

# Deep generative neural networks

## Fundamentals & problem solving

---

JAMAL TOUTOUH

[jamal@uma.es](mailto:jamal@uma.es)

jamal.es

@jamtou

Sergio Nesmachnow

[sergion@fing.edu.uy](mailto:sergion@fing.edu.uy)

# Jamal Toutouh, Ph.D.

---

Researcher Assistant Professor at the **University of Málaga** (Spain)

Affiliate Researcher at **Massachusetts Institute of Technology (MIT)**

- MIT Computer Science & Artificial Intelligence Lab
- PhD in Computer Science, **University of Malaga**
- M.Sc. in Software Engineering and Artificial Intelligence, **University of Malaga**
- M.Sc. in Information and Computer Sciences **University of Luxembourg**



[jamal@uma.es](mailto:jamal@uma.es)

[www.jamal.es](http://www.jamal.es)

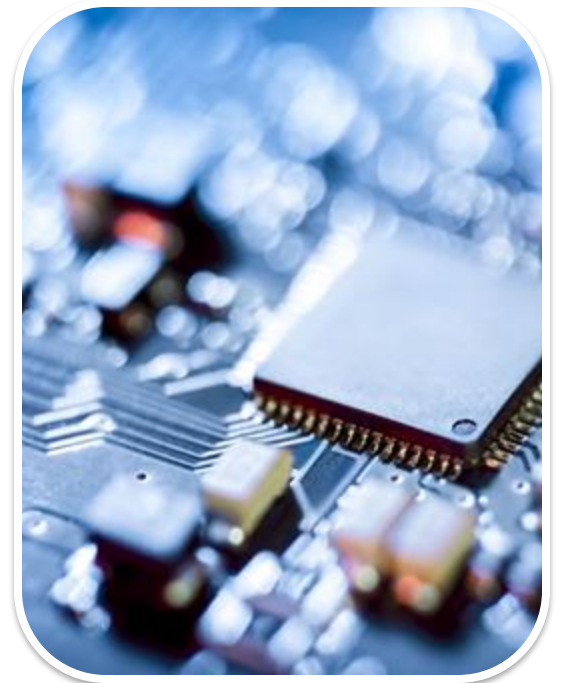
@jamtou

# Intended Learning Outcomes

---

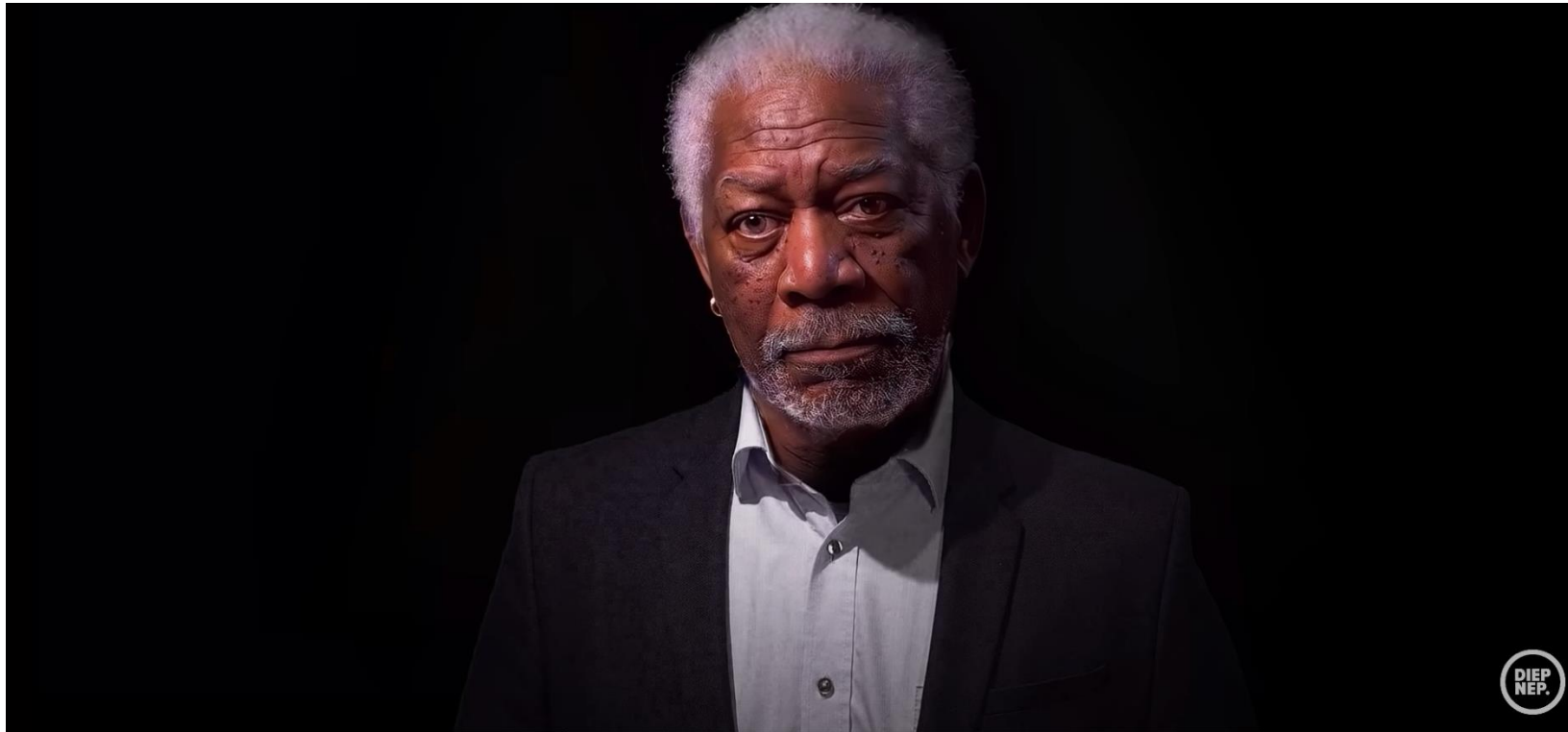
Attendees will, at the end of the course, be able to:

- describe the main principles of Artificial Neural Networks and Generative Adversarial Networks and their design
- identify problems that can be addressed by using Artificial Neural Networks
- use Python code to create and use Artificial Neural Networks to address classification and prediction problems
- identify problems that can be solved using Generative Machine Learning
- use Python code to create and use Generative Adversarial Networks to generate synthetic data



# Generative Adversarial Networks for fun

---



<https://youtu.be/oxXpB9pSETo?si=c1cE2ocExaJzK4q>

# Generative Adversarial Networks for fun

---



<https://thispersondoesnotexist.com/>

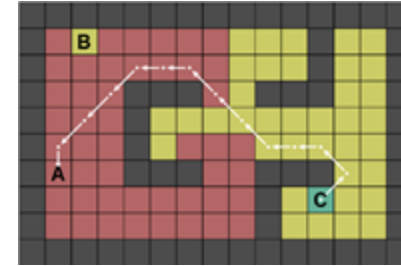
---

# Main Concepts

# Artificial Intelligence

---

- **Artificial Intelligence:** intelligence exhibited by machines and software
- Goal: automate “intellectual tasks” performed by humans
- AI models can be simple or complex
  - Simple: search, pathfinding, simple game playing, etc.
  - Complex: computer vision, control tasks, speech recognition, etc.



