Deep generative neural networks Fundamentals & problem solving

Class 2

JAMAL TOUTOUH

jamal@uma.es

jamal.es

@jamtou

Artificial Neural Networks

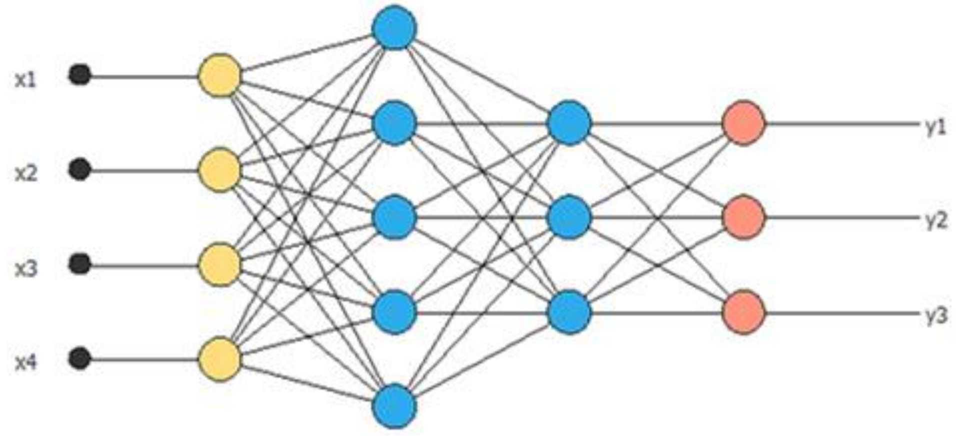
A neural network is a bunch of neurons connected together.

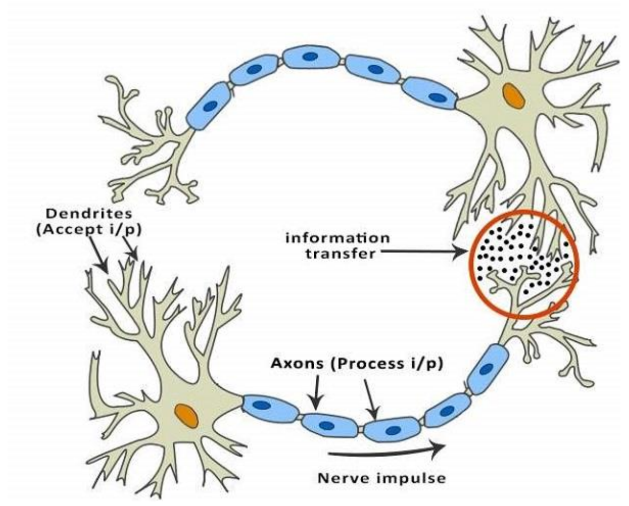
Neural networks are typically **organized in layers**.

Layers are made up of a number of **interconnected neurons.**

Inputs are presented to the network via the **input layer**, which communicates to **one or more hidden layers** through **weighted connections**.

The hidden layers then link to an **output layer**.

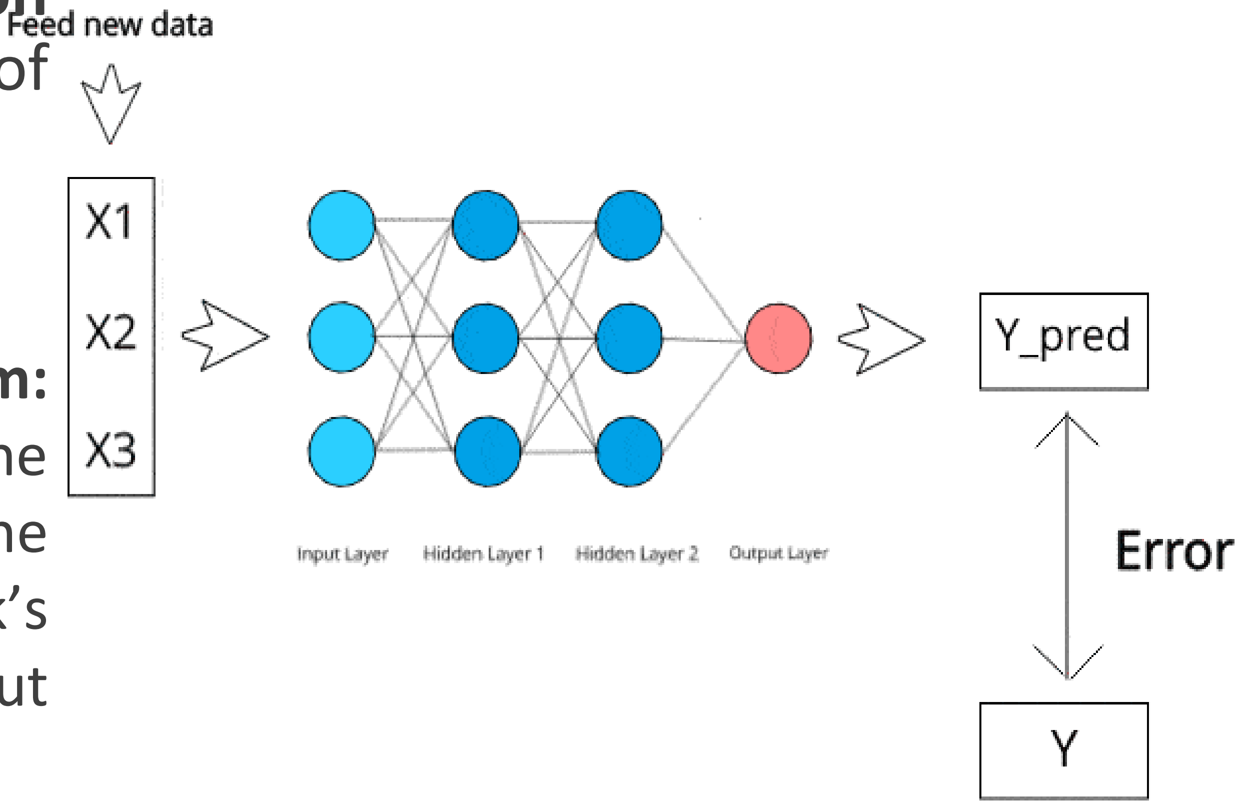




How do ANNs Learn?

ANNs learn by solving the **optimization problem** of reducing the error in terms of ***error***, ***loss*** or ***cost function***.

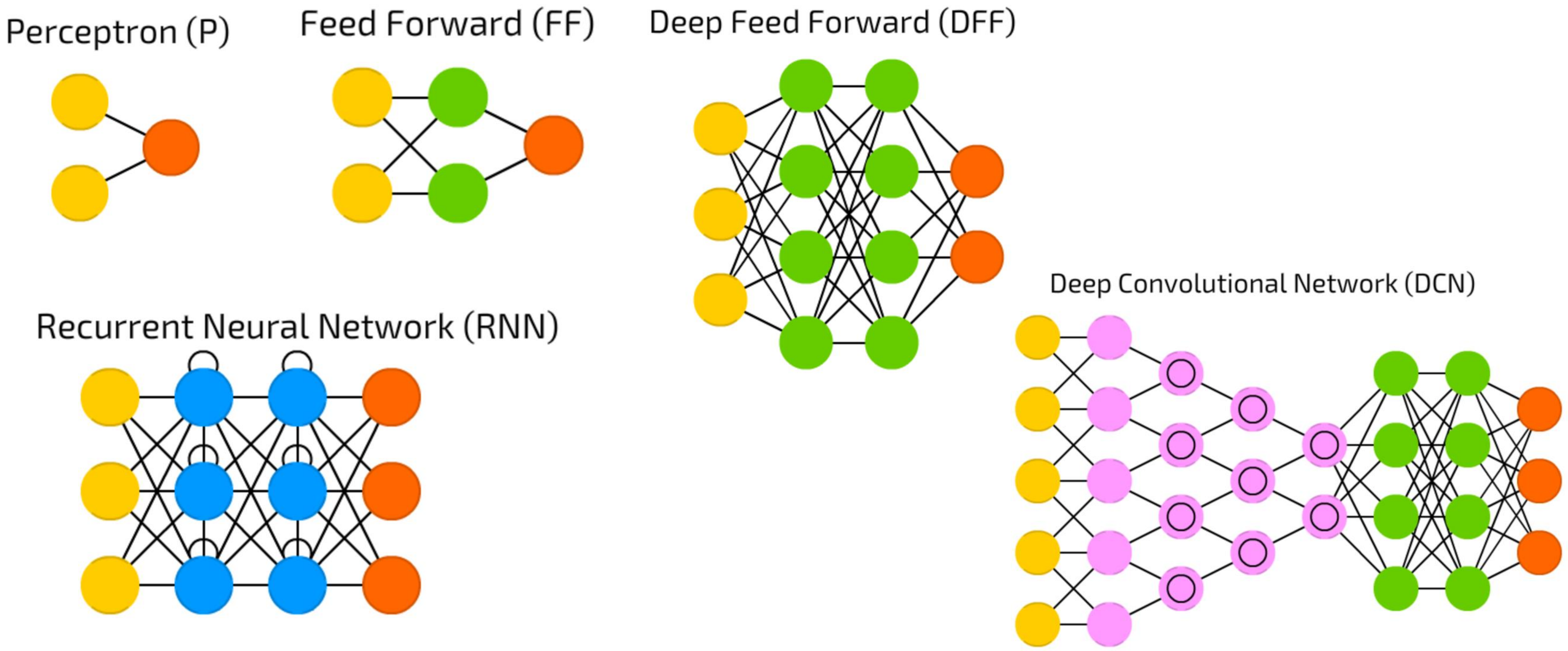
Back Propagation Algorithm:
It learns by example. If you submit to the algorithm the example of what you want the network to do, it changes the network's weights so that it can produce desired output for a particular input on finishing the training.



Deep Learning

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

Artificial Neural Networks. Types


Perceptron (P)
Feed Forward (FF)
Deep Feed Forward (DFF)
Recurrent Neural Network (RNN)
Deep Convolutional Network (DCN)

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

Implementation. Google colab examples

- **Generic use of Colab**: image handling and processing using OpenCV
  https://drive.google.com/file/d/1doMYqK1EyVU3YvtVPAPvZX5I8pYpoBQt/view?usp=sharing

- **ANN examples in Colab**
  1. Neuron with different activation functions
  https://drive.google.com/file/d/1RsznOUh0s2fZM_pHiQ751u9nO4zc3_7_/view?usp=sharing
  2. Two layers ANN
  https://drive.google.com/file/d/15zBRGg_qSS7SJyrUOLLS6vWt696kmmvP/view?usp=sharing

Implementation. Google colab examples

- **ANN examples in Colab**

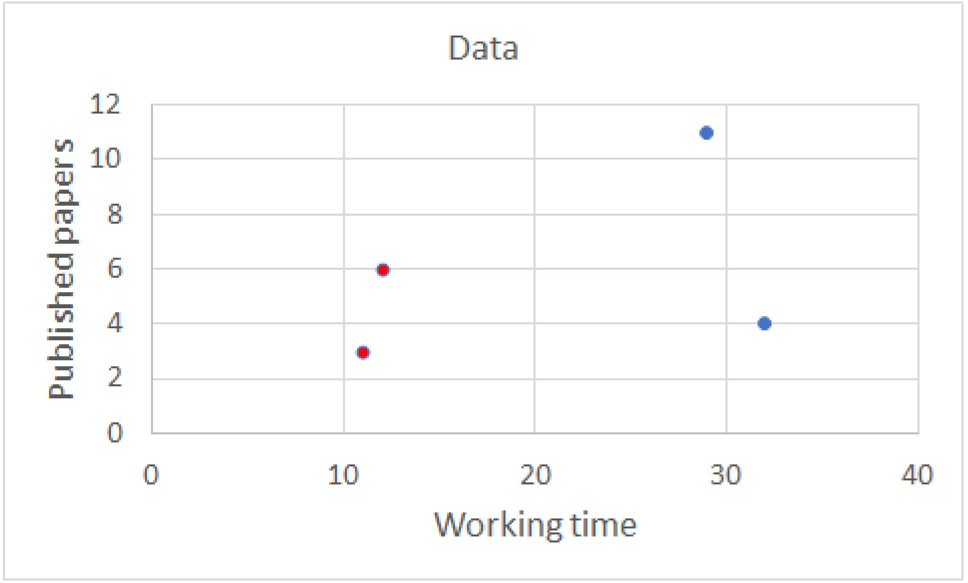
  4. Training two layers ANN

  https://drive.google.com/file/d/1FsCoTHksE6cJ-unrwgFGuZh6O2FP_X0d/view?usp=sharing

  5. Training two layers ANN (using PyTorch library)

  https://drive.google.com/file/d/1P3rQxcfZr2C7PBETMEmvOgVQD8WUbVQM/view?usp=sharing

	Working time	Published papers	Position
Michael	12	6	Researcher
Shash	32	4	Data Scientist
Aruna	11	3	Researcher
Lisa	29	11	Data Scientist
Jamal	14	3	?
Mina	31	0	?



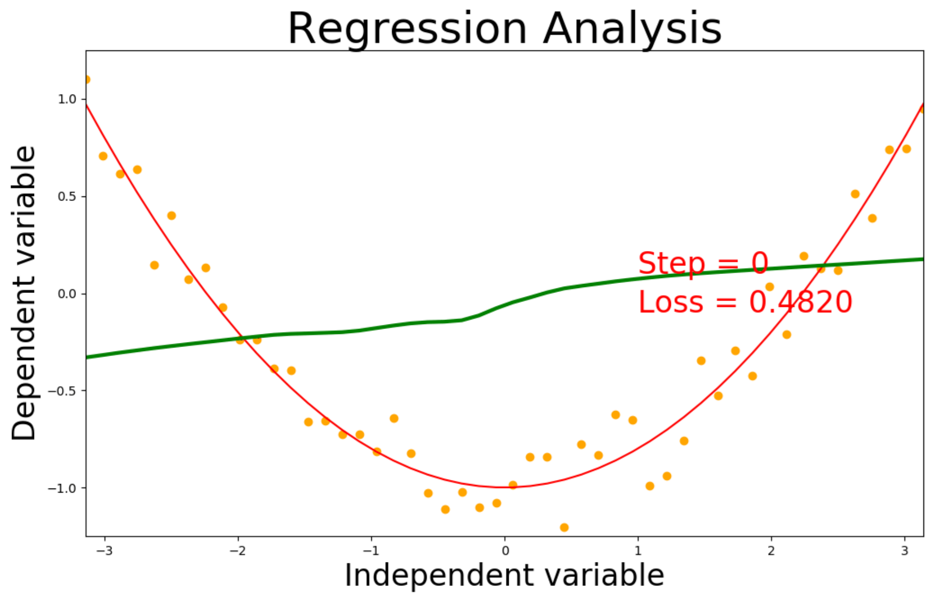
Deep Learning

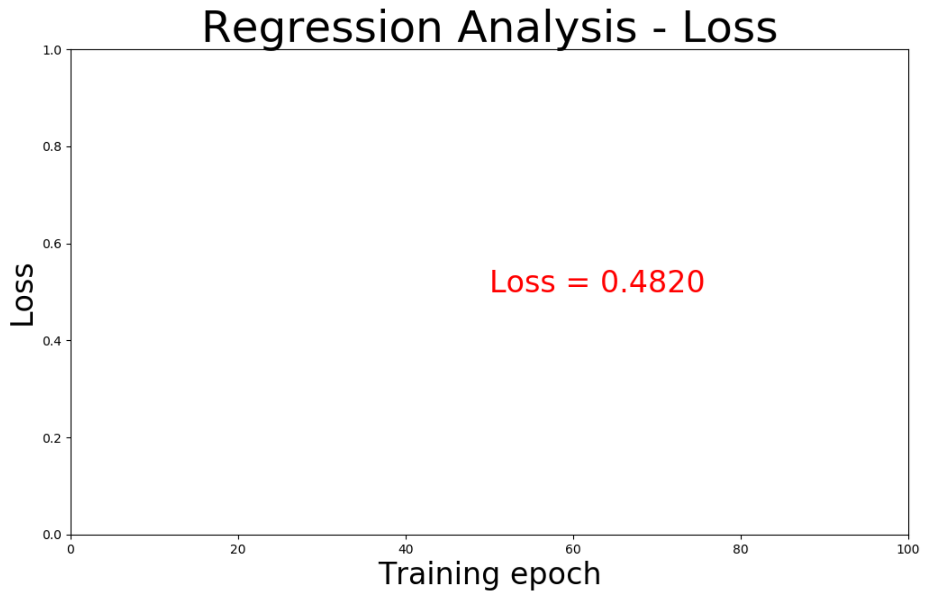
- ANN results get better with
  - more/better **data**
  - bigger **models**
  - more **computation**

Example:

Deep Learning for regression

https://drive.google.com/file/d/115dbtD8Zxs2I98t0wSzvauEkie4IPI5P/view?usp=sharing





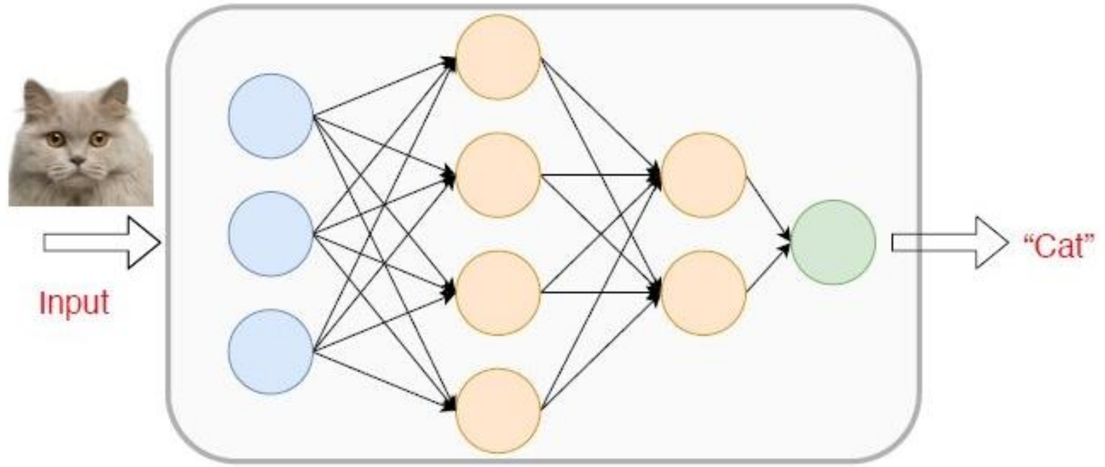
Artificial Neural Networks. DNN

- 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)**

Artificial Neural Networks. MLPs

- They are comprised of one or more layers of neurons. Data is fed to the **input layer**, there may be one or more **hidden layers** providing levels of abstraction, and final predictions are made on the **output layer**, also called the visible layer.



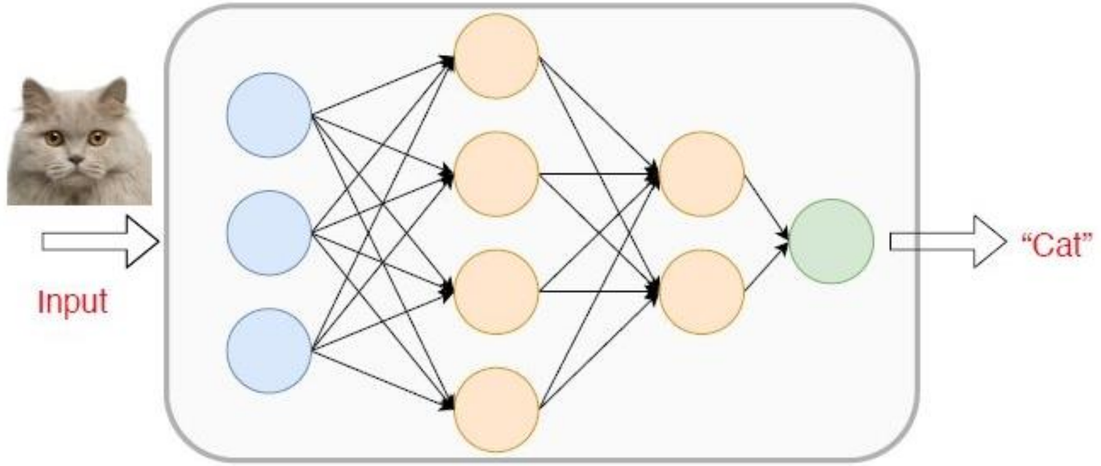
- Each new layer is a set of nonlinear functions of a weighted sum of all outputs (fully connected) from the prior one.

