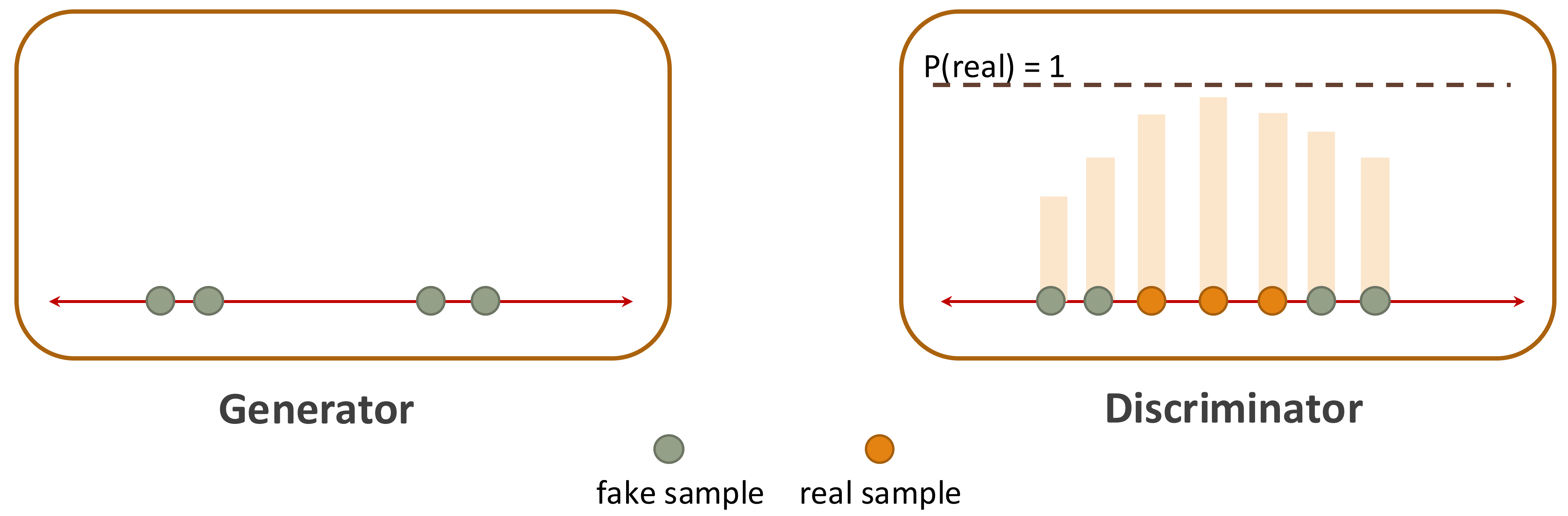
Generator get discriminator feedback and updates its parameters (weights) to improve the synthetic data



GAN Training

Step 5:

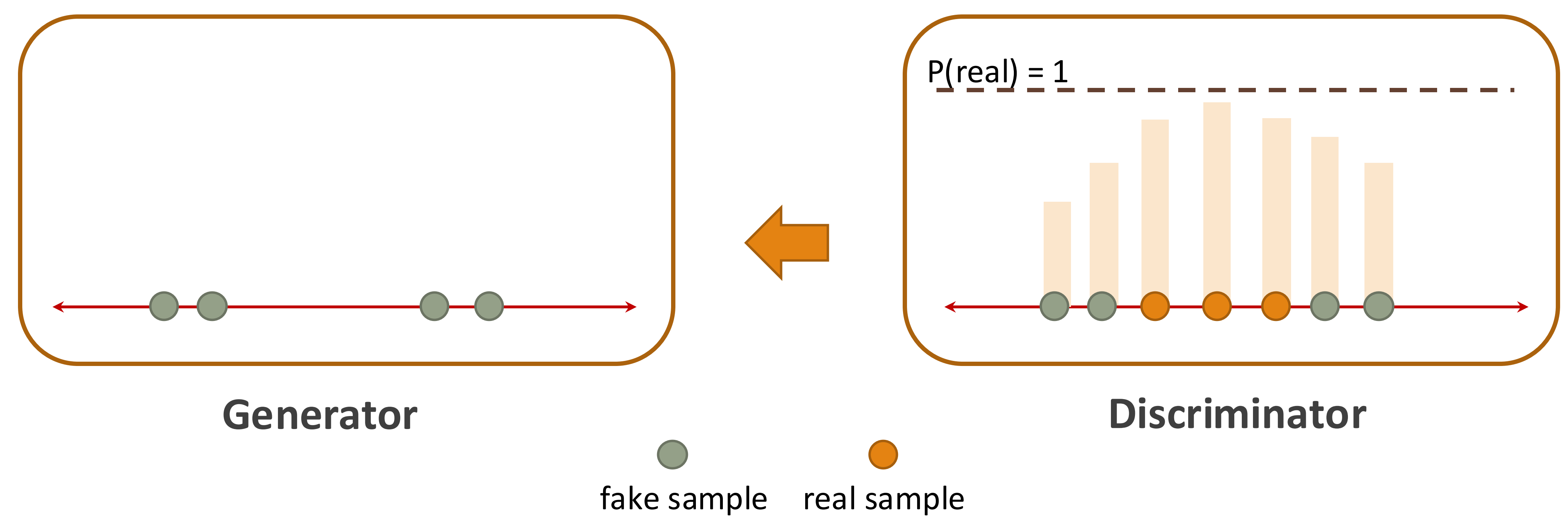
Discriminator gets samples (real and fake) and learns how to distinguish between real and fake data (*supervised learning*)