# Artificial intelligence

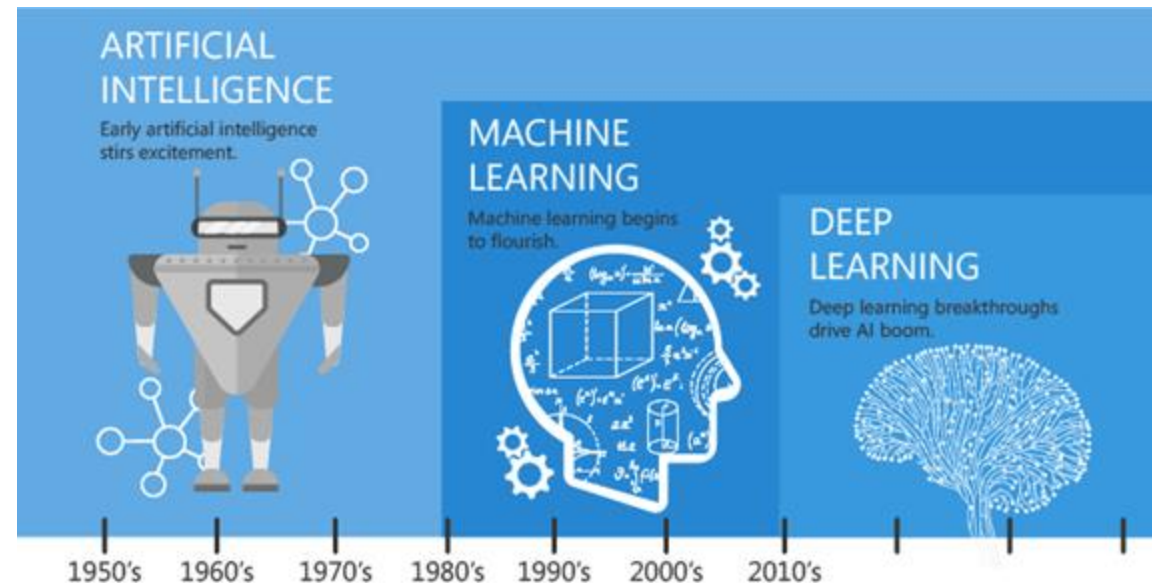
---

- The main idea is that AI imitates the human cognition process (perception, learning, pattern recognition, etc.)
- Key aspects: reasoning, problem solving, learning, knowledge representation
- Many types of algorithms: search, optimization, logic programming, or **machine learning algorithms**.



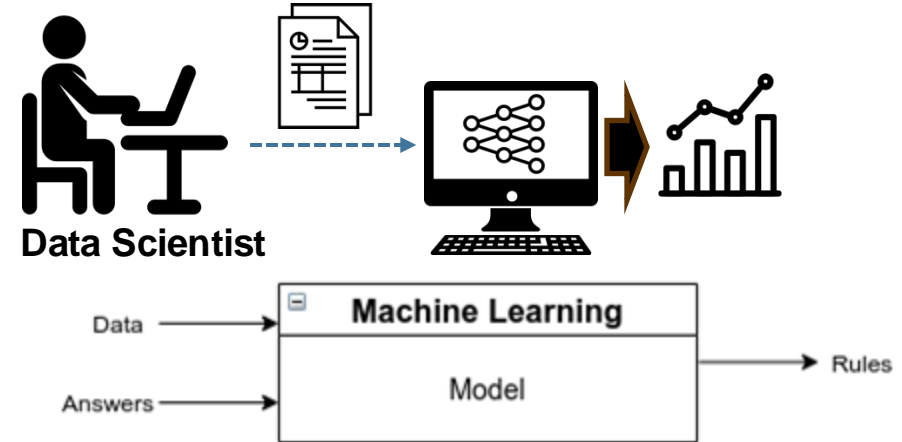
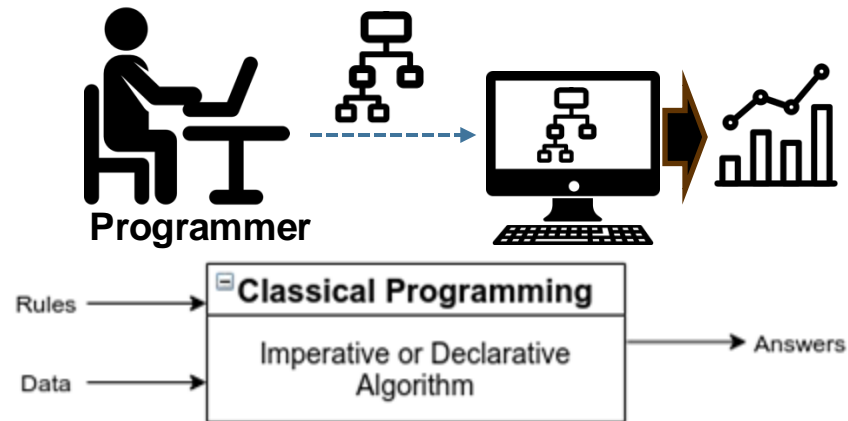
# Machine Learning, Deep Learning, AI ....