Artificial Neural Networks. MLPs

- They are very flexible and can be used to learn a mapping from inputs to outputs.
  - This flexibility allows them to be applied to many different types of data.

- MLP are suitable for:
  - Tabular datasets
  - Classification prediction problems
  - Regression prediction problems

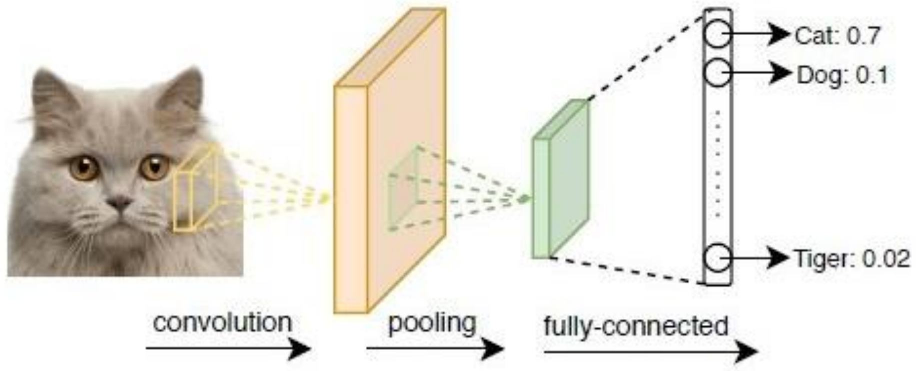
- They can be used aslo for:
  - Image data
  - Text Data
  - Time series data
  - Other types of data



Artificial Neural Networks. CNNs

- Inspired by complex/simple cells, Fukushima proposed Neocognitron (1980)
  - Hierarchical neural network used for handwritten Japanese character recognition
  - First CNN, had its own training algorithm

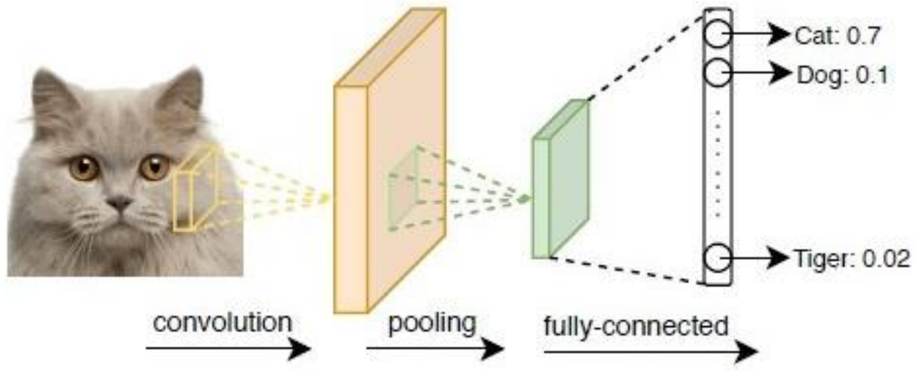
- In 1989, LeCun proposed CNN that was trained by backpropagation



Artificial Neural Networks. CNNs

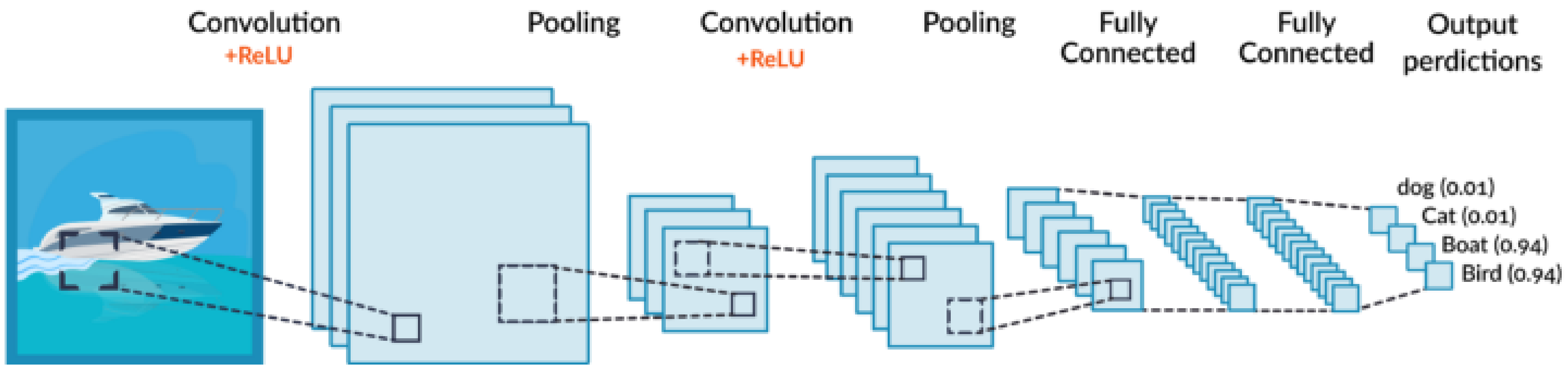
- CNN got popular when outperformed other models at ImageNet Challenge
  - Competition in object classification/detection
  - On hundreds of object categories and millions of images
  - Run annually from 2010 to present

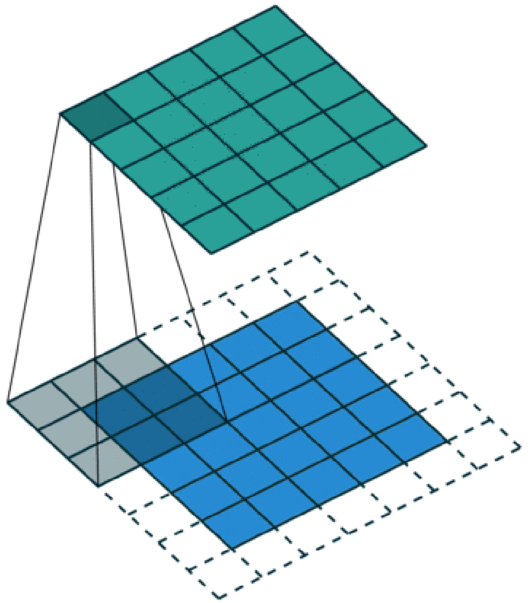
- Notable CNN architectures that won ImageNet challenge
  - AlexNet (2012), ZFNet (2013), GoogLeNet & VGG (2014), ResNet (2015)



Artificial Neural Networks. CNNs

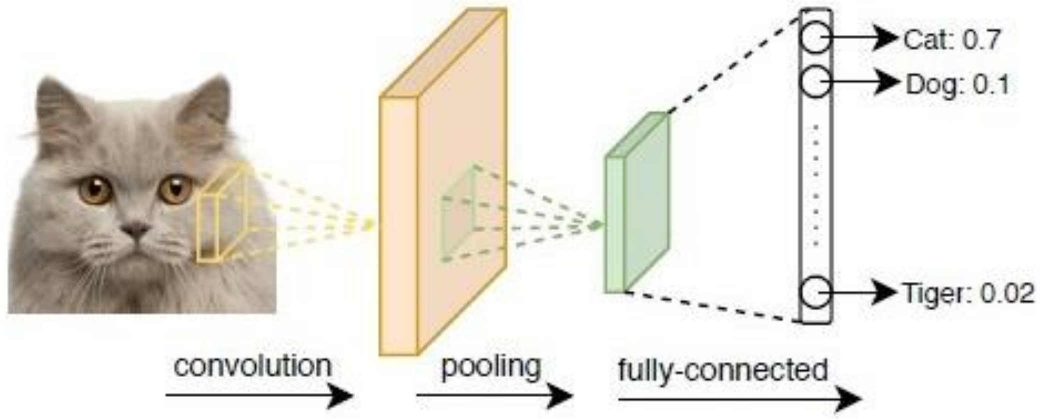
- A typical CNN has 4 types of layers:
  - Input layer
  - Convolution layer
  - Pooling layer
  - Fully connected layer





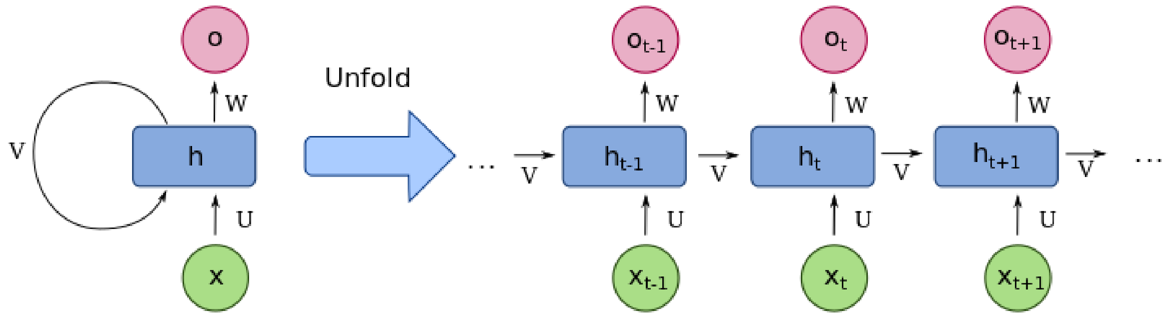
Artificial Neural Networks. CNNs

- The benefit of using CNNs is their ability to develop an internal representation of a two-dimensional image. This allows the model to learn position and scale in variant structures in the data, which is important when working with images.
- CNN are suitable for:
  - Image data
  - Classification prediction problems
  - Regression prediction problems
- They can be used aslo for:
  - Text data
  - Time series data
  - Sequence input data



Artificial Neural Networks. RNNs

- RNNs were designed to work with sequence prediction problems.
- Sequence prediction problems come in many forms and are best described by the types of inputs and outputs supported. Some examples of sequence prediction problems include:
    - **One-to-Many**: An observation as input mapped to a sequence with multiple steps as an output.
    - **Many-to-One**: A sequence of multiple steps as input mapped to class or quantity prediction.
    - **Many-to-Many**: A sequence of multiple steps as input mapped to a sequence with multiple steps as output. They are also known as **sequence-to-sequence** or **seq2seq.**

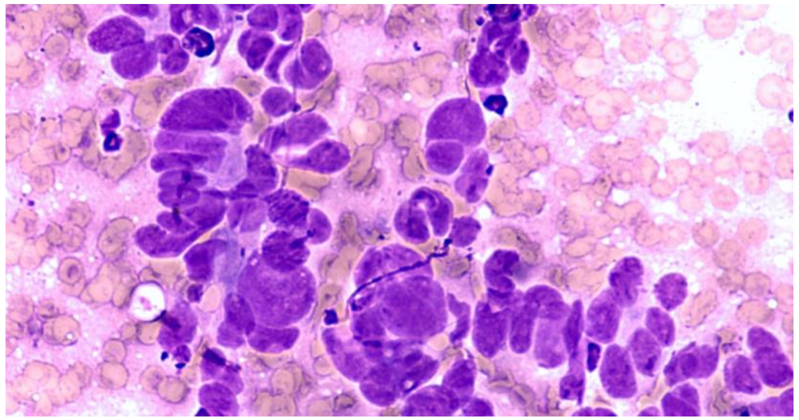


Artificial Neural Networks. RNNs

- RNNs have received the most success when working with sequences of words and paragraphs, generally called **natural language processing**.
- This includes both sequences of text and sequences of spoken language represented as a time series. They are also used as **generative models**.
- RNN are suitable for:
    - Text data
    - Speech data
    - Classification prediction problems
    - Regression prediction problems
    - Generative models

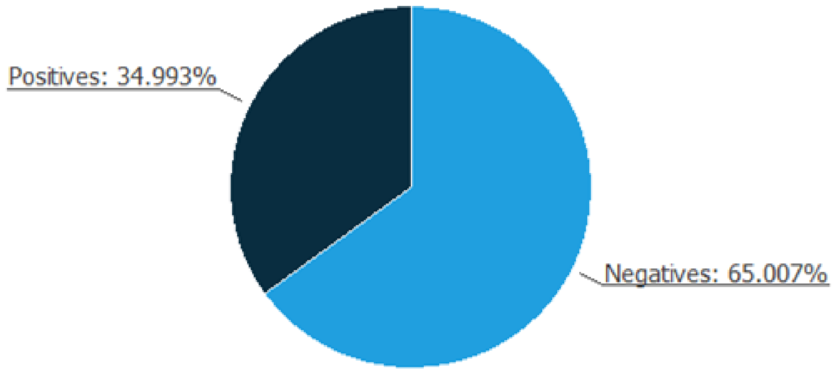
Deep learning. Sample applications

- Breast cancer prediction
  - Assess whether a lump in a breast is malignant (**cancerous**) or benign (**non-cancerous**) from digitized images of a fine-needle aspiration biopsy.
  - The dataset contains 30 features from the images.



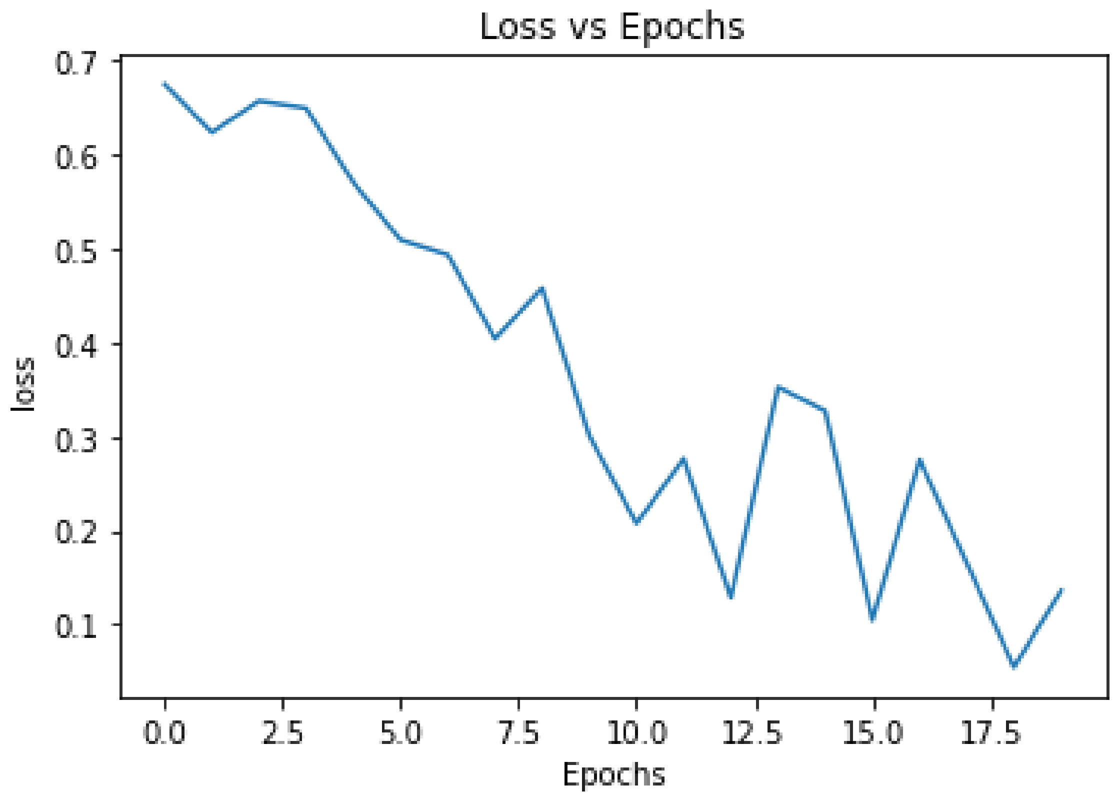
- Training dataset, to train the ANN
  - malignant or benign cases.
- Testing dataset, never seen by the ANN during the training phase:
  - guarantee not over-fitting the ANN to training dataset

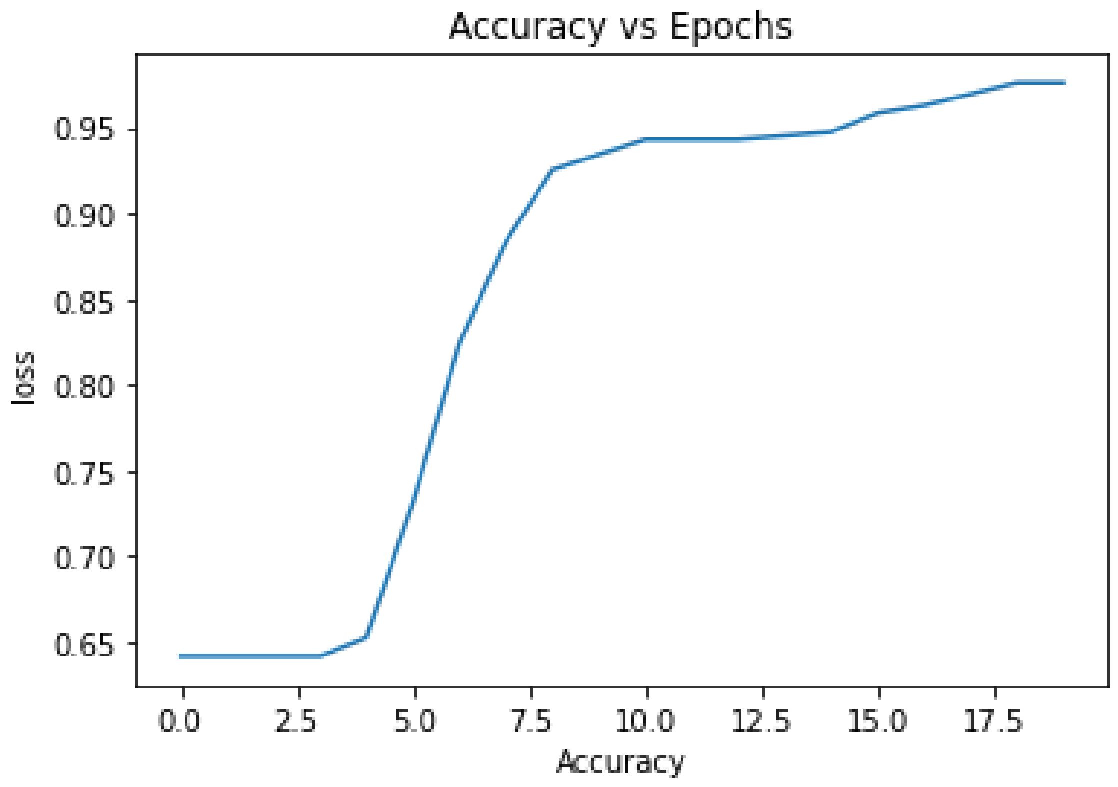
https://drive.google.com/file/d/1tW1UymqY9gq_UrTY9uUnsC1jwUo_ufJx/view?usp=sharing