GAN Training

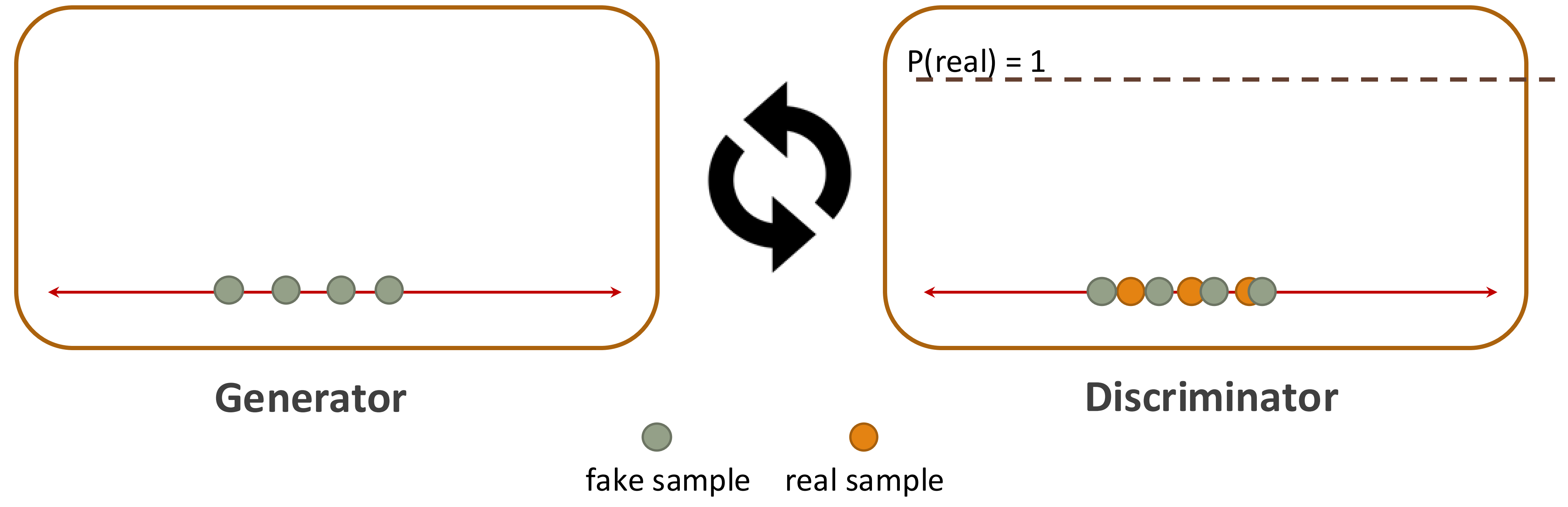
Step 5:

Generator get discriminator feedback and updates its parameters (weights) to improve the synthetic data



GAN Training

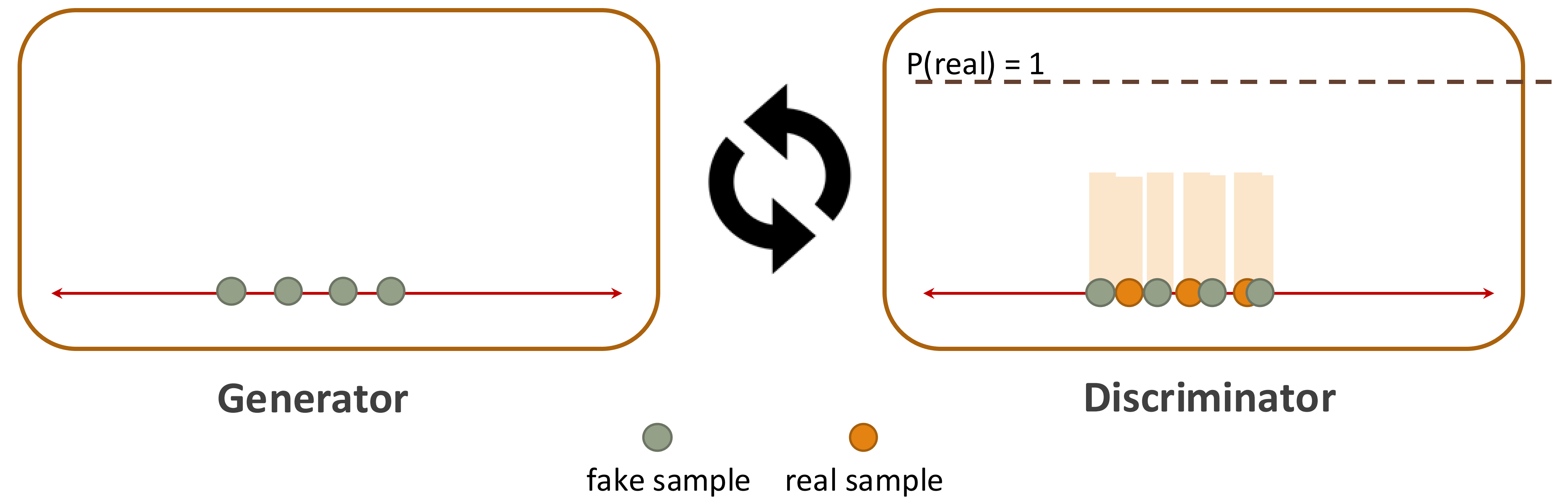
This steps are repeated until the generator is able to fool the discriminator by generating fake data samples that are indistinguishable from the real ones