- **Artificial Intelligence** is human-like “intelligence” exhibited by computers
- **Machine Learning** is the field of study that gives the computers the ability to learn without being explicitly programmed
- **Deep Learning** uses deep neural networks to implement machine learning



# Machine Learning

- Instead of being explicitly programmed (i.e. with a set of rules), machine learning algorithms try to infer the rules using a model.

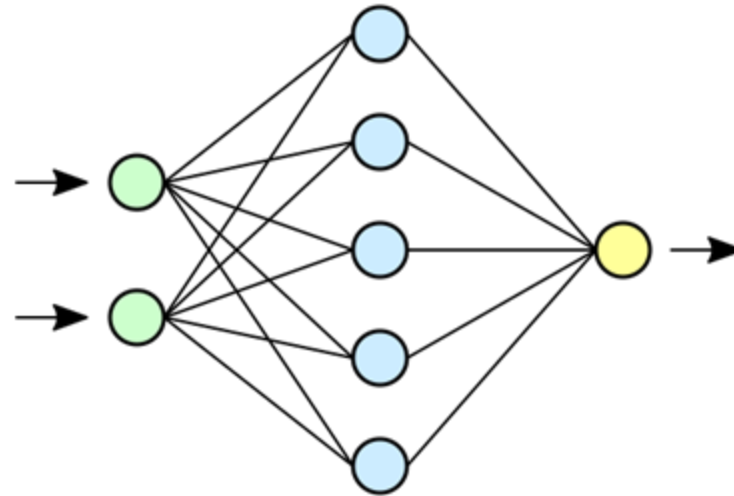


Probabilistic (i.e., not deterministic) outputs.  
Characterized by an accuracy rate.

# Neural networks

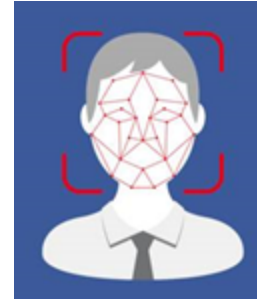
---

- A type of machine learning algorithms specialized on handling layered representations of data.
- Multi stage information extraction process: allows modeling complex (non-linear) functions.



# Artificial Intelligence Applications

- Facial recognition
- Game playing
- Speech recognition
- Language translation
- Self-driving cars
- Image translation: edges to photo
- Fake images
- Fake videos



---

# Artificial Neural Networks

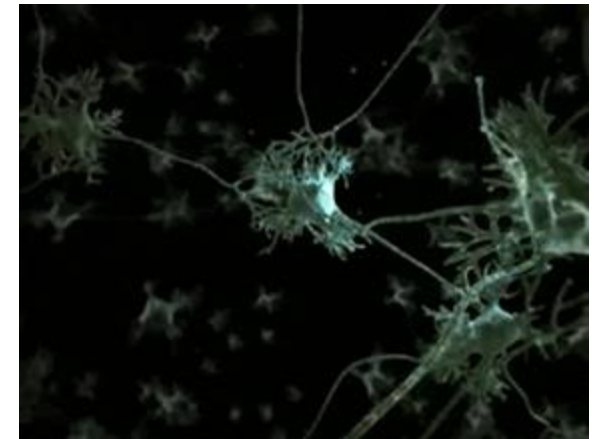