Deep learning. Sample applications

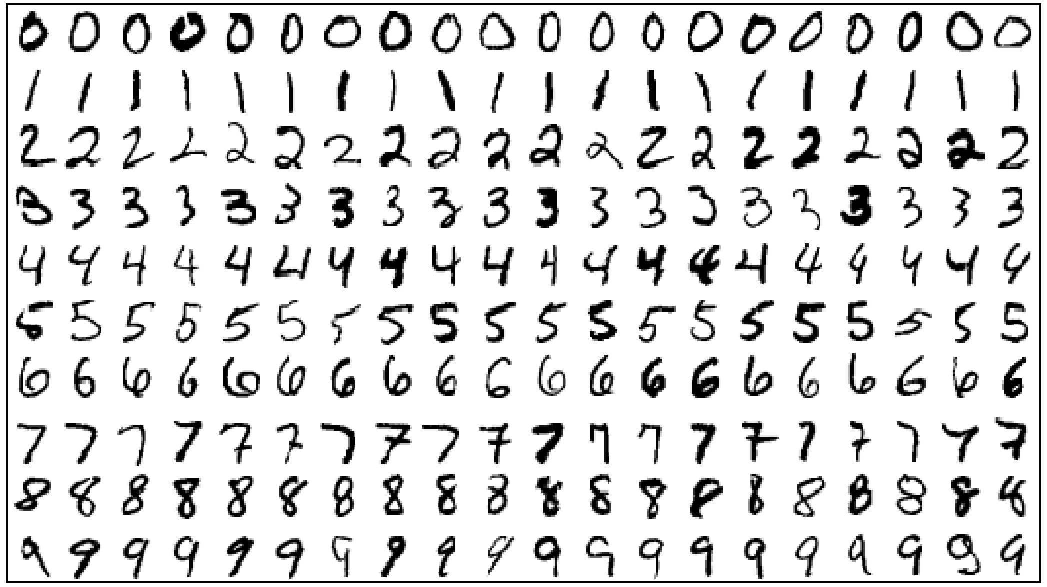
- Breast cancer prediction: results





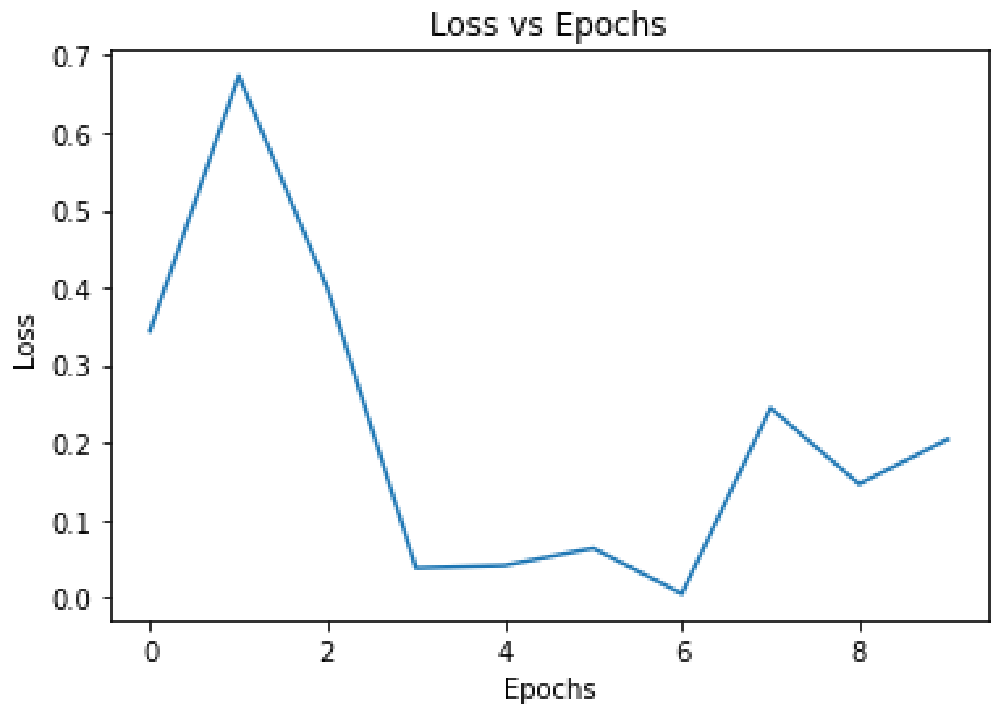
Deep learning. Example: MNIST - MLP vs CNN

- Handwritten digits classification (MNIST repository).
- MLP to assign each image a label in the set {0,1,...,9}
- Training dataset, to train the ANN → 'what numbers 0 through 9 look like'.
- Testing dataset, never seen by the ANN during the training phase:
  - guarantee the ANN is not over-fitted to the training dataset,
  - assure the ANN can label independent items properly.

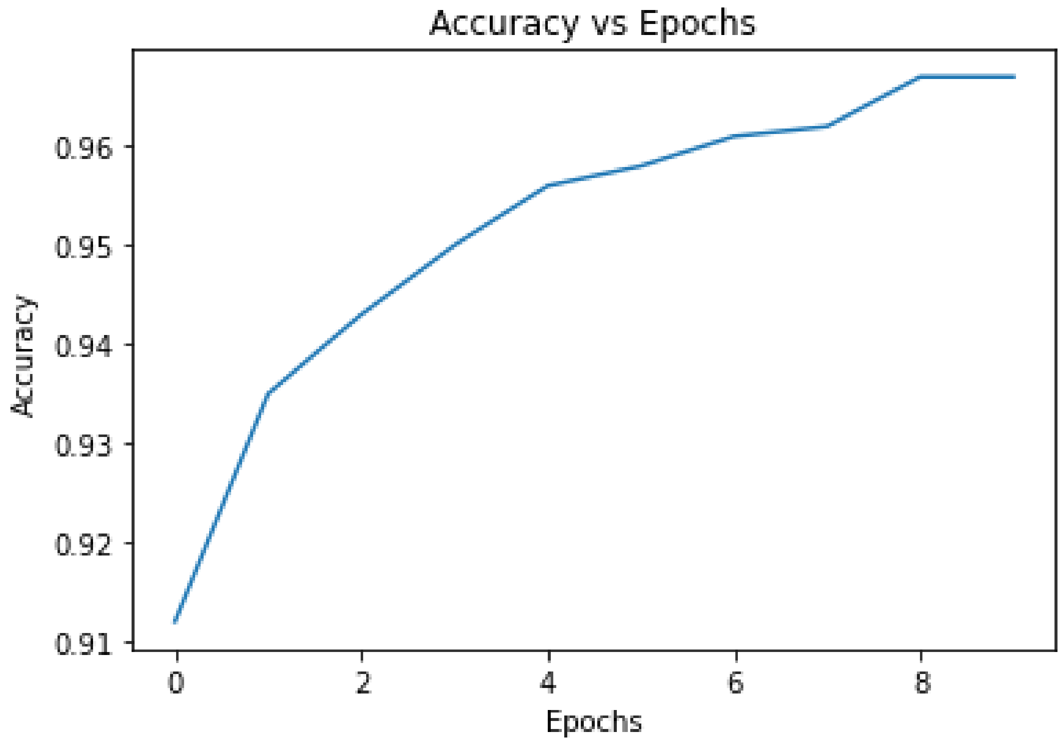


Deep learning. Example: MNIST - MLP vs CNN

- **MLP** for handwritten digits classification: https://drive.google.com/file/d/1hHih46Z2Eth7sXCGHTgqv9CYr_PY8i5S/view?usp=sharing



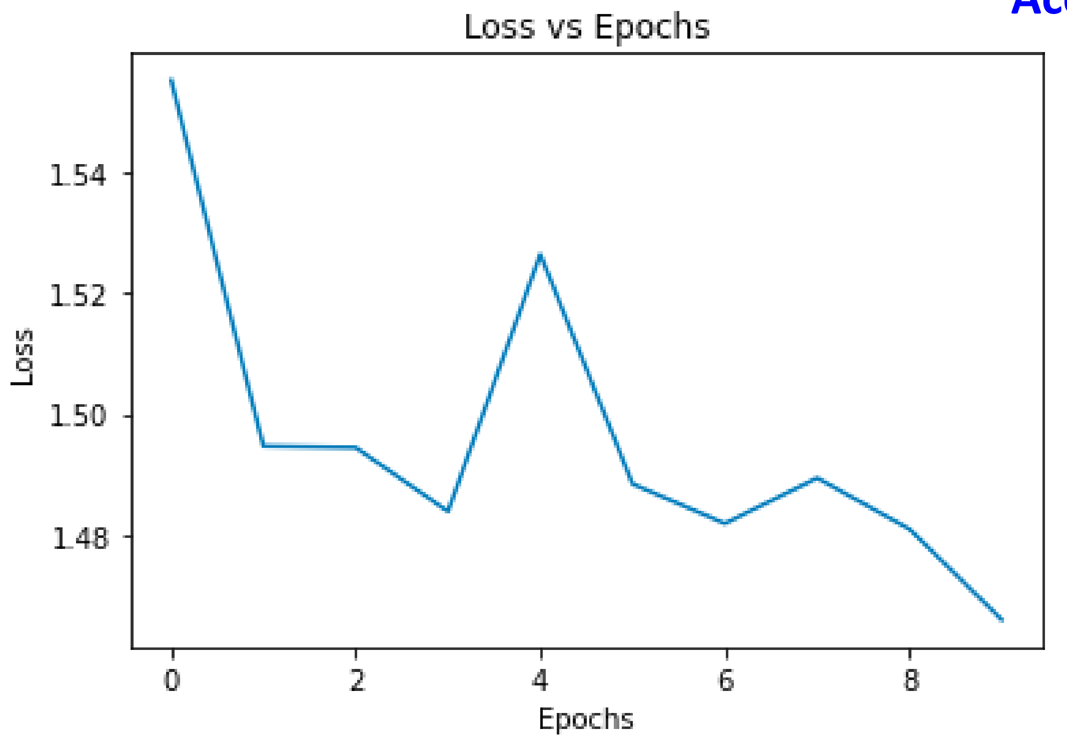
Accuracy = 96.1%

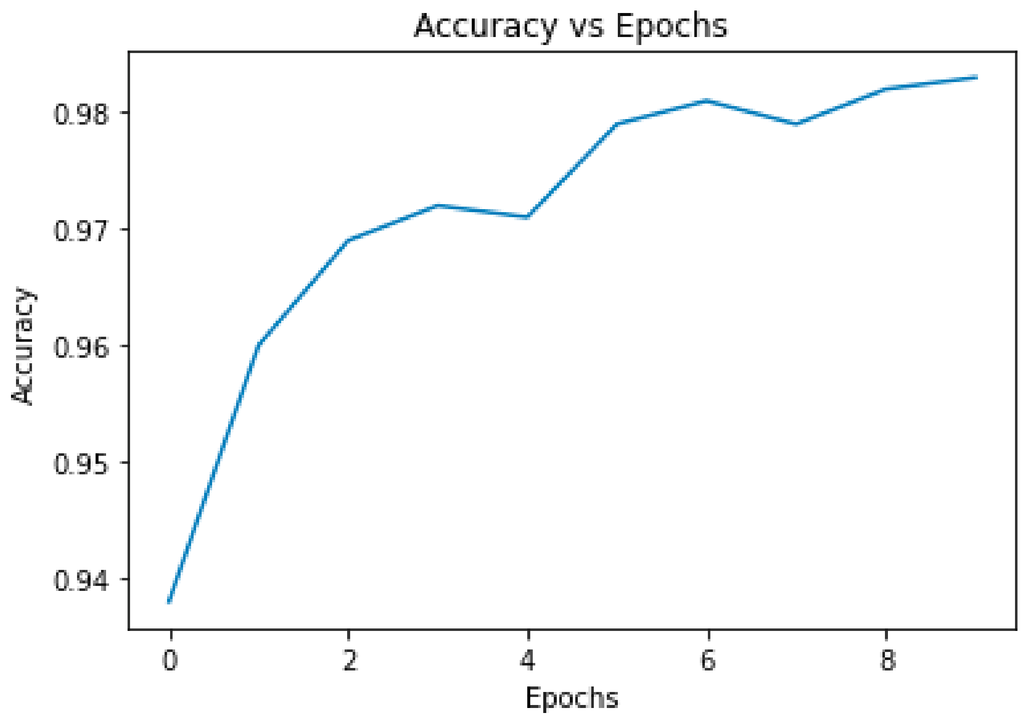


Deep learning. Example: MNIST - MLP vs CNN

- **CNN** for handwritten digits classification: https://drive.google.com/file/d/12bAg-w9eWJChU-AXtRx8f42bdLNvIwGj/view?usp=sharing

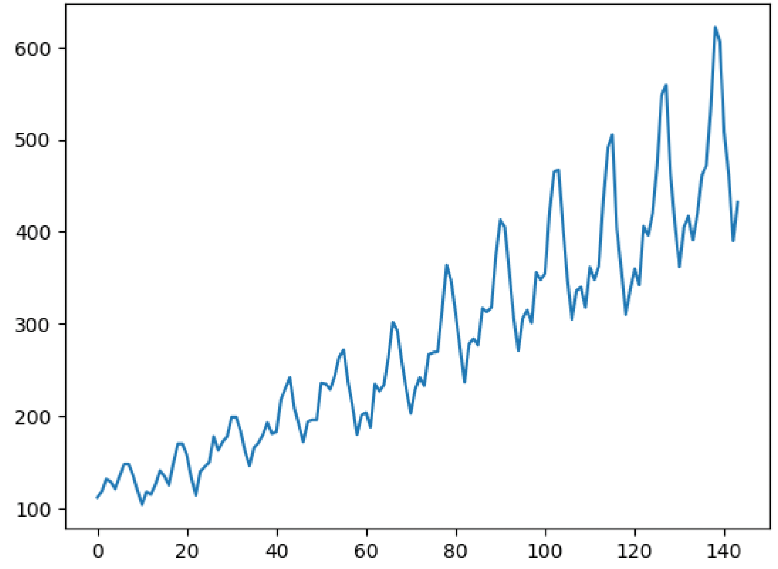
Accuracy = 98.3%





Deep learning. Example: Basic use of RNN

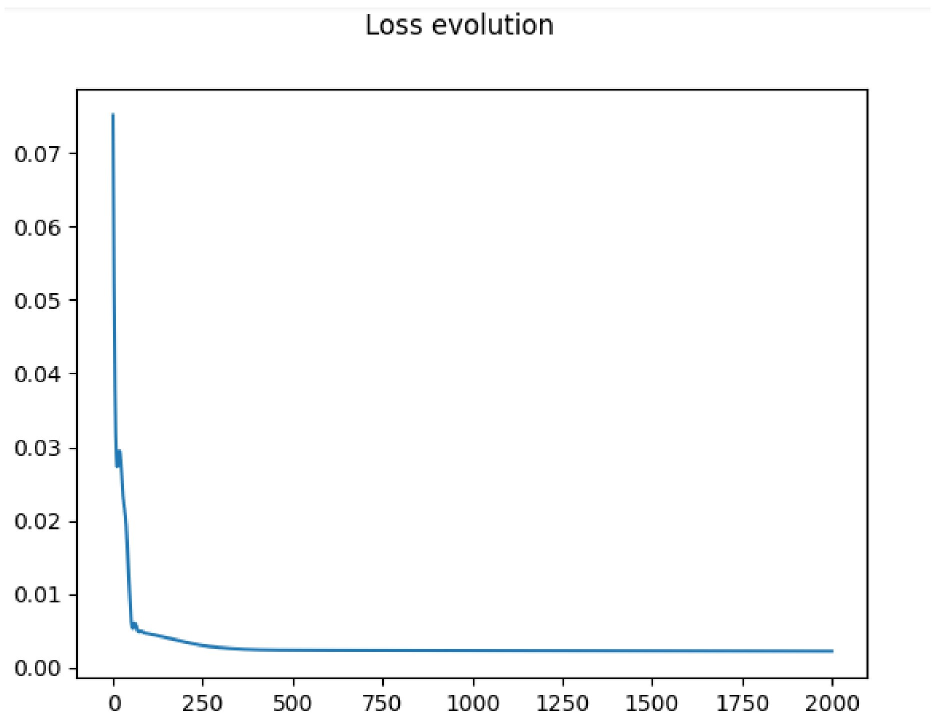
- RNN based on LSTM cells → Many-to-One example
- Training dataset is a sequence of passengers of an Airline (80% of data)
- Testing dataset never seen by the ANN during the training phase (i.e, the next 20% of the sequence)

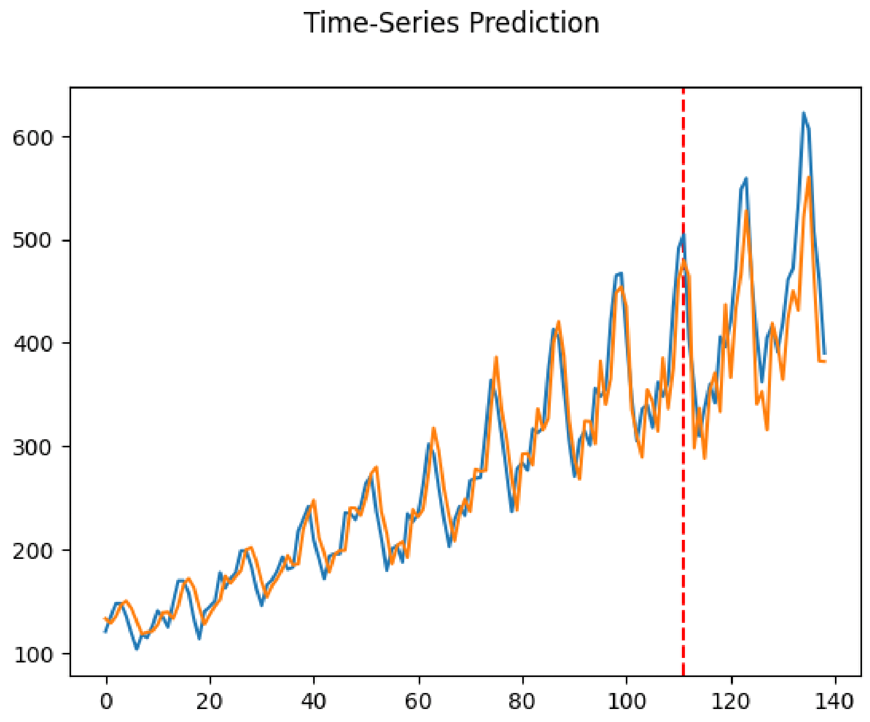


https://colab.research.google.com/drive/1sNDE114j7xAIIxaYbs3G2A8TtpC_DyNc?usp=sharing

Deep learning. Example: Basic use of RNN

- RNN for sequence prediction https://colab.research.google.com/drive/1sNDE114j7xAIIxaYbs3G2A8TtpC_DyNc?usp=sharing