GAN Training

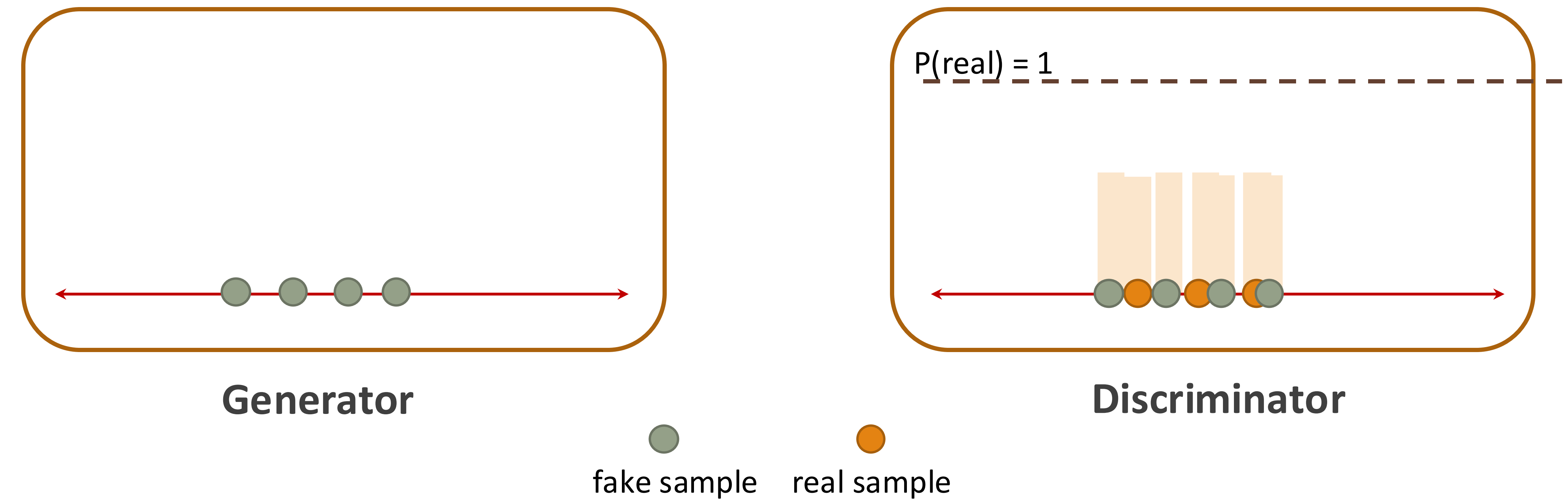
This steps are repeated until the generator is able to fool the discriminator by generating fake data samples that are indistinguishable from the real ones

- Discriminator is not able to distinguish between real and fake (random output)



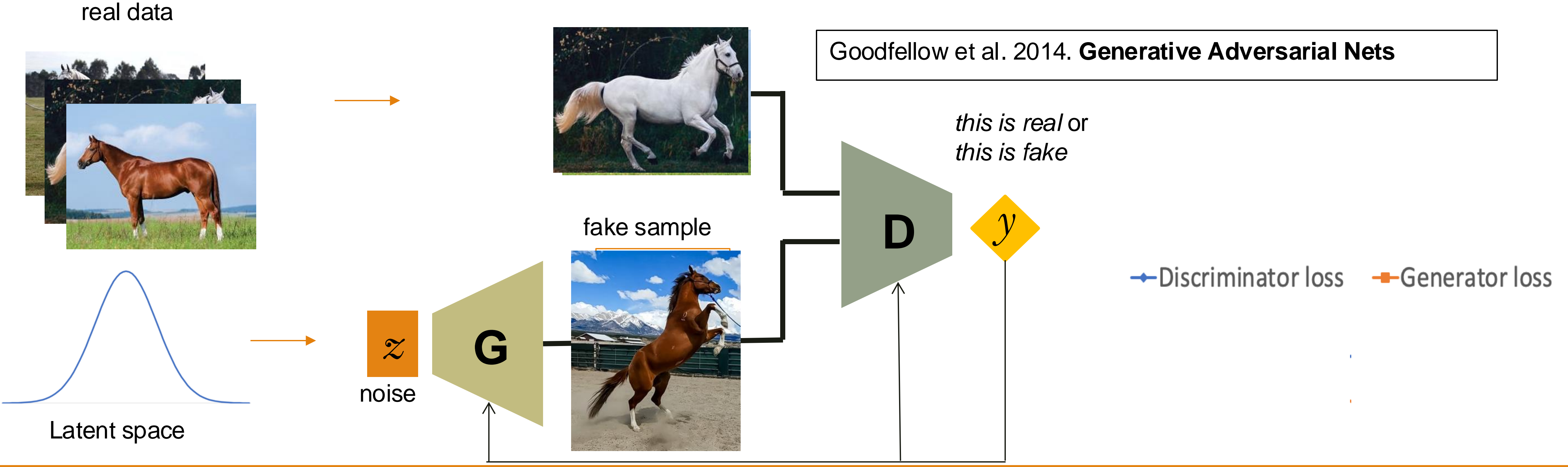
GAN Training

After finishing the training process, the generator network can be used to create samples



GAN Training

- Discriminator attributes a probability *p* of confidence of a sample being real
- Generator learns the real data distribution to generate fake samples
  - Generator slightly changes the generated data based on Discriminator’s feedback

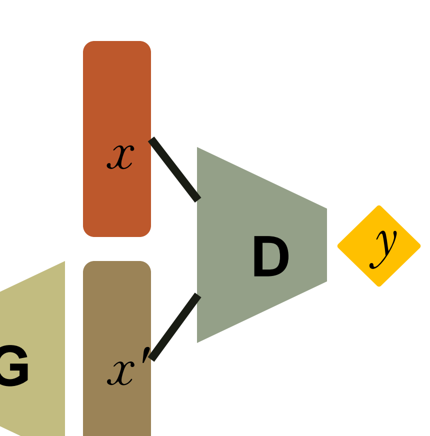


GAN Training

Generator and Discriminator are trained together (**minimax game**)

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

- **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))

- 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

GAN Training. General Algorithm

Steps of the main training loop for a classifier training:

1. Train the model

1.1 Sample a batch of labeled data

1.1.1 Extract a batch of input samples (xxx) and corresponding labels (yyy) from the dataset.

1.2 Forward pass

1.2.1 Pass the input samples (xxx) through the model to get predictions (y^\hat{y}y^).

1.3 Compute loss

1.3.1 Compare the model’s predictions (y) with the true labels (yyy) using a loss function

1.4 Backward pass

1.4.1 Compute gradients of the loss with respect to the model’s parameters using backpropagation.

1.5 Update weights

1.5.1 Adjust the model’s parameters using an optimizer to minimize the loss.

GAN Training. General Algorithm

Steps of the main training loop for GAN Training:

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

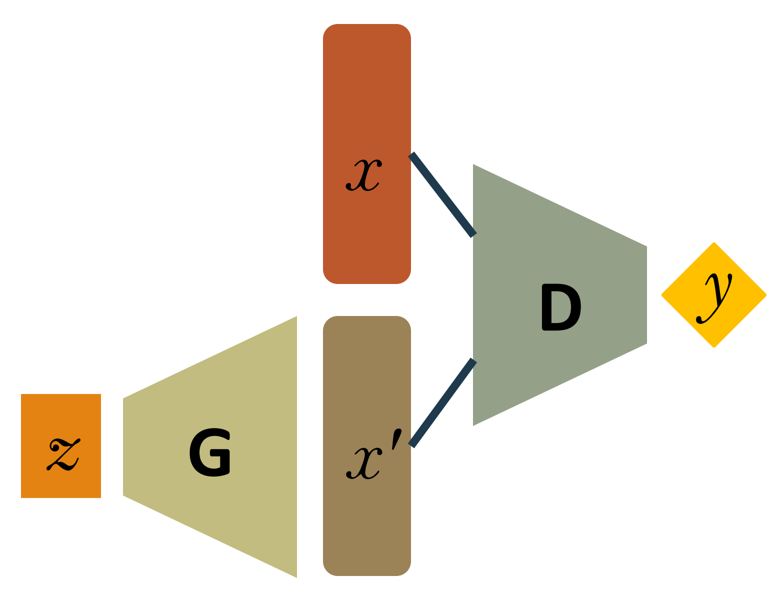
2. Train the generator

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



GAN Training. General Algorithm

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 $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ 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 $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ 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 Training. General Code

0. Create ANNs

```
class Generator(nn.Module):
    """
    Class that defines the the Generator Neural Network
    """
    def __init__(self, input_size, hidden_size, output_size):
        super(Generator, self).__init__()

        self.net = nn.Sequential(
            nn.Linear(input_size, hidden_size),
            nn.SELU(),
            nn.Linear(hidden_size, hidden_size),
            nn.SELU(),
            nn.Linear(hidden_size, output_size),
            nn.SELU(),
        )

    def forward(self, x):
        x = self.net(x)
        return x
```

```
class Discriminator(nn.Module):
    """
    Class that defines the the Discriminator Neural Network
    """
    def __init__(self, input_size, hidden_size, output_size):
        super(Discriminator, self).__init__()

        self.net = nn.Sequential(
            nn.Linear(input_size, hidden_size),
            nn.ELU(),
            nn.Linear(hidden_size, hidden_size),
            nn.ELU(),
            nn.Linear(hidden_size, output_size),
            nn.Sigmoid()
        )

    def forward(self, x):
        x = self.net(x)
        return x
```

GAN Training. General Code

1. Train discriminator

```
# 1. Train the discriminator
discriminator.zero_grad()
# 1.1 Train discriminator on real data
input_real = get_data_samples(batch_size)
discriminator_real_out = discriminator(input_real.reshape(batch_size, 2))
discriminator_real_loss = discriminator_loss(discriminator_real_out, real_data_target(batch_size))
discriminator_real_loss.backward()
# 1.2 Train the discriminator on data produced by the generator
input_fake = read_latent_space(batch_size)
generator_fake_out = generator(input_fake).detach()
discriminator_fake_out = discriminator(generator_fake_out)
discriminator_fake_loss = discriminator_loss(discriminator_fake_out, fake_data_target(batch_size))
discriminator_fake_loss.backward()
# 1.3 Optimizing the discriminator weights
discriminator_optimizer.step()
```

GAN Training. General Code

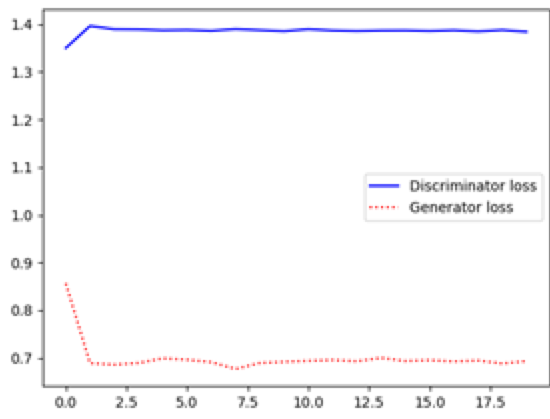
2. Train discriminator

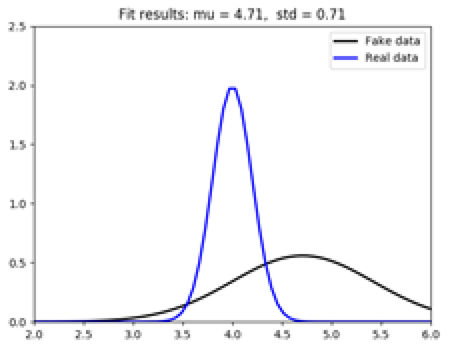
```
# 2. Train the generator
if batch_number % freeze_generator_steps == 0:
  generator.zero_grad()
  # 2.1 Create fake data
  input_fake = read_latent_space(batch_size)
  generator_fake_out = generator(input_fake)
  # 2.2 Try to fool the discriminator with fake data
  discriminator_out_to_train_generator = discriminator(generator_fake_out)
  discriminator_loss_to_train_generator = generator_loss(discriminator_out_to_train_generator,
                                                         real_data_target(batch_size))
  discriminator_loss_to_train_generator.backward()
  # 2.3 Optimizing the generator weights
  generator_optimizer.step()
```

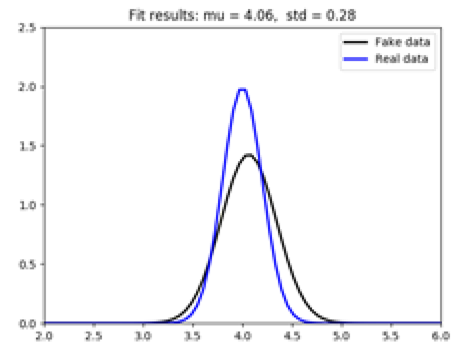
GAN Training. Source Code Example 1

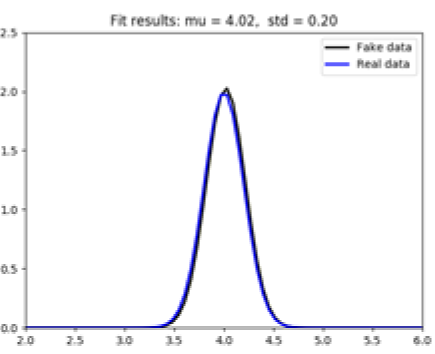
Example: Train a generator to create vectors of a given size that contains float numbers that follow a normal distribution given the mean and the standard deviation