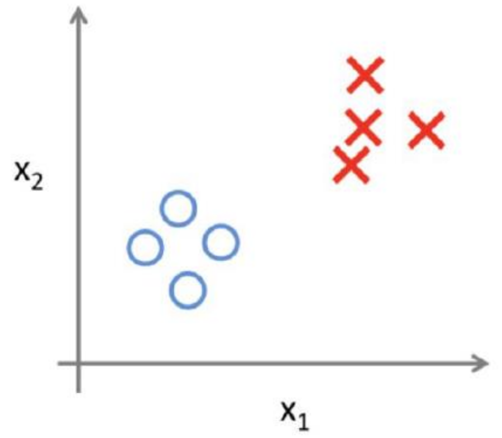


Generative Models

Supervised vs Unsupervised Learning

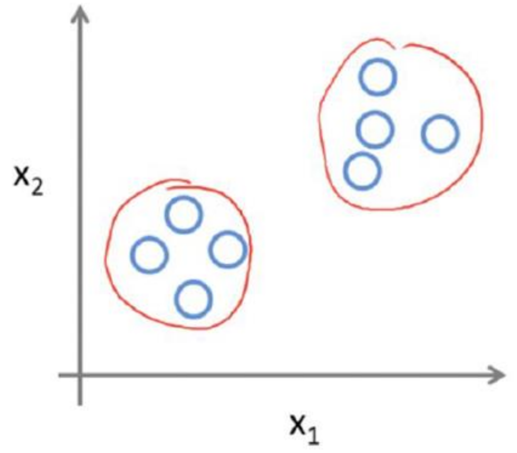
Supervised Learning

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



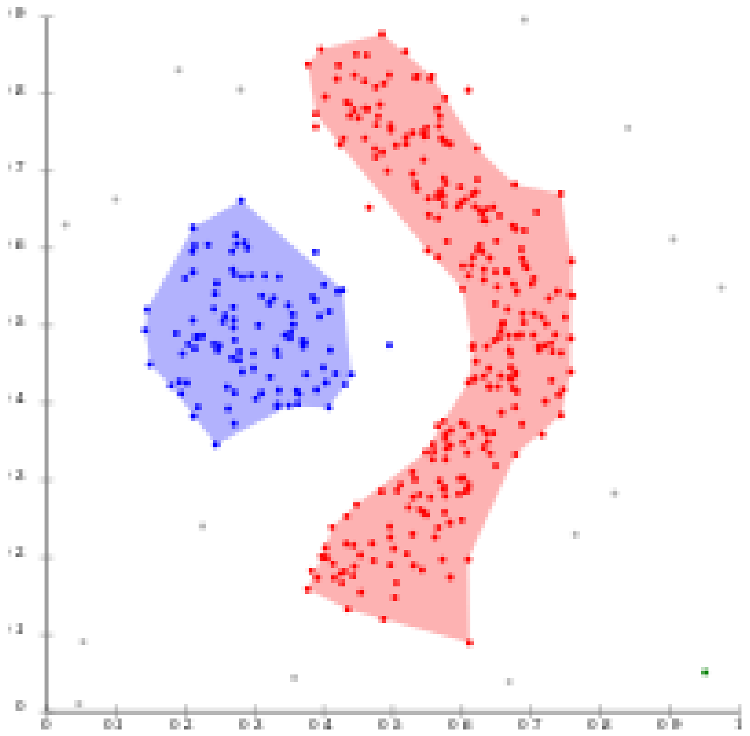
Unsupervised Learning

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

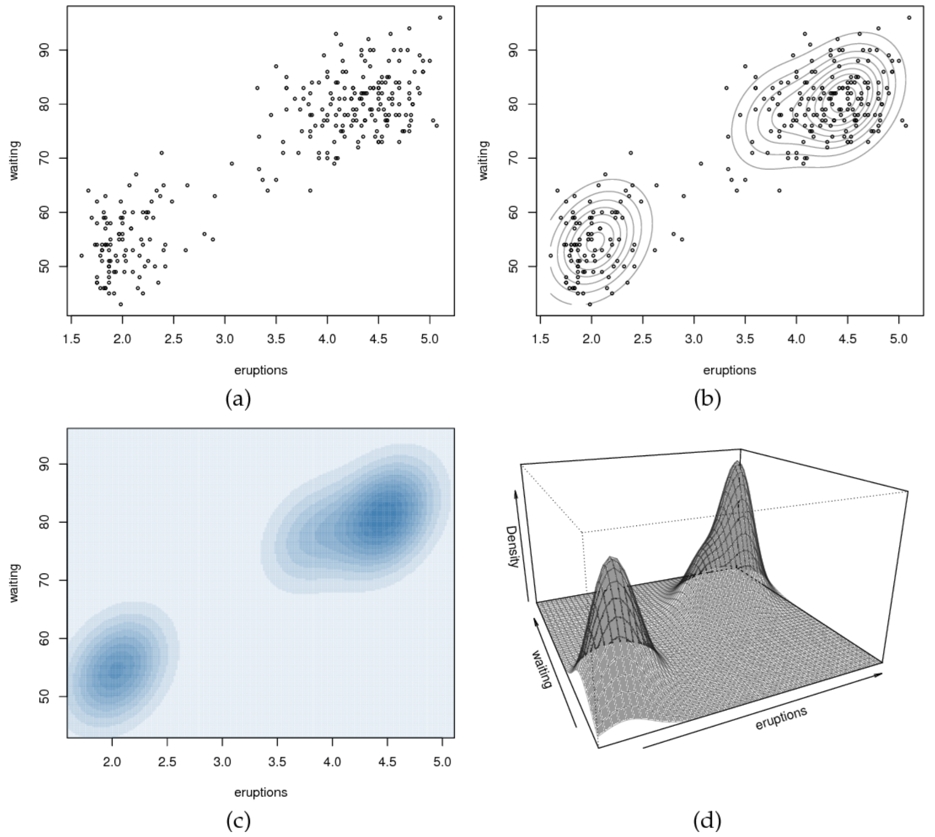


Unsupervised Learning

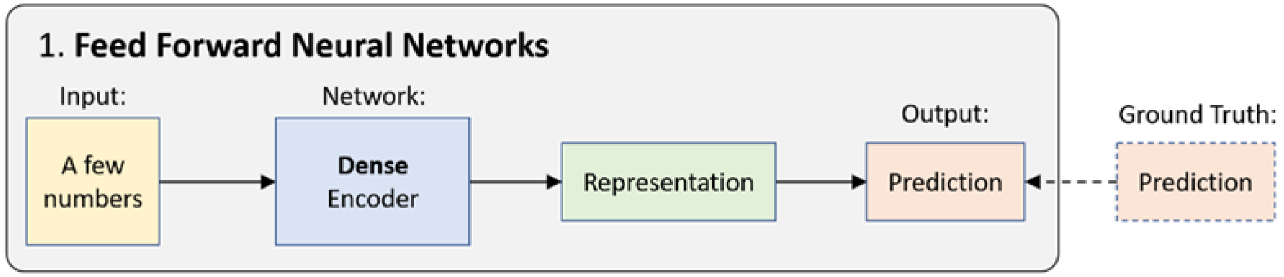
Clustering

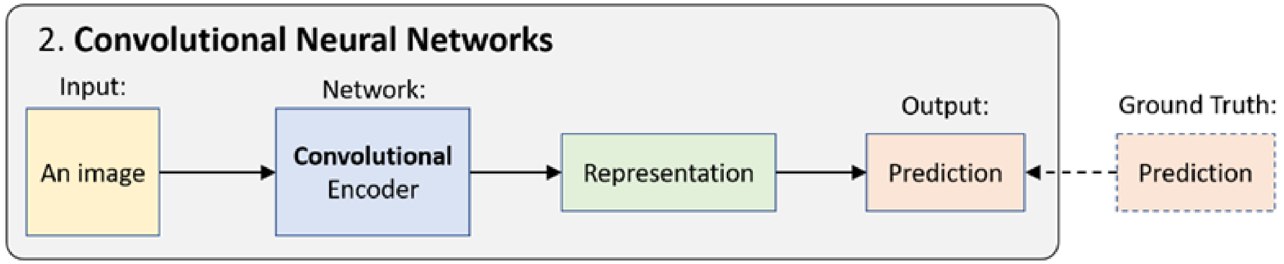


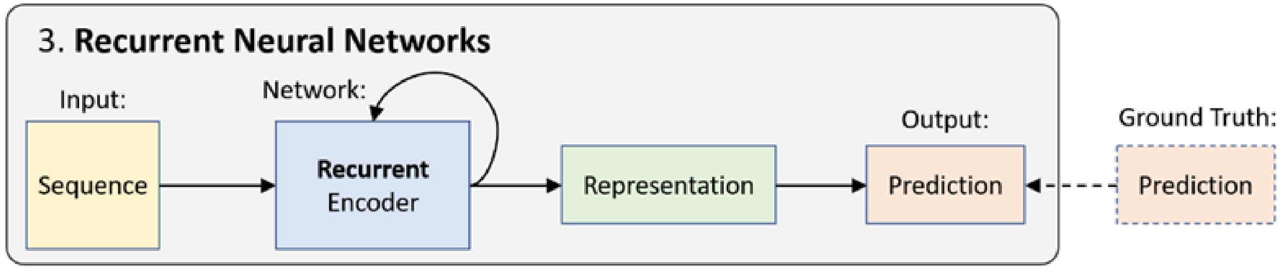
Density estimation

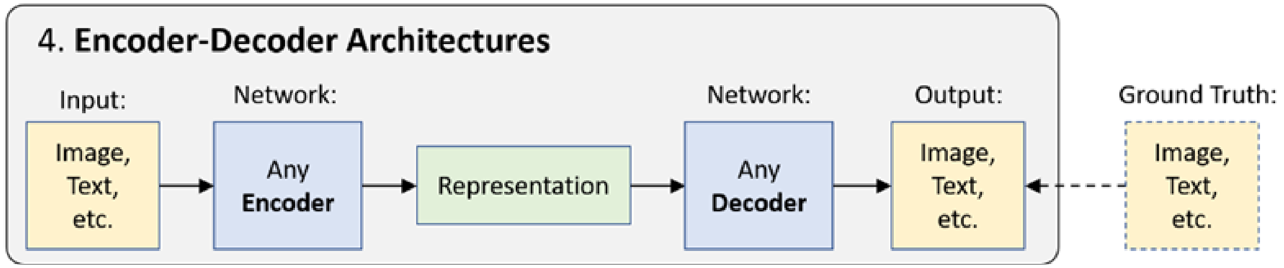


Supervised Learning

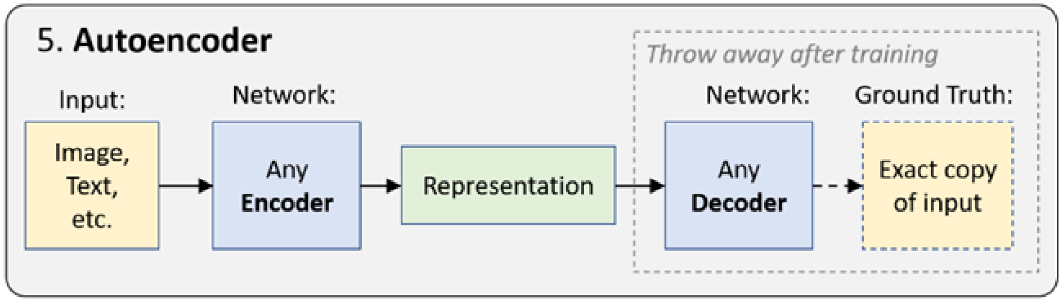


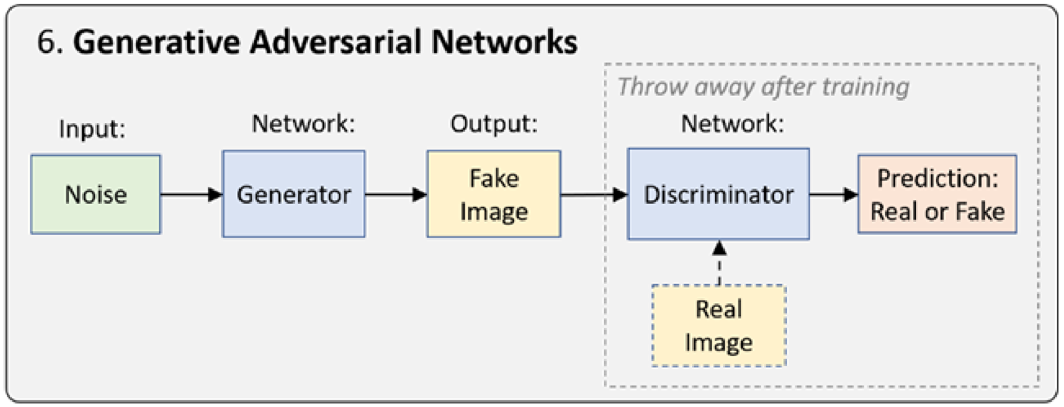




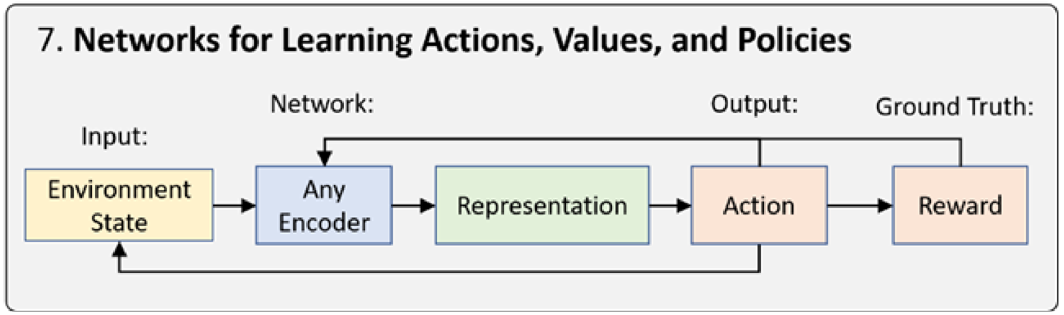


Unsupervised Learning





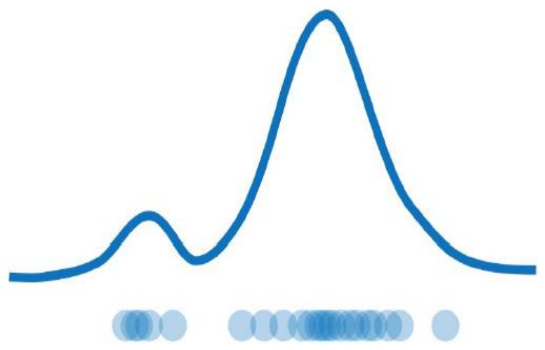
Reinforcement Learning



Generative Modeling

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

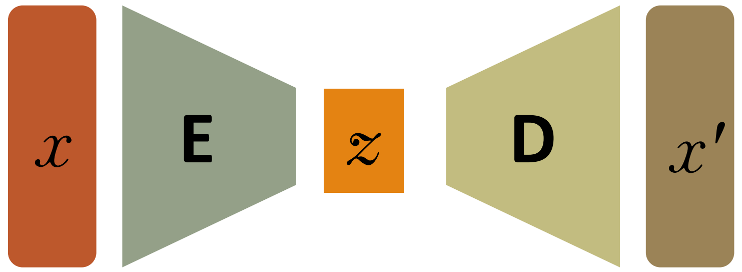
- **Density estimation**



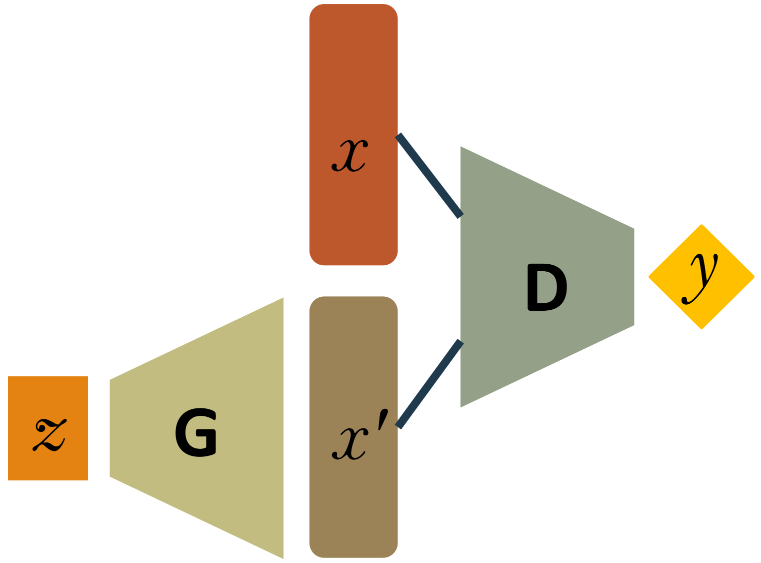
  - Understand better the data distribution
  - Compress the data representation
  - Generate samples

Models

Autoencoders and Variational Autoencoders (VAEs)



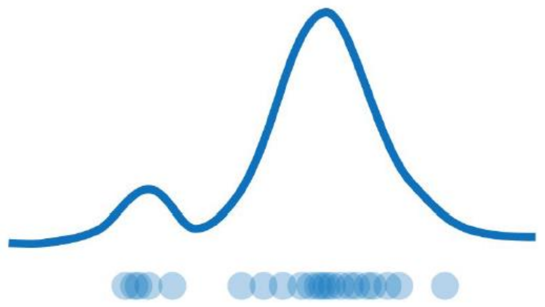
Generative Adversarial Networks (GANs)



Generative Modeling

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

- **Density estimation**

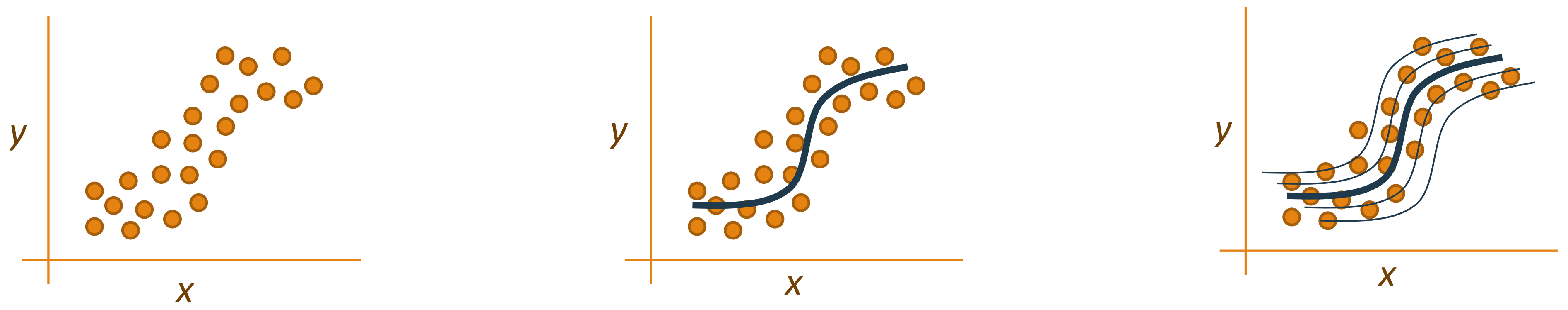


- **Synthetic samples generation**

Training samples

Synthetic samples

Generating synthetic samples



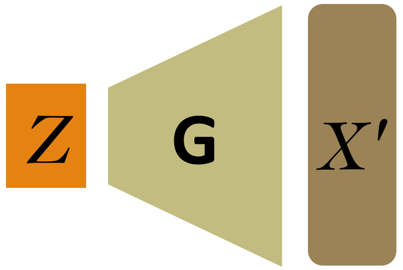
Generating synthetic samples

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

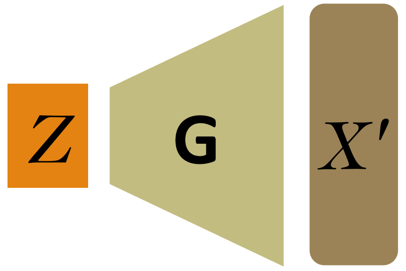
Solution: Sampling from something simple (noise) and learning a transformation to the real (training) distribution

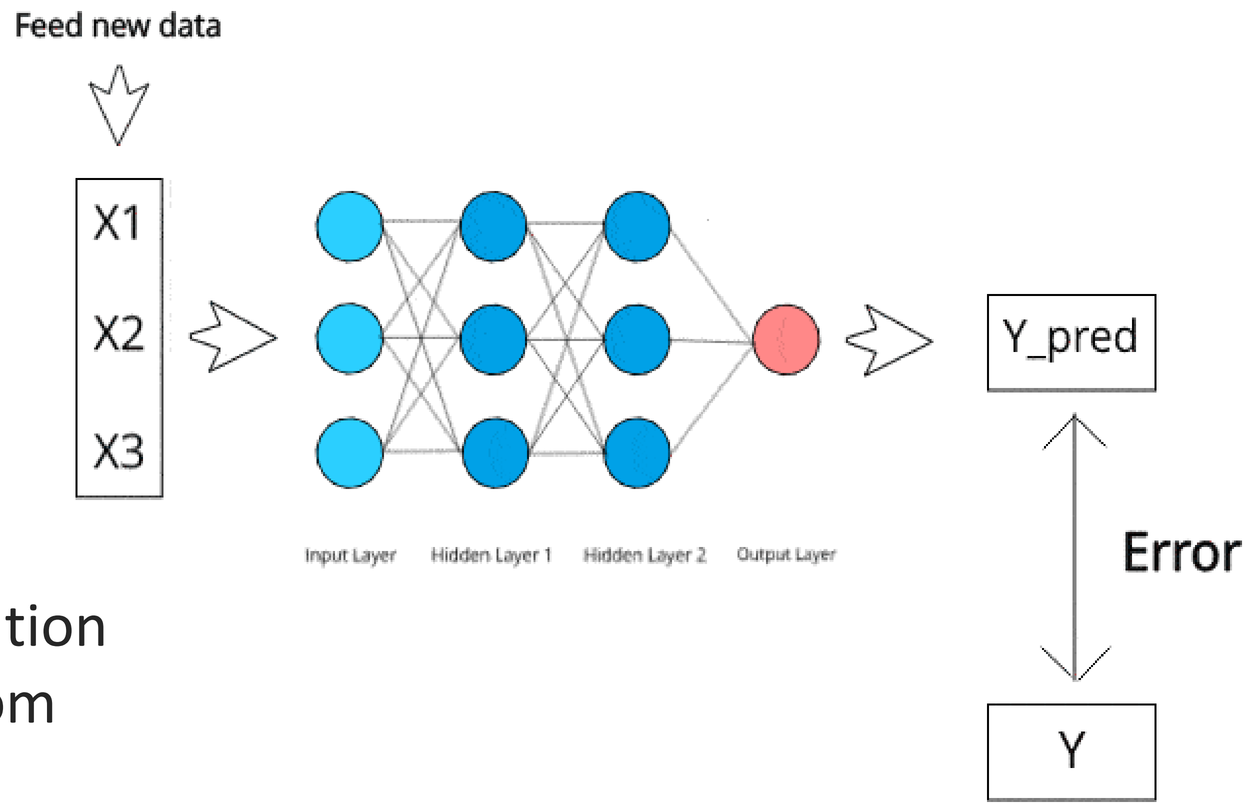
Main components of the **Generative Model:**

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



How Generator Learn?





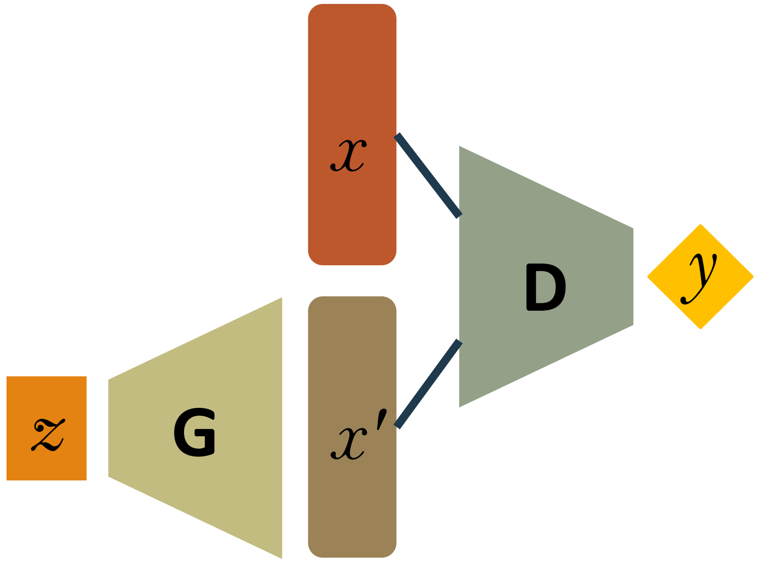
Using **another model** that gives information about how close/far are the samples from real data → **Discriminator**

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**)

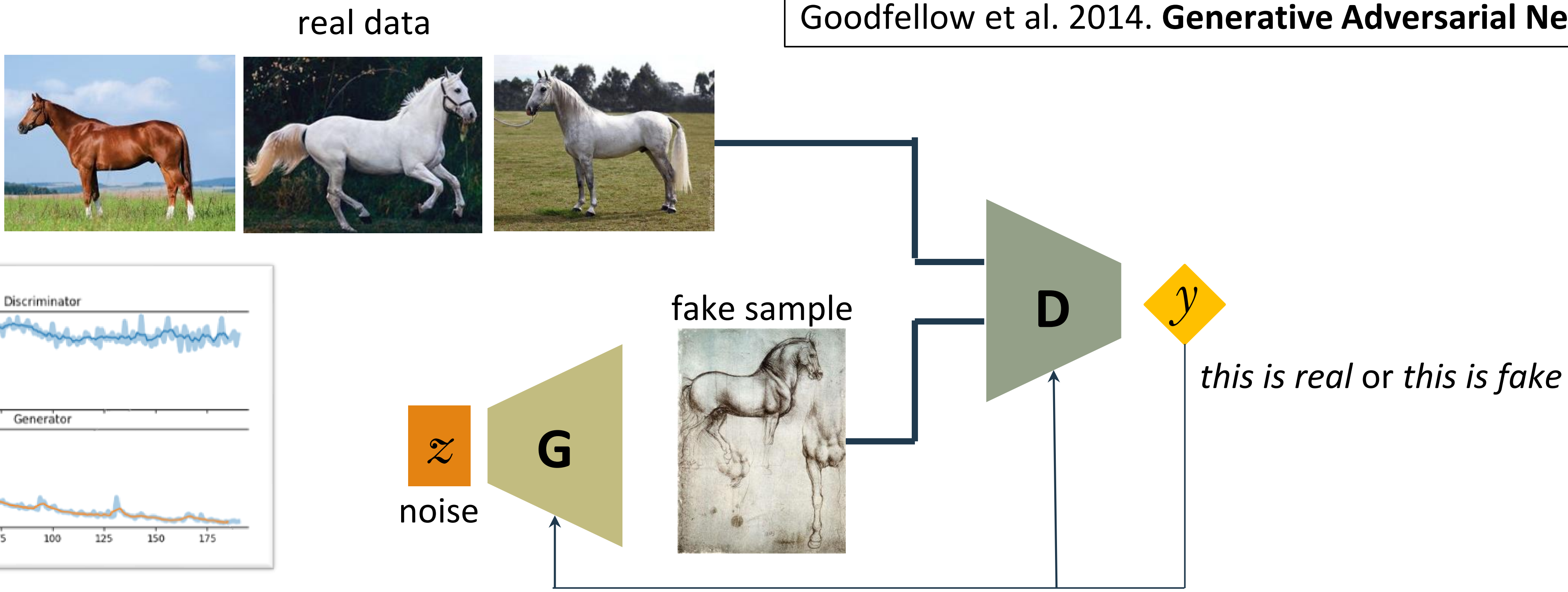
- Generator (**G**) learns how to create synthetic/fake data samples (X') by sampling random noise (Z) to fool the discriminator (**D**)

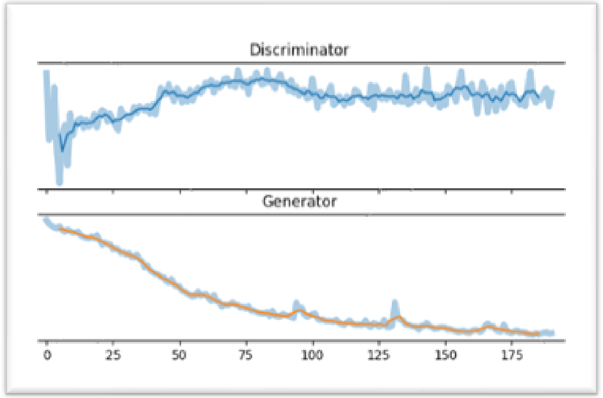


Generative Adversarial Networks

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

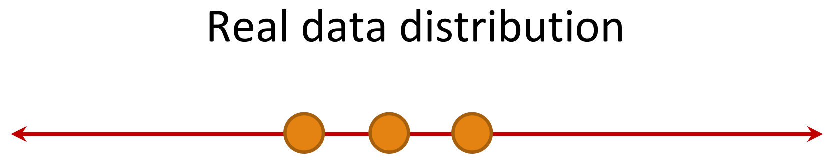
Goodfellow et al. 2014. **Generative Adversarial Nets**





GAN Training

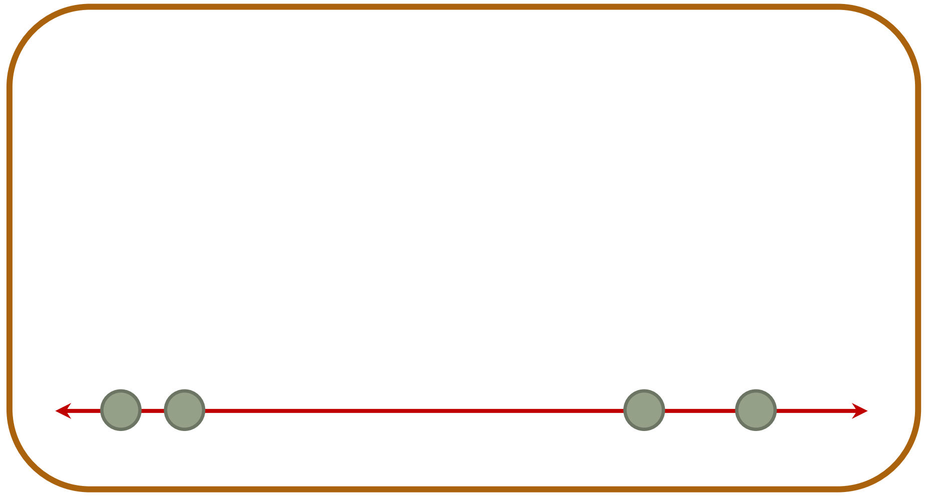
GAN training intuition



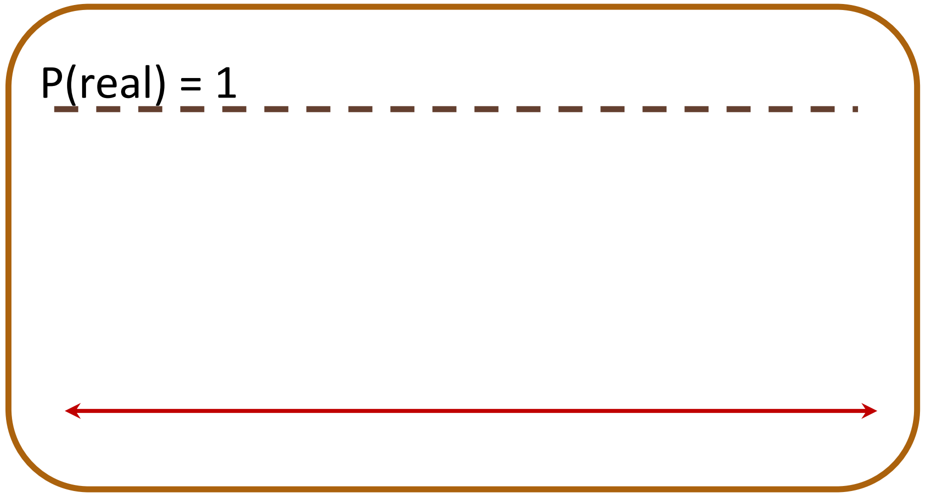
GAN Training

Step 1:

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



Generator



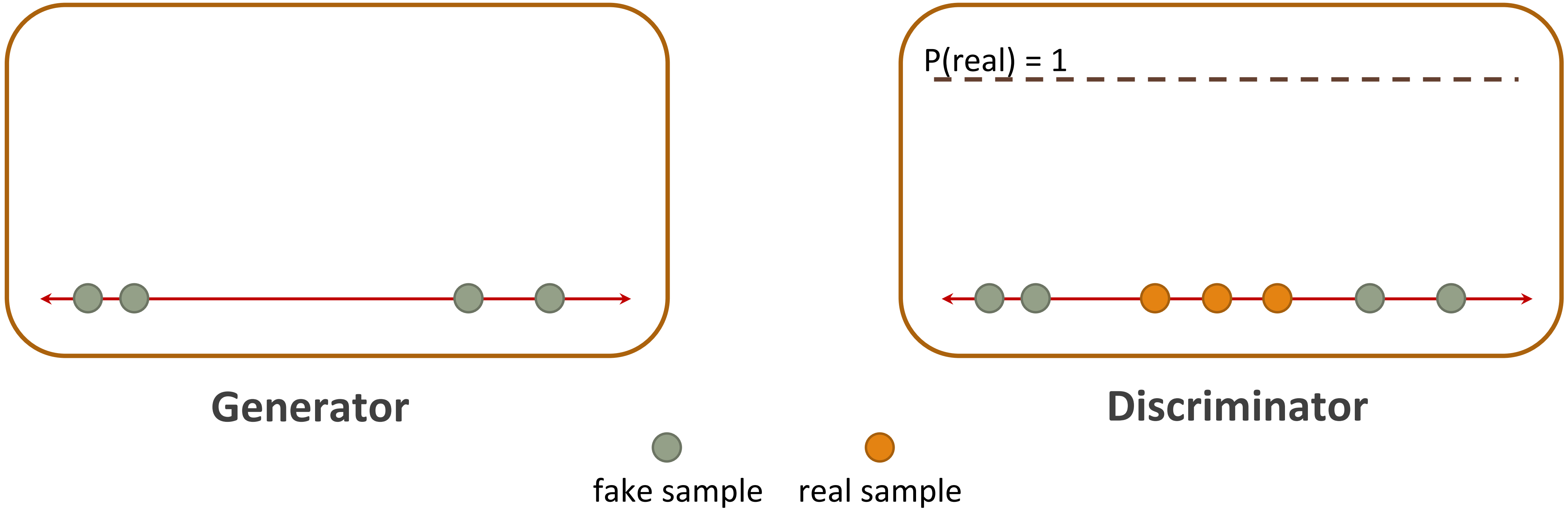
Discriminator

● fake sample ● real sample

GAN Training

Step 2:

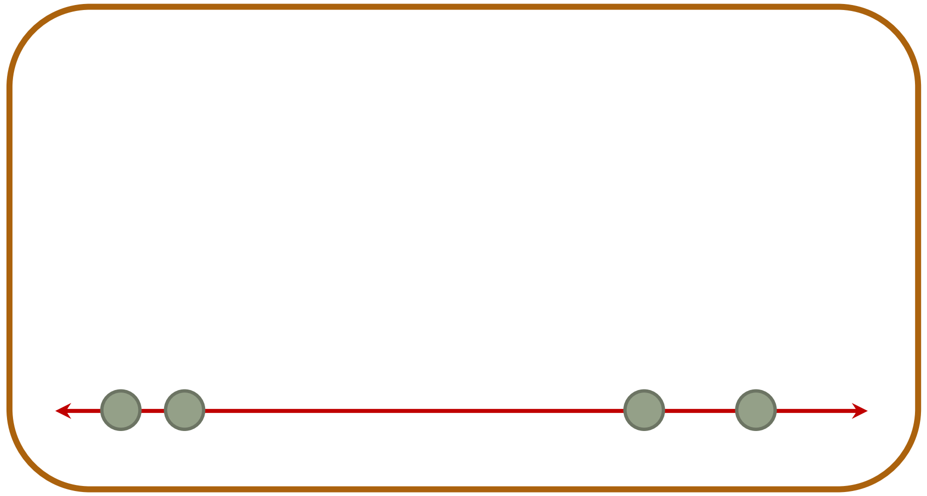
Discriminator gets fake samples from the generator and real samples from the real data distribution



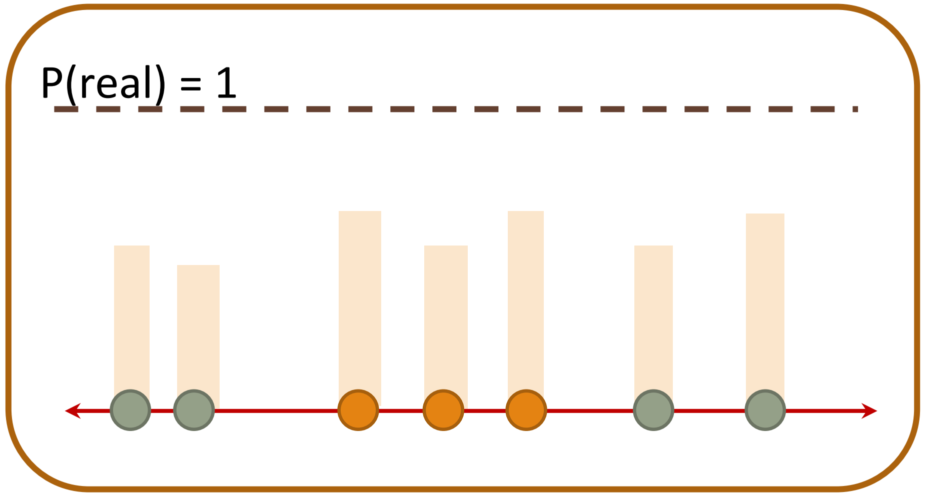
GAN Training

Step 3:

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



Generator



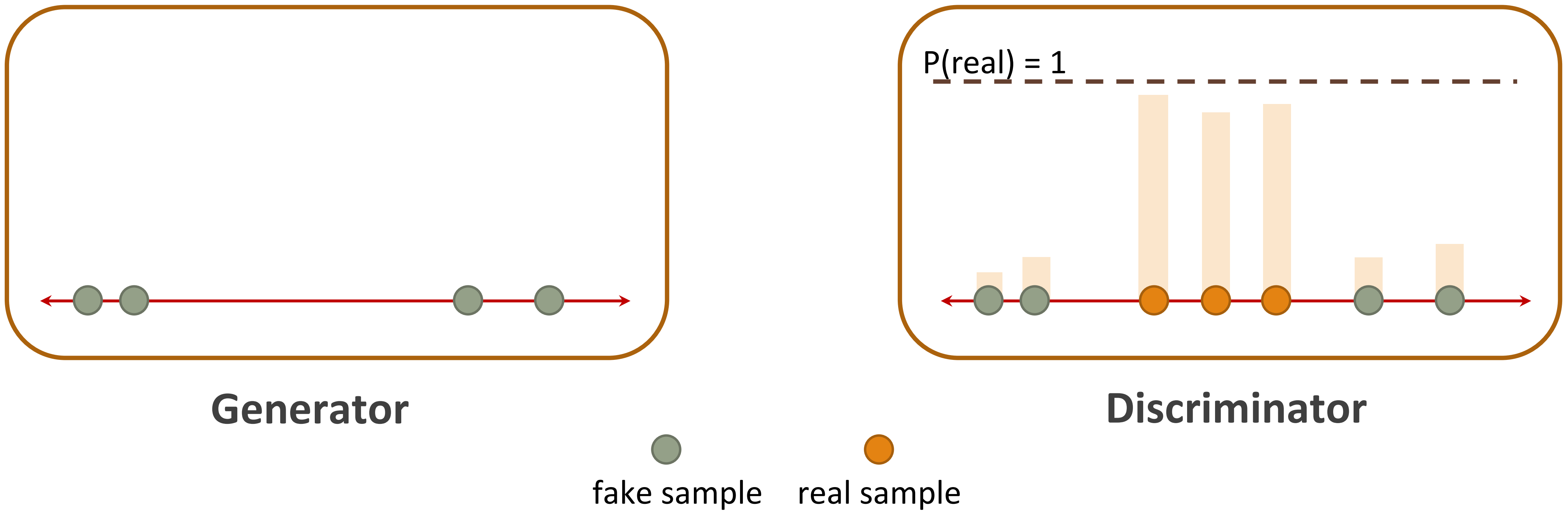
Discriminator

● fake sample ● real sample

GAN Training

Step 3:

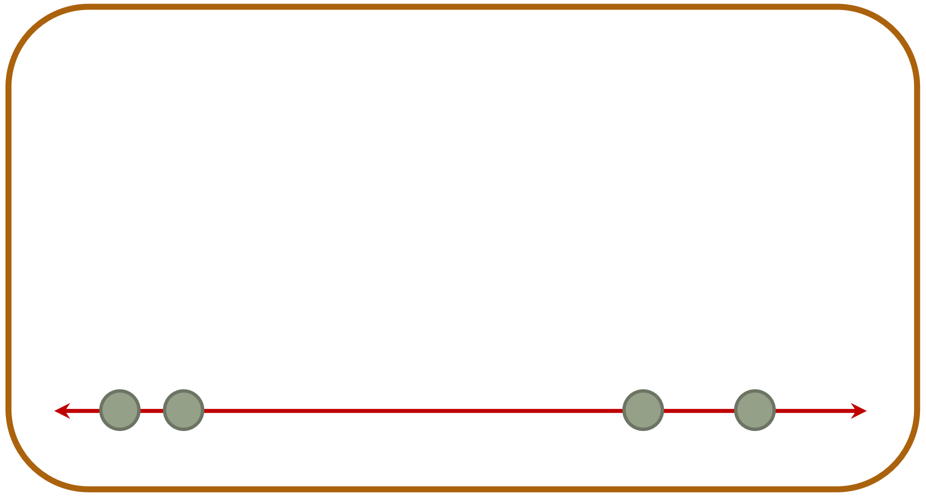
Discriminator learn how to distinguish between real and fake data (*supervised learning*)



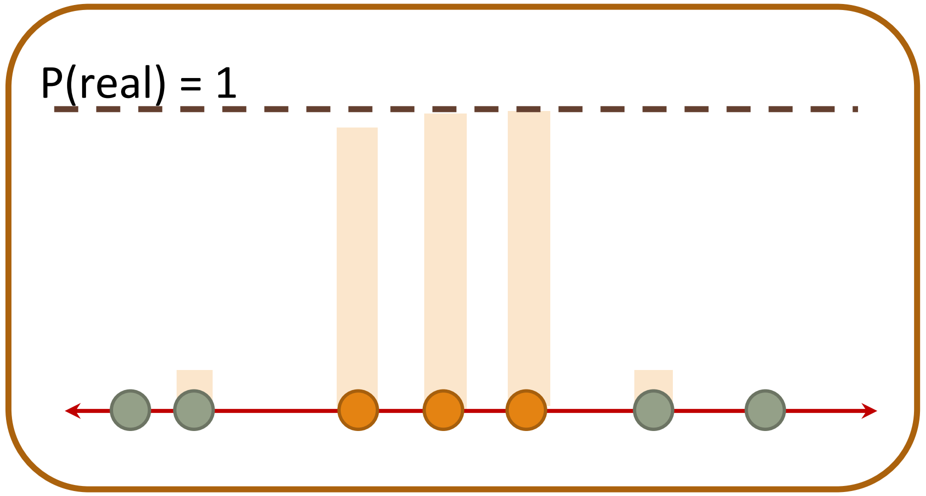
GAN Training

Step 3:

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



Generator



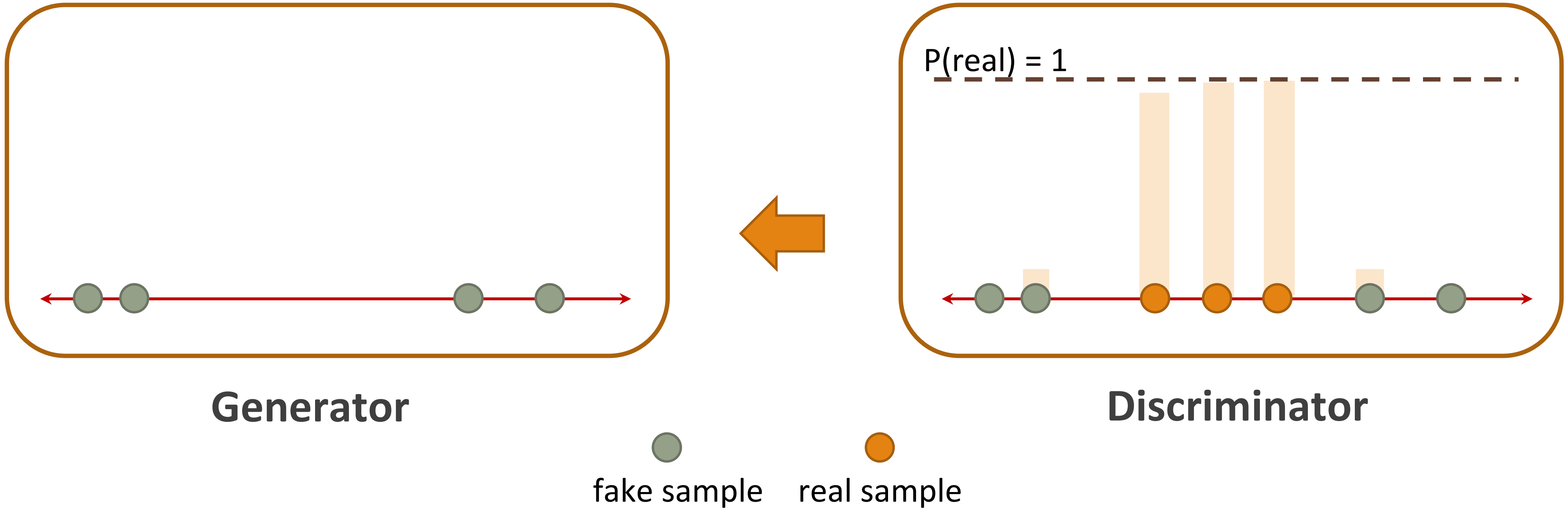
Discriminator

● fake sample ● real sample

GAN Training

Step 4:

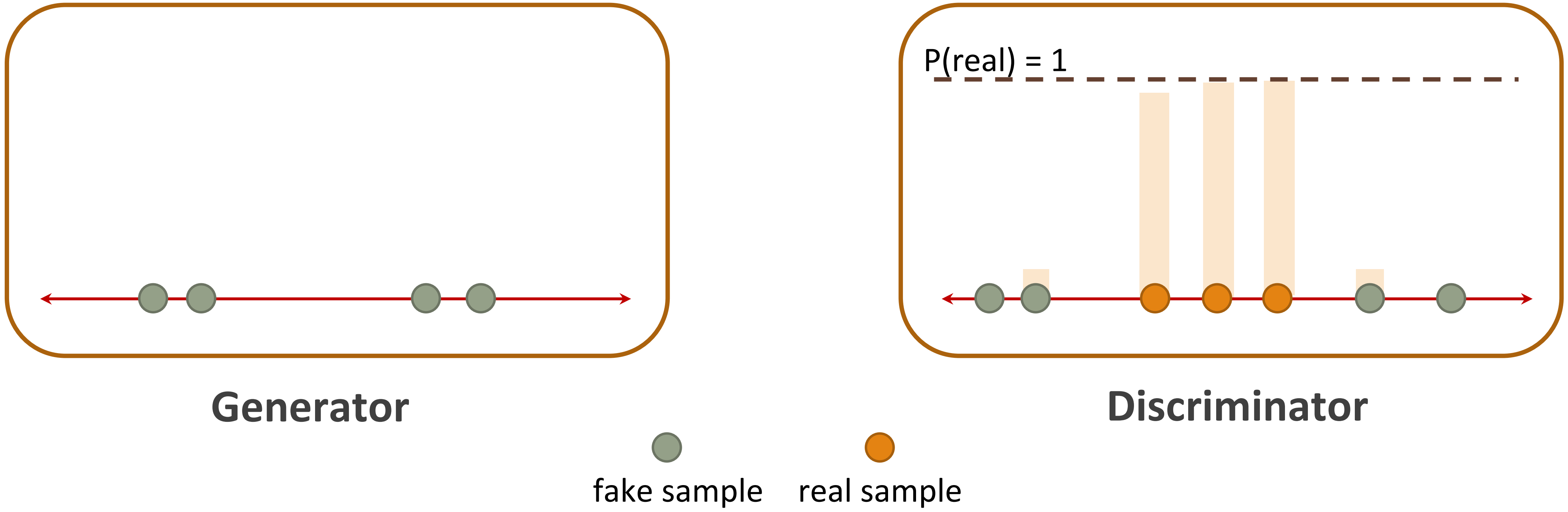
Generator get discriminator feedback and updates its parameters (weights) to improve the synthetic data



GAN Training

Step 4:

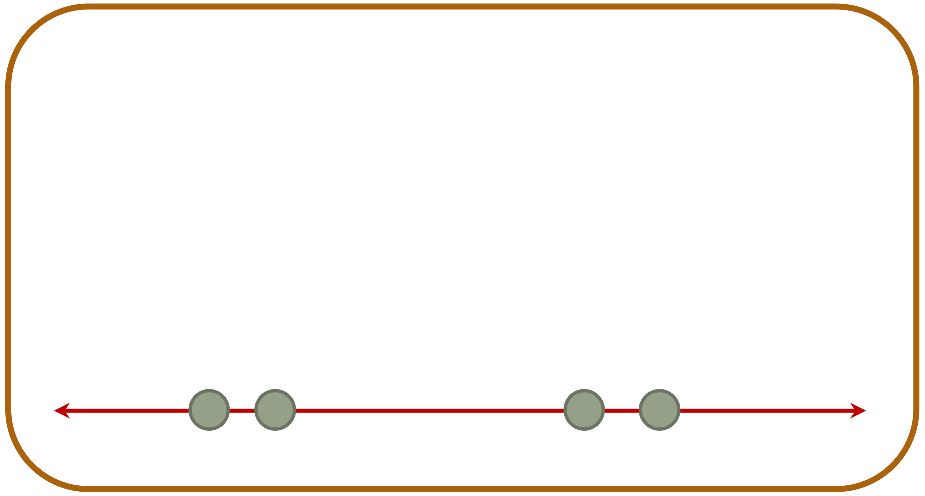
Generator get discriminator feedback and updates its parameters (weights) to improve the synthetic data

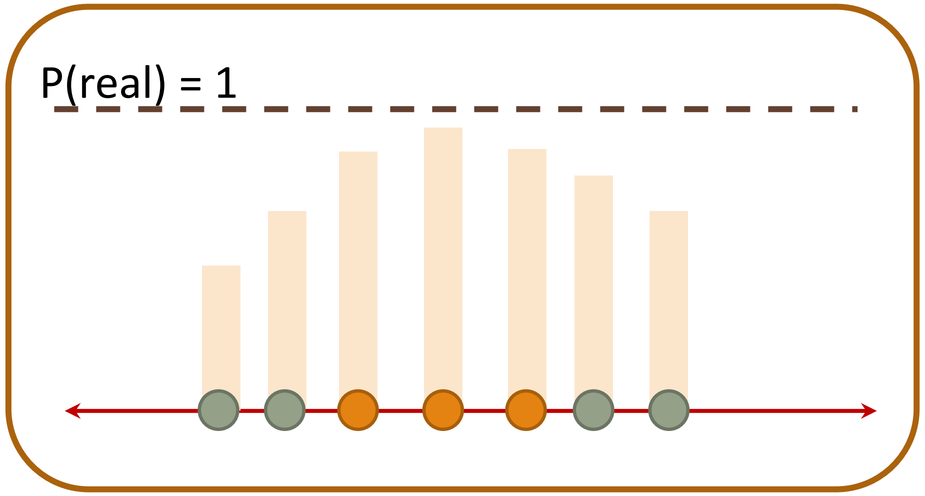


GAN Training

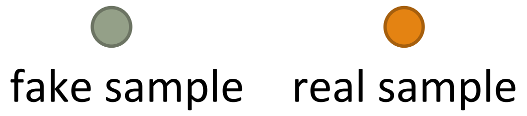
Step 5:

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


Generator



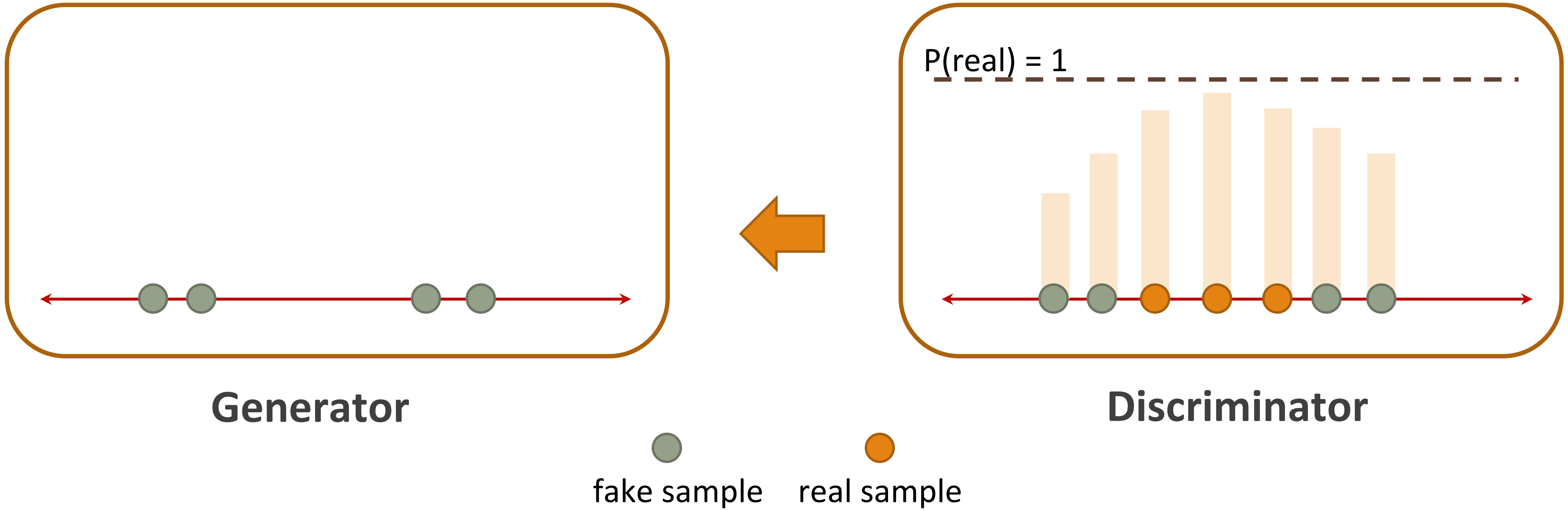
Discriminator



GAN Training

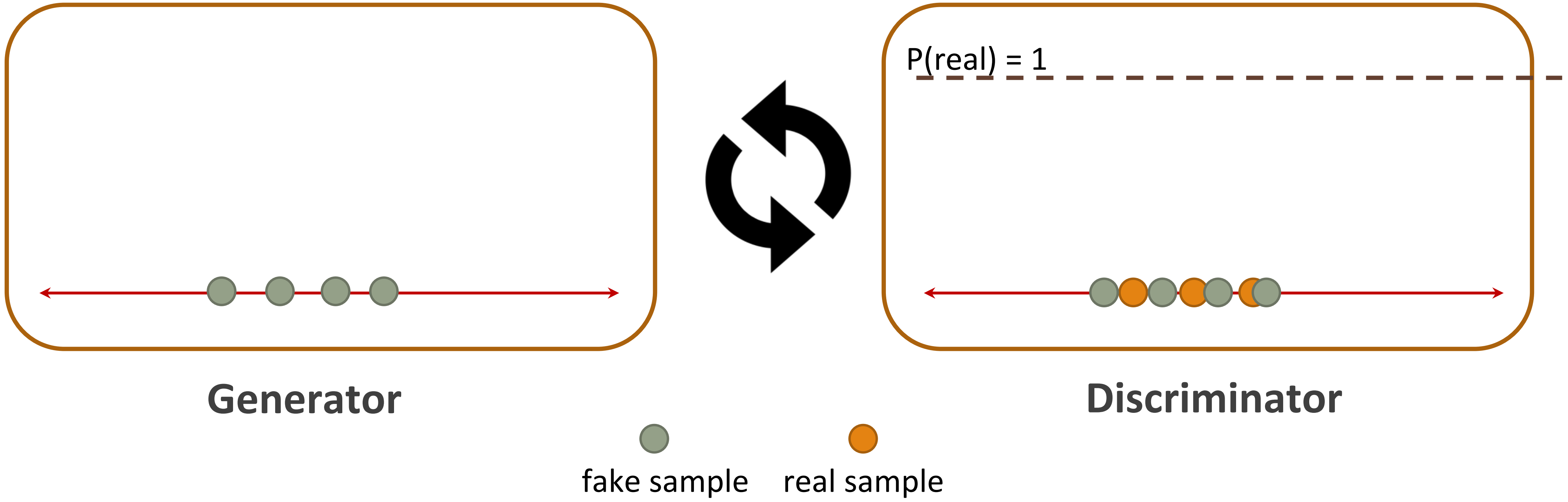
Step 5:

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



GAN Training

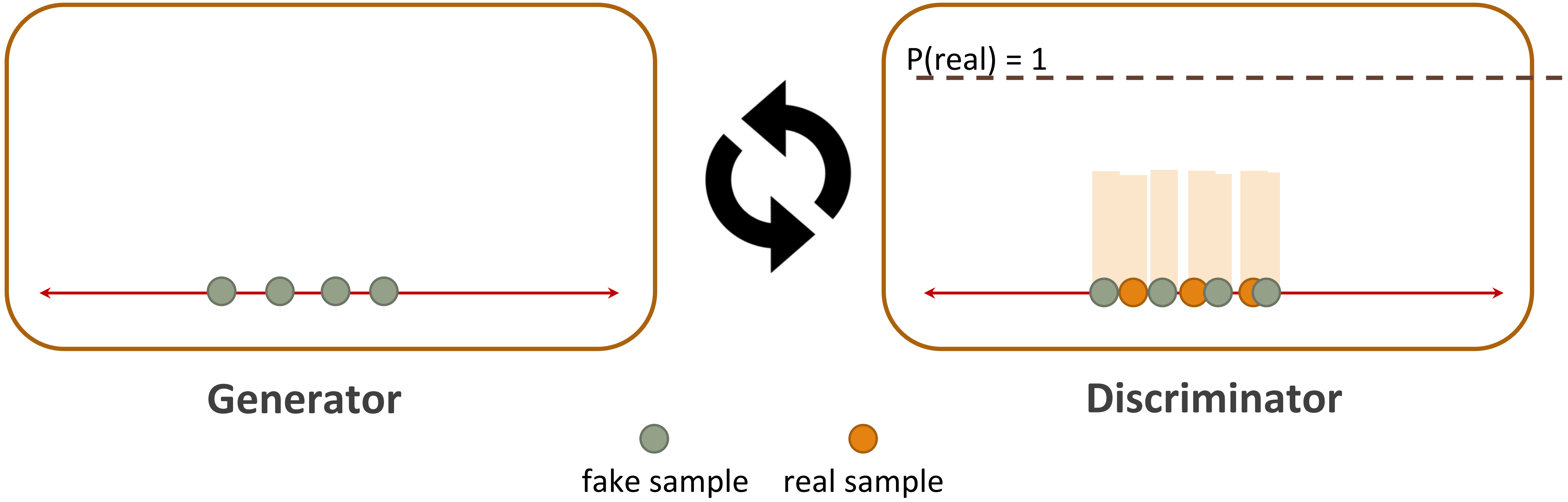
This steps are repeated until the generator is able to fool the discriminator by generating fake data samples that are indistinguishable from the real ones