- Real dataset samples: Vectors of real numbers that follow a normal distribution
- Source code: https://drive.google.com/file/d/1BzRQlhVCO7TsIPNVzqW-jU9YOKqDFyzg/view?usp=sharing





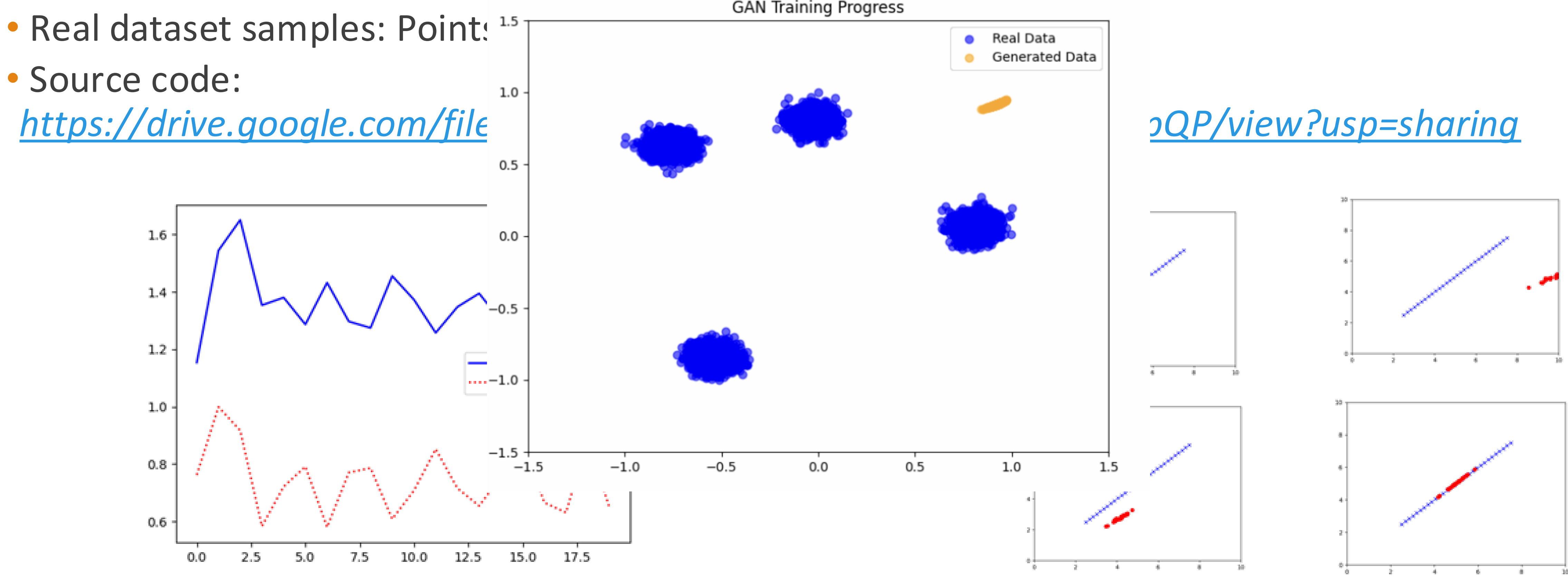




GAN Training. Source Code Example 2

Example: Train a generator to create 2D points (x, y) that belong to a line in the 2D space

- Real dataset samples: Point
- Source code:
  https://drive.google.com/file ... QP/view?usp=sharing

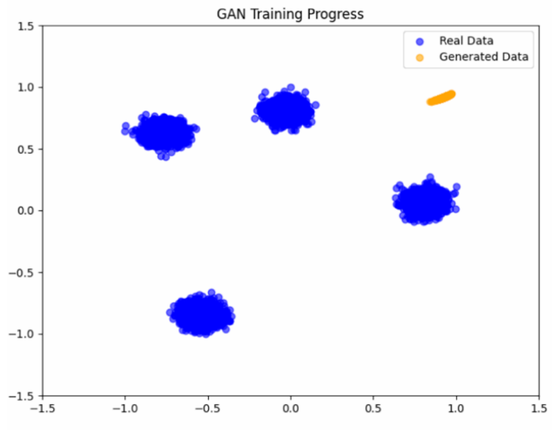


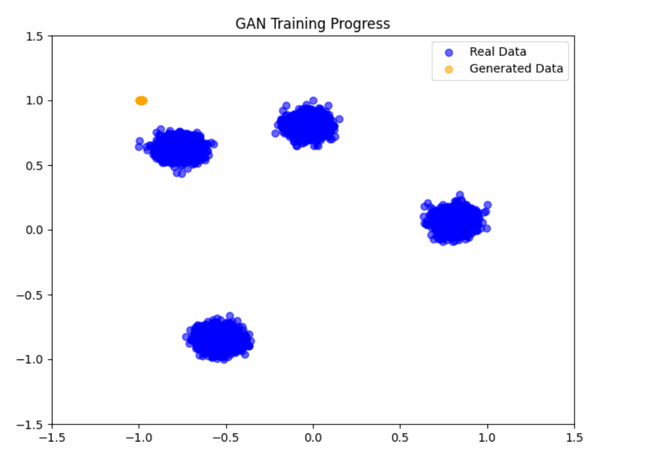
- Test *freezing* the generator and increasing the number of epochs

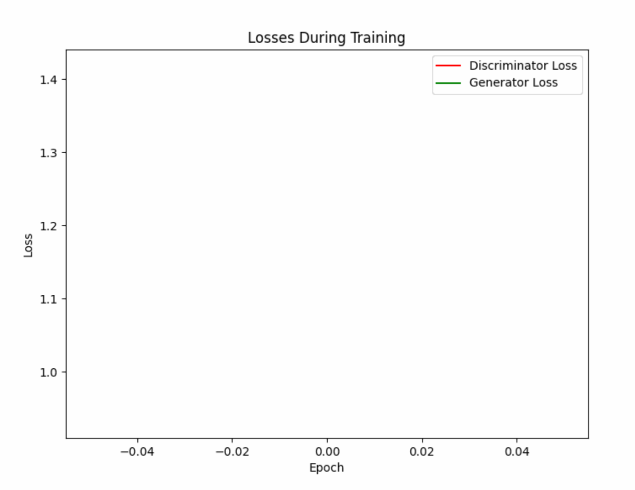
GAN Training. Source Code Example 3

Example: Train a generator to create 2D points organized in blobs

- Real dataset samples: Points (x, y) organized in blobs
- Source code: https://colab.research.google.com/drive/1ZMH2cCRA9j2MqJU_v2qfHsAhHaeZ5mLo?usp=sharing







Can you improve the results?

GAN Training. Source Code Example 4

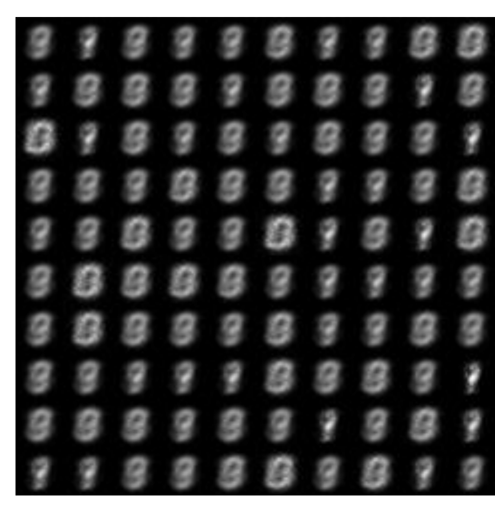
Example: Train a generator to create samples of handwritten digits of MNIST dataset.

The MNIST dataset is one of the most common datasets used for image classification and generation. It contains 60,000 training images and 10,000 testing images of handwritten digits (from 0 to 9)

- Real dataset samples: Digits from MNIST dataset
- Source code:
  https://drive.google.com/file/d/1yhZ1yubqfPAaxJqHdF3D7jTI5hLBooBK/view?usp=sharing

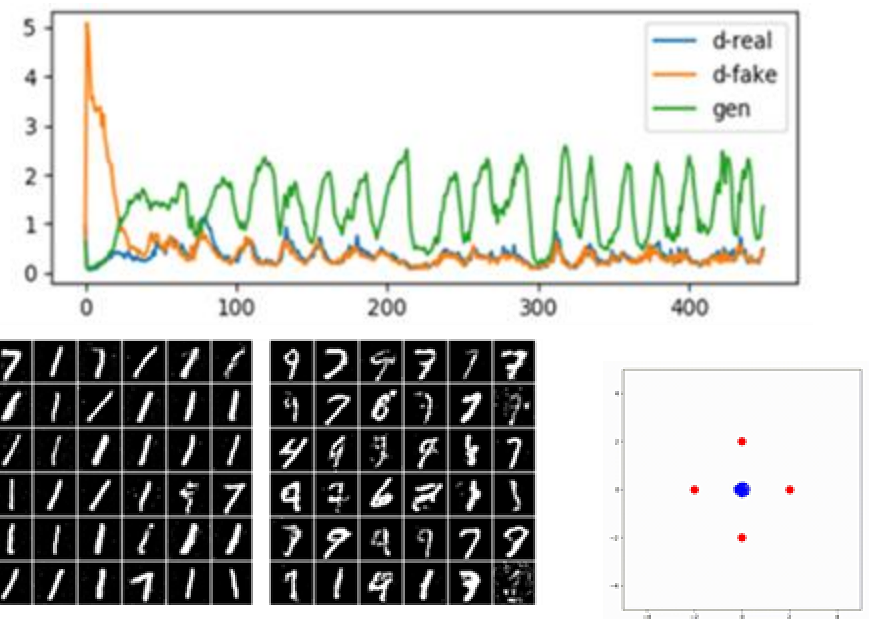
Real data

Generated data during training



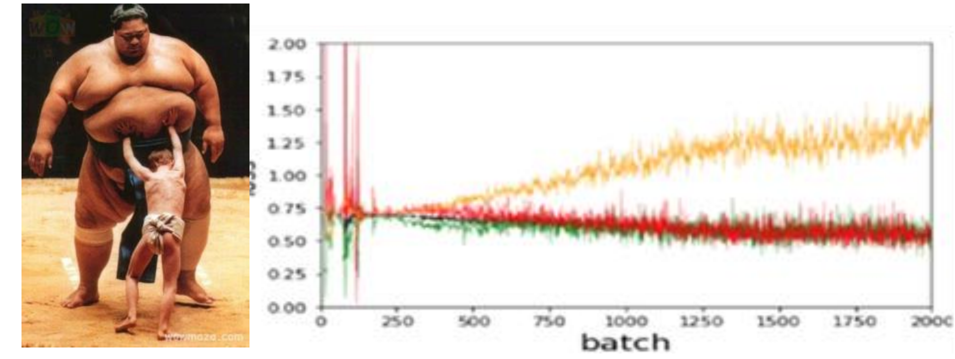
Not all is good news

- **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



Some GAN Applications

Generate New Samples of Image Datasets

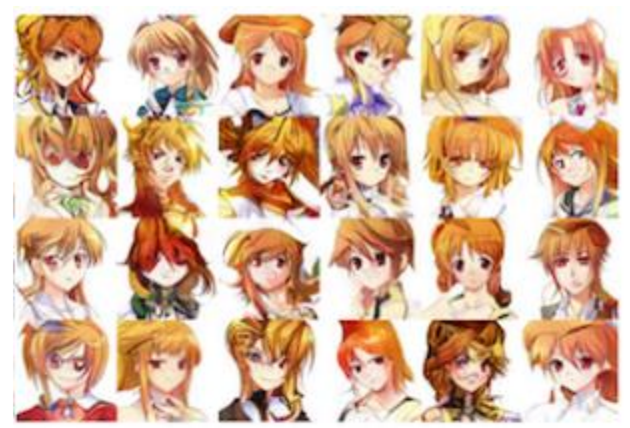
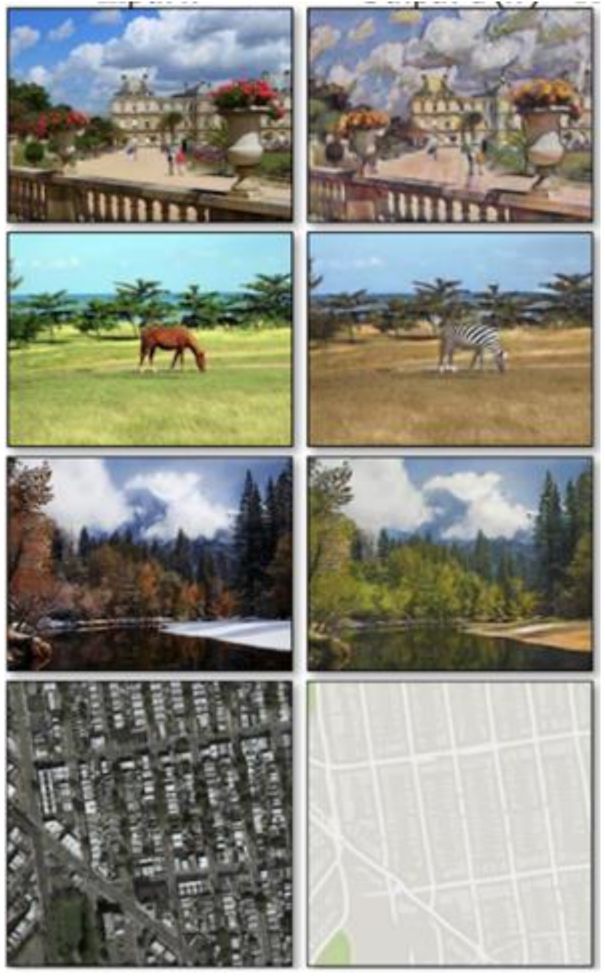


Image-to-Image Translation



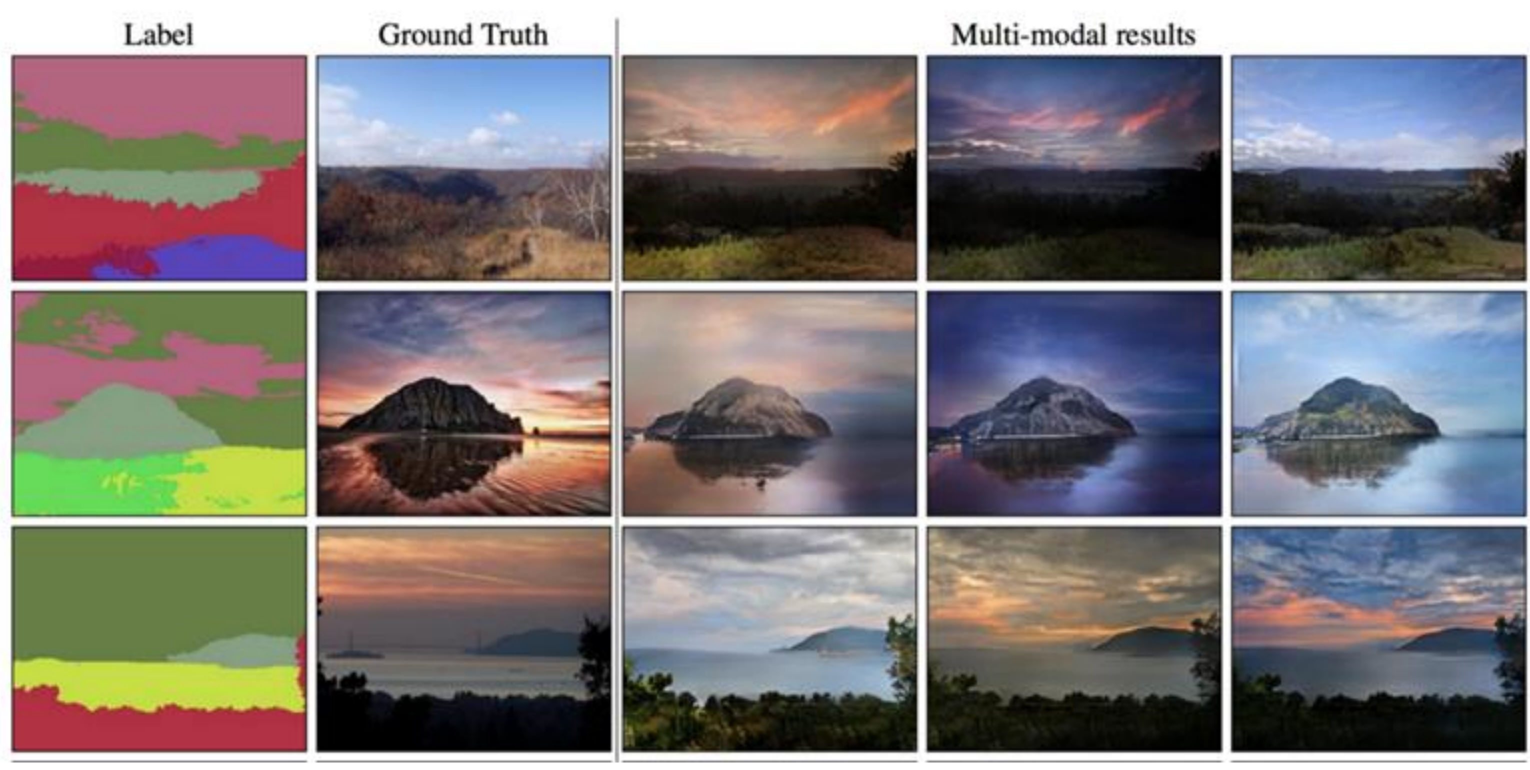
Text-to-Image Translation

Text description

Stage-I images

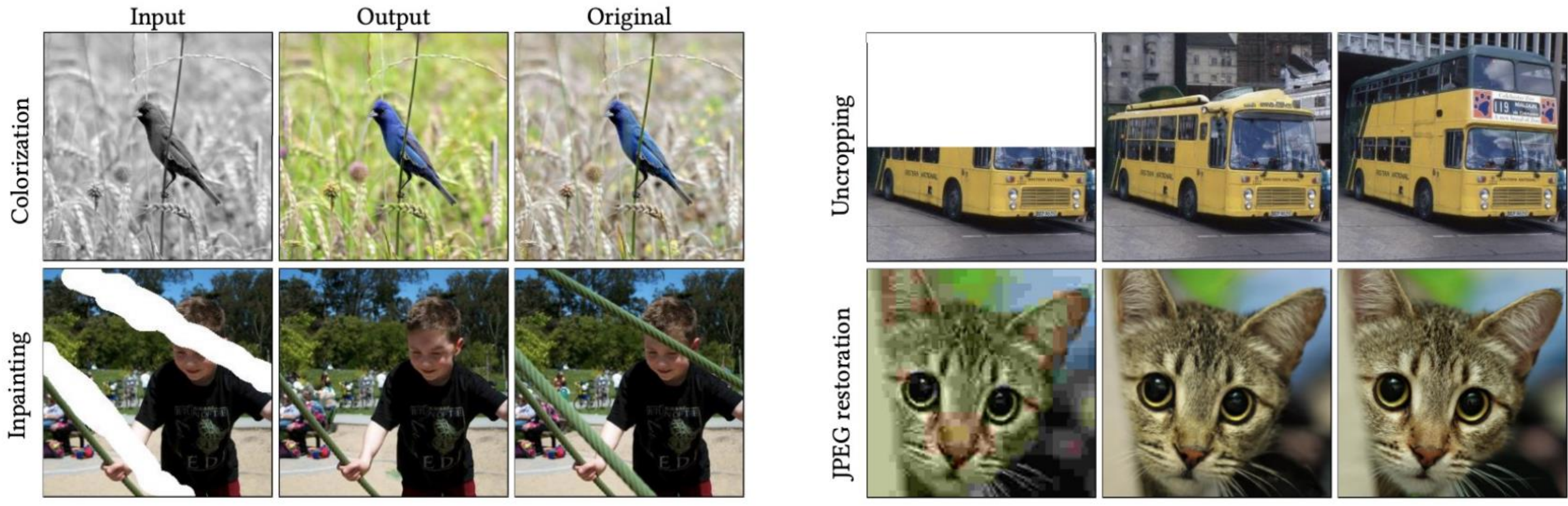
Representation Learning

Semantic-Image-to-Photo Translation

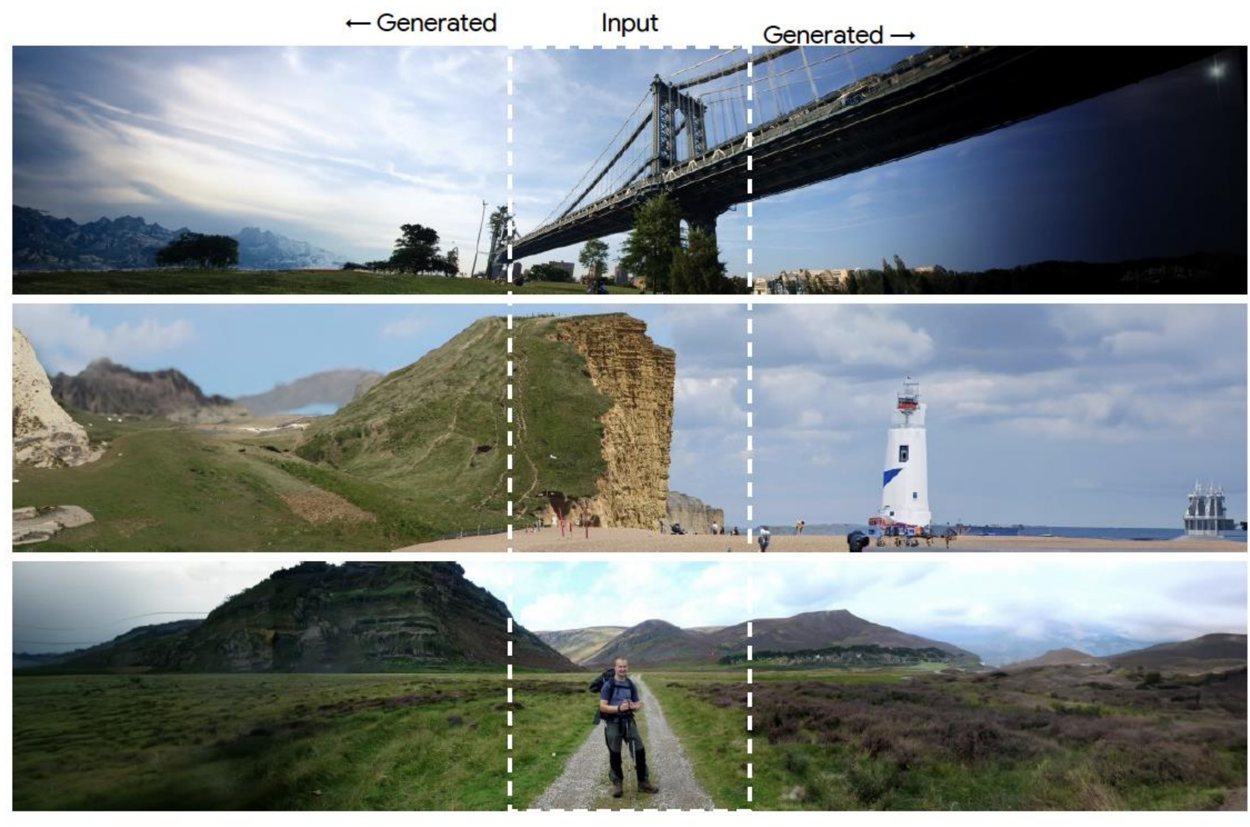


http://nvidia-research-mingyuliu.com/gaugan

Colorization, Inpainting, Restoration



Outfitting



Now!!

Imagen Video

Stable Diffusion

Much more …

- **Data augmentation**

- Train better classifiers through **semi-supervised learning**

ANNs Development

ANNs en PyTorch

- Material elaborado en ppt

ANNs en Tensorflow

- Material elaborado en ppt

PyTorch vs Tensorflow

- Material elaborado en ppt

Thanks!

Comments?

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

Sergio Nesmachnow
sergion@fing.edu.uy