GAN Training

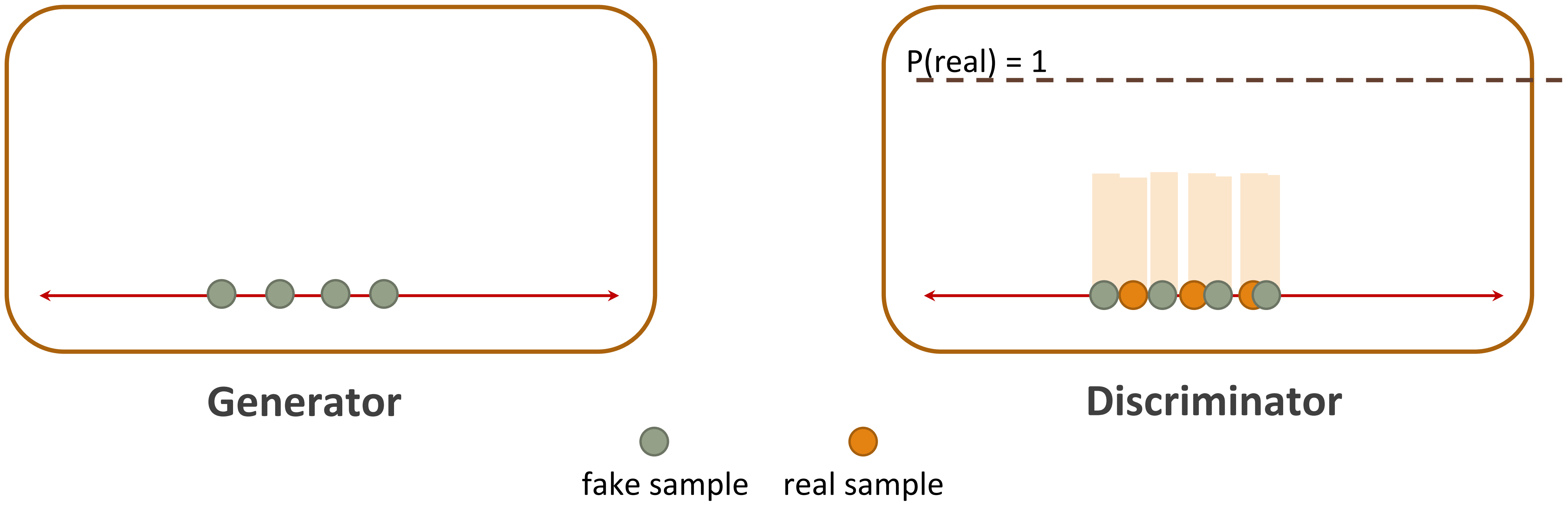
This steps are repeated until the generator is able to fool the discriminator by generating fake data samples that are indistinguishable from the real ones

- 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. 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))**

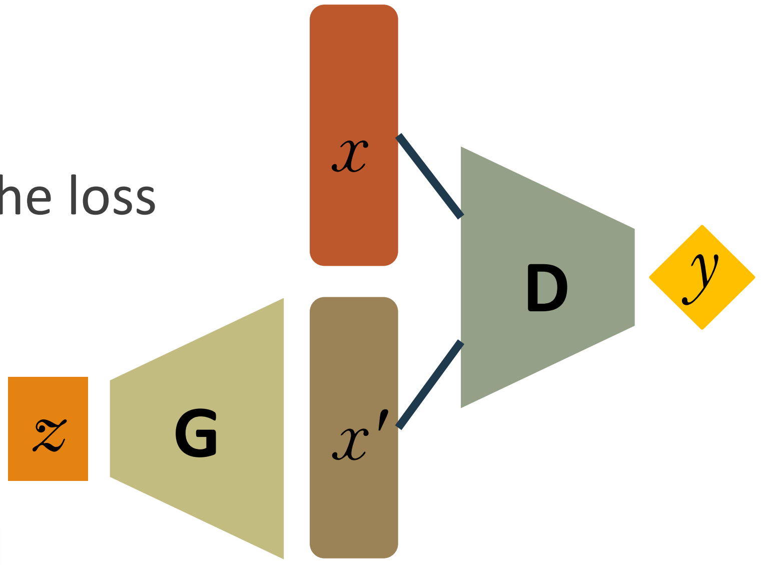
Generator is trained to fool the Discriminator by generating realistic data

- **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

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



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

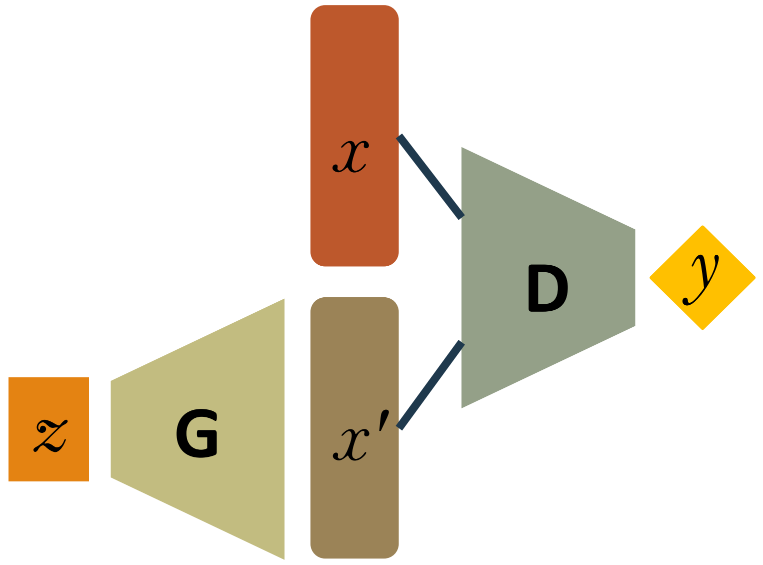
2. Train the generator

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

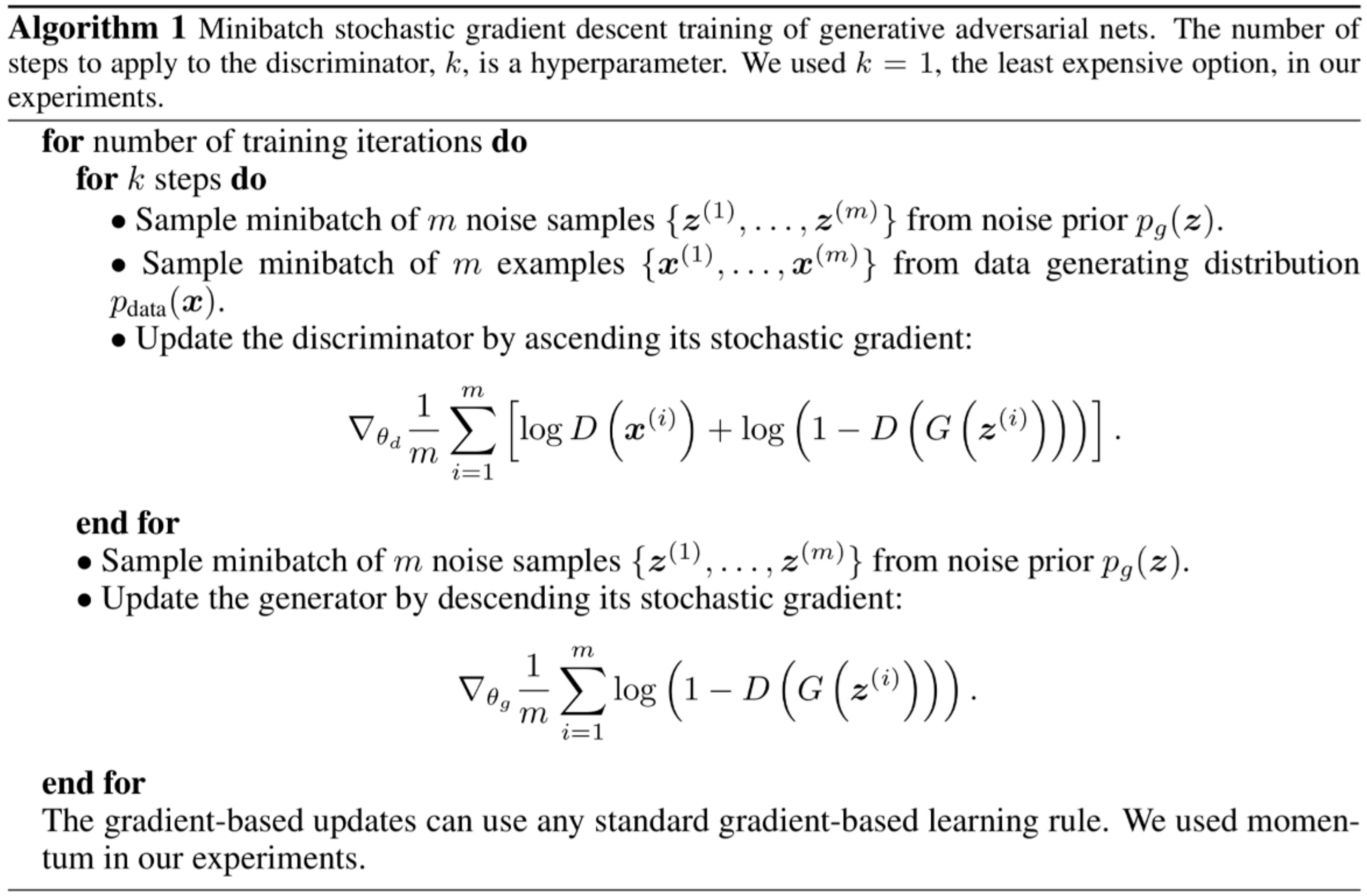
2.2 Get samples (x’) from the generator with input z

2.3 Get loss from the discriminator output with input x’

2.4 Update generator weights according to the losses



GAN Training. General Algorithm



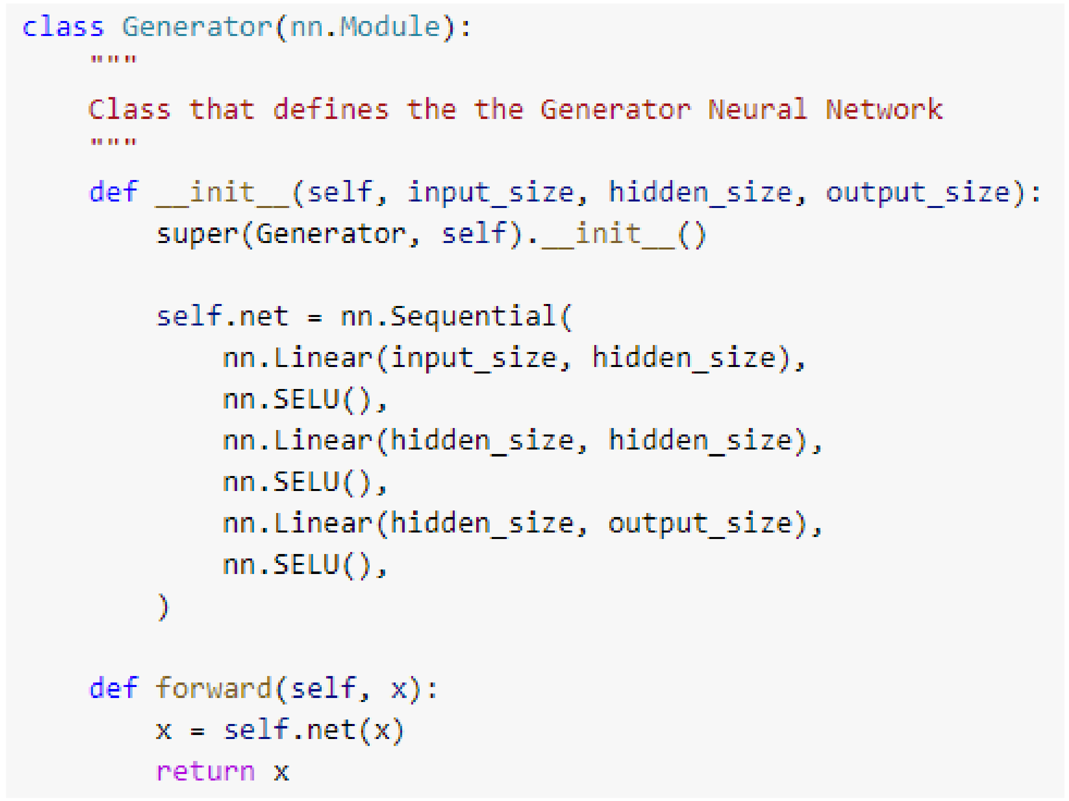
Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

```
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

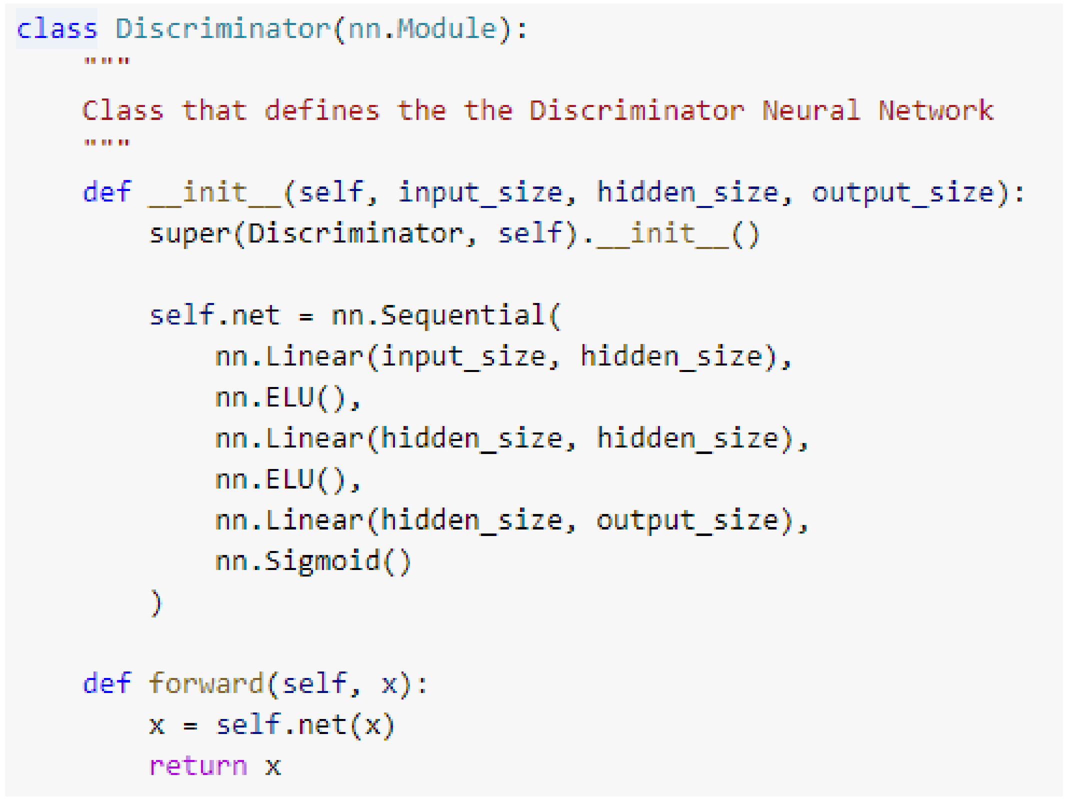
0. Create ANNs



```python
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
```



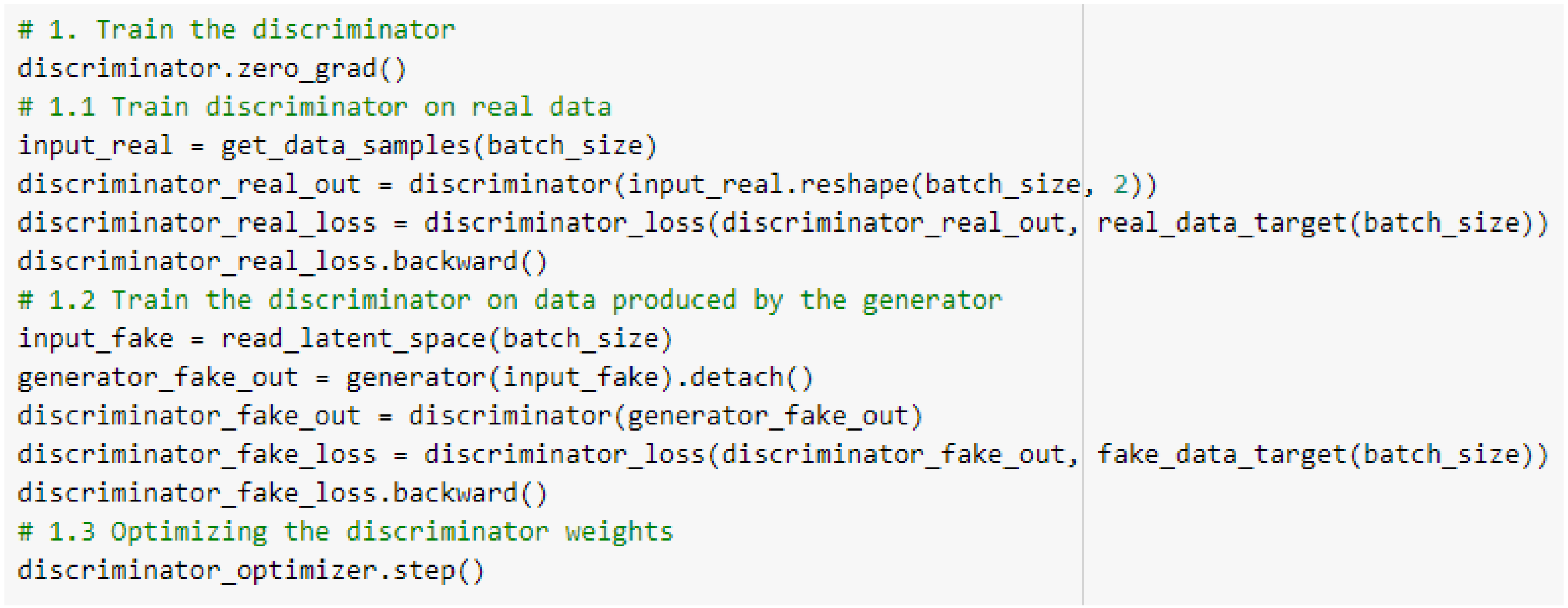
```python
class Discriminator(nn.Module):
    """
    Class that defines the the Discriminator Neural Network
    """
    def __init__(self, input_size, hidden_size, output_size):
        super(Discriminator, self).__init__()

        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

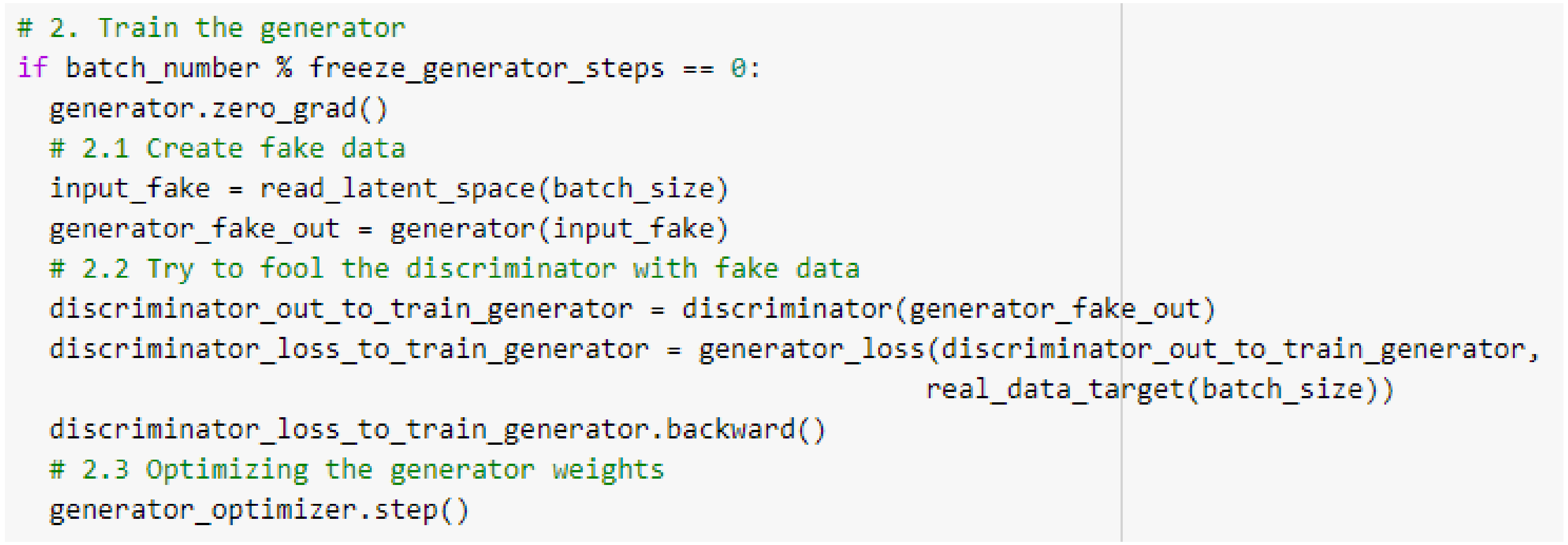
1. Train discriminator



```
# 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

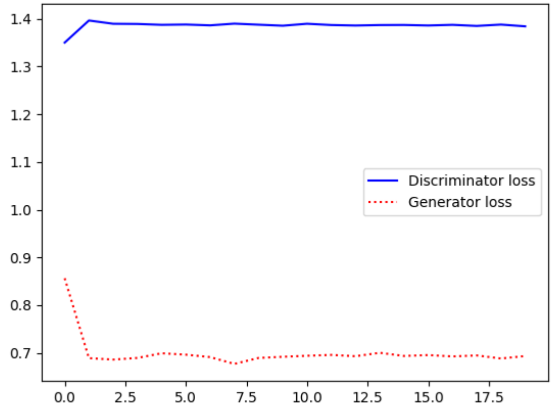


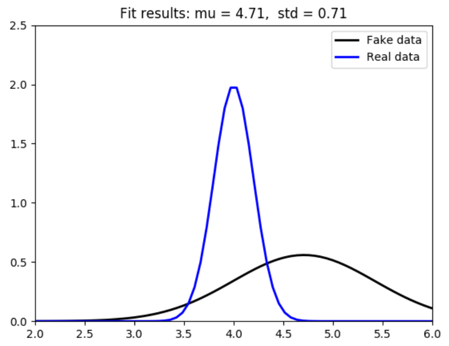
```
# 2. Train the generator
if batch_number % freeze_generator_steps == 0:
  generator.zero_grad()
  # 2.1 Create fake data
  input_fake = read_latent_space(batch_size)
  generator_fake_out = generator(input_fake)
  # 2.2 Try to fool the discriminator with fake data
  discriminator_out_to_train_generator = 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()
```

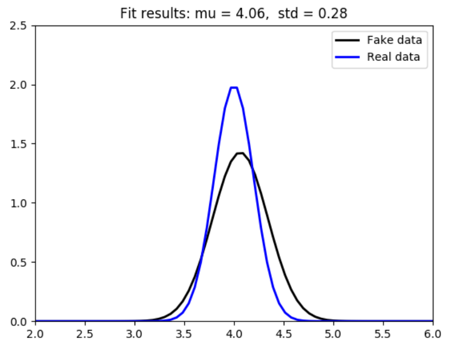
GAN Training. Source Code Example 1

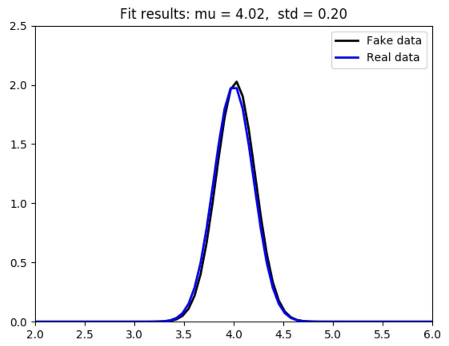
Example: Train a generator to create vectors of a given size that contains float numbers that follow a normal distribution given the mean and the standard deviation

- Real dataset samples: Vectors of real numbers that follow a normal distribution
- Source code: https://drive.google.com/file/d/1E_fbQreOWqlMrs5sqsioyufhlhgUNWNh/view?usp=sharing





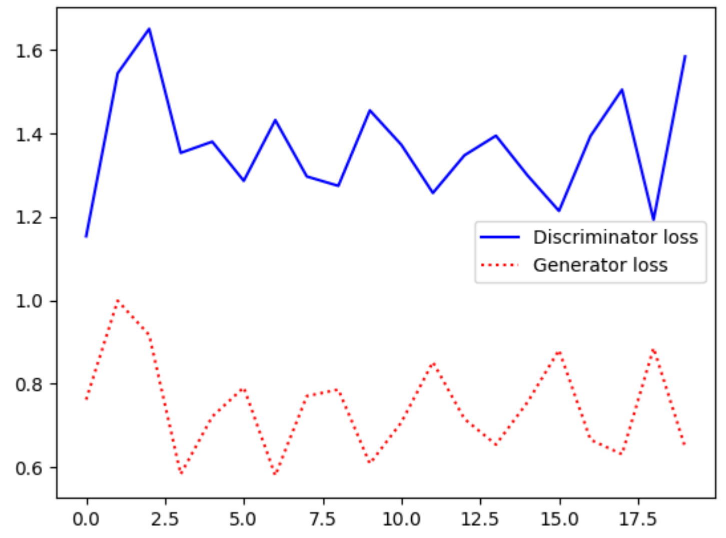


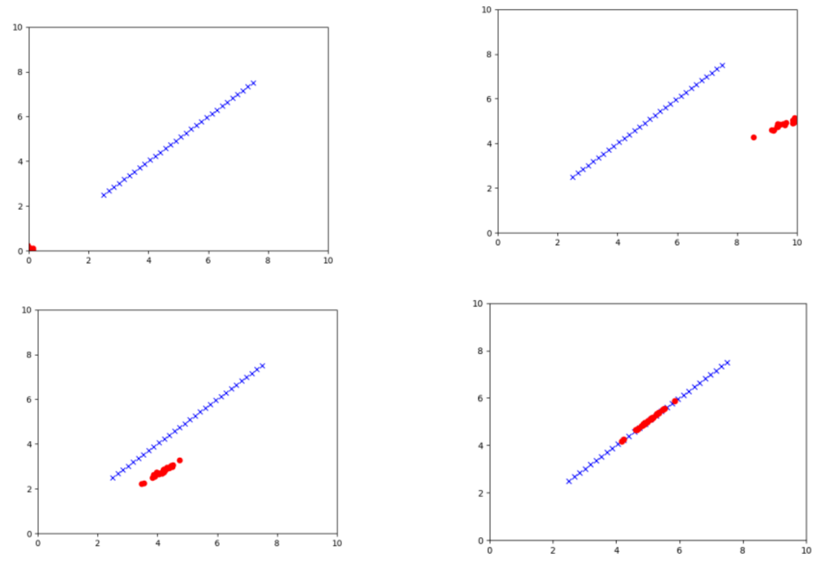


GAN Training. Source Code Example 2

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

- Real dataset samples: Points (x, y) that belong to the line
- Source code:
  https://drive.google.com/file/d/1bfBQ8CZyi9Ht5Nr7rgAPAD9E6Xcg5pQP/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 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)

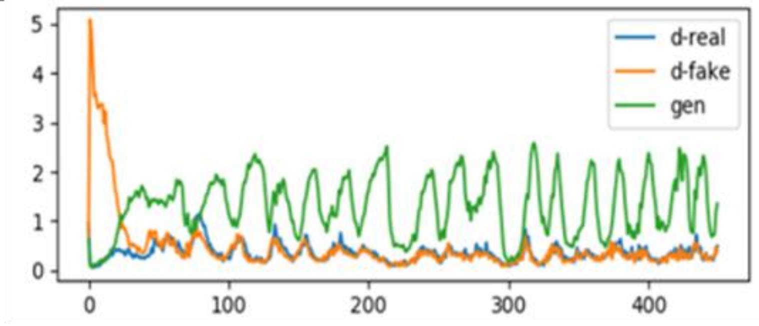
- Real dataset samples: Digits from MNIST dataset
- Source code: https://drive.google.com/file/d/1yhZ1yubqfPAaxJqHdF3D7jTI5hLBooBK/view?usp=sharing

Real data

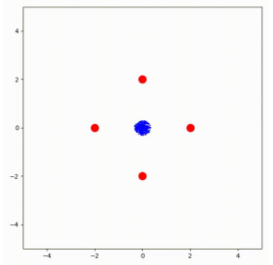
Generated data during training

Not all is good news

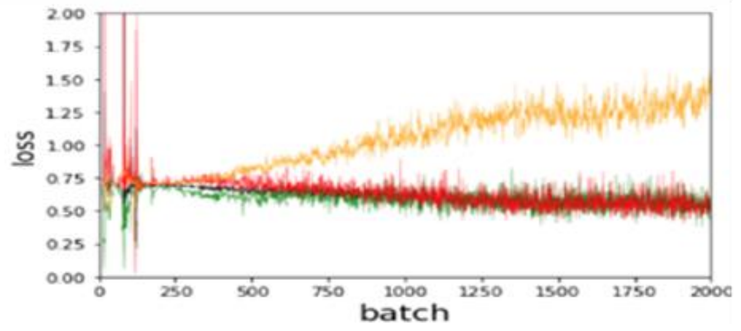
- **Non-convergence:** the model parameters oscillate, destabilize and never converge



- **Mode collapse:** the generator collapses which produces limited varieties of samples



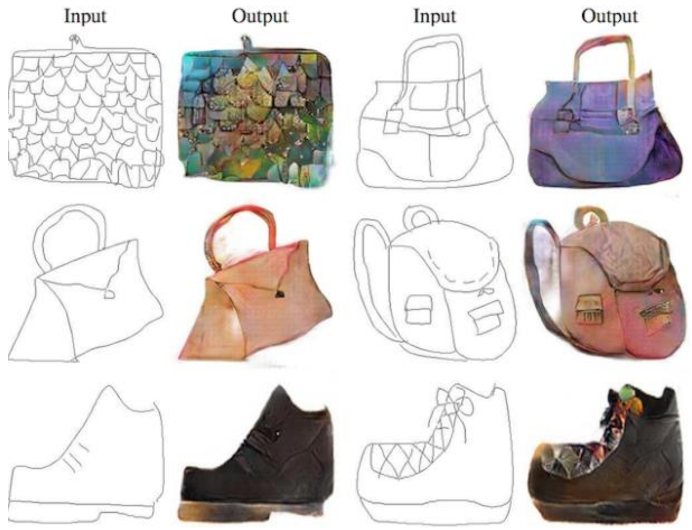
- **Diminished gradient:** the discriminator gets too successful that the generator gradient vanishes and learns nothing

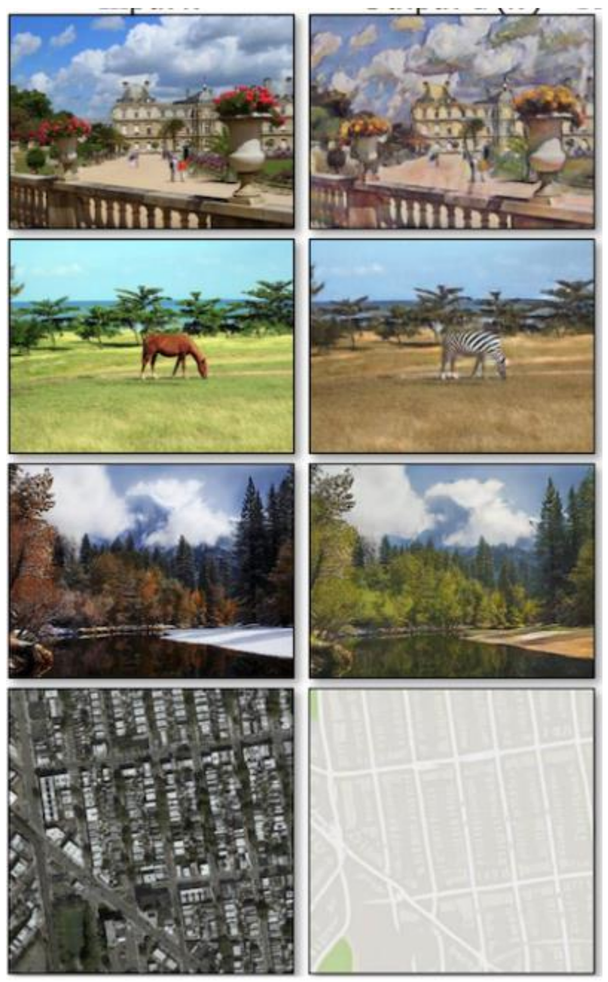


Some GAN Applications

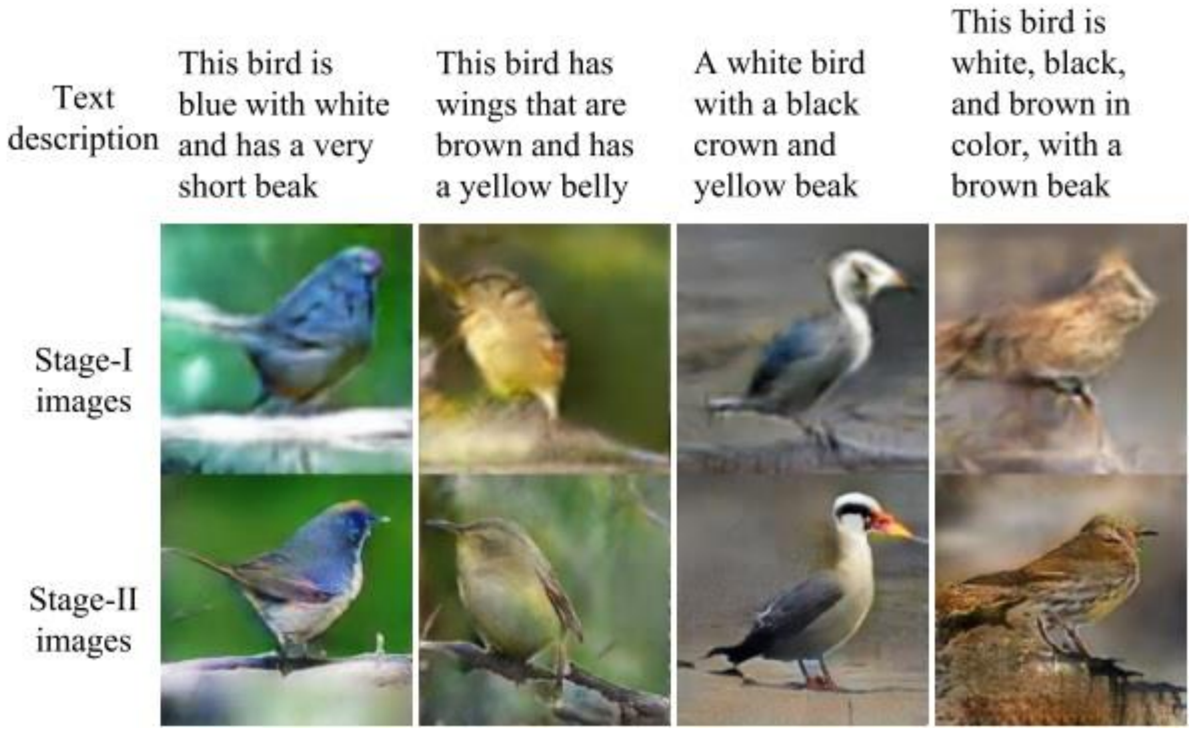
Generate New Samples of Image Datasets

Image-to-Image Translation

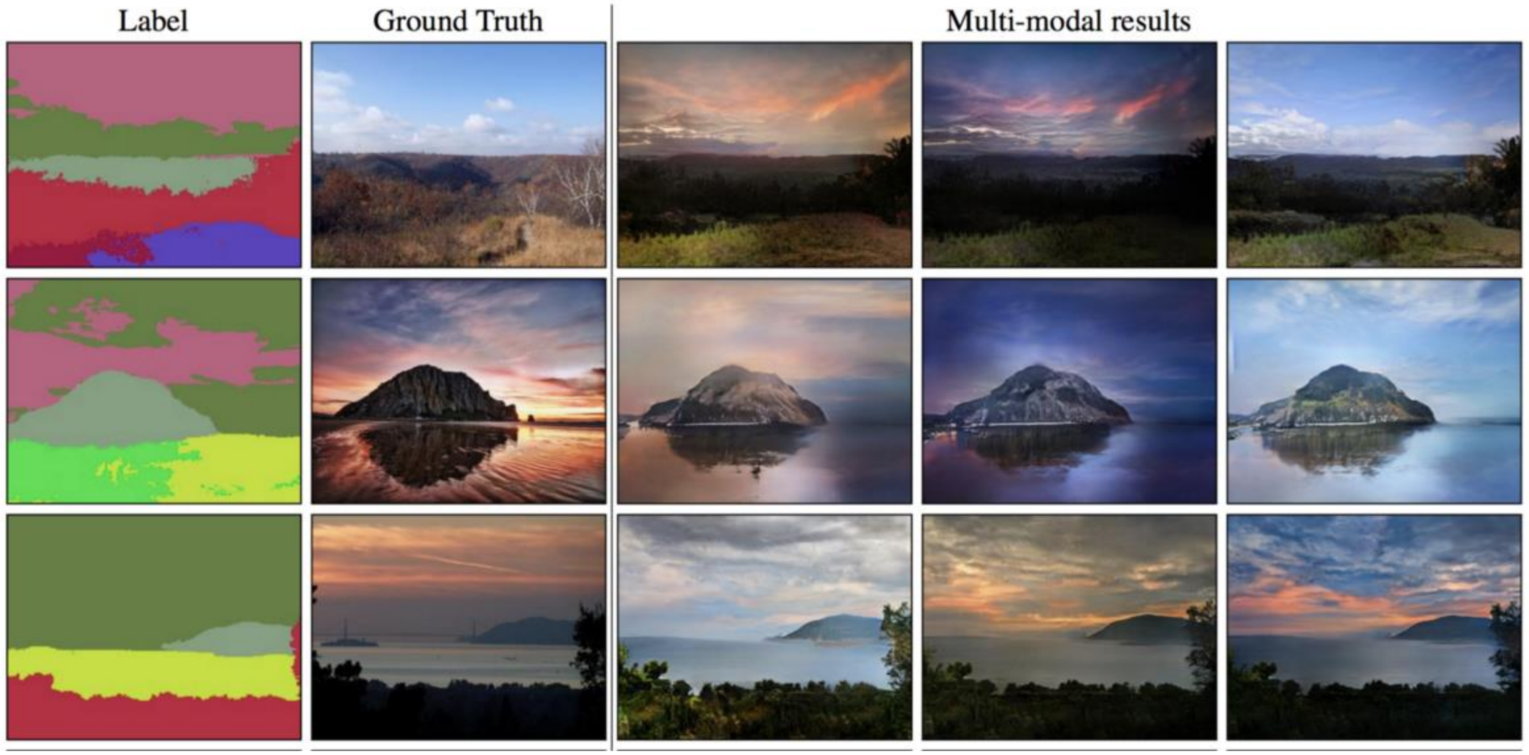




Text-to-Image Translation


Text description
This bird is blue with white and has a very short beak
This bird has wings that are brown and has a yellow belly
A white bird with a black crown and yellow beak
This bird is white, black, and brown in color, with a brown beak
Stage-I images
Stage-II images

Semantic-Image-to-Photo Translation



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

Deepfake Videos

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

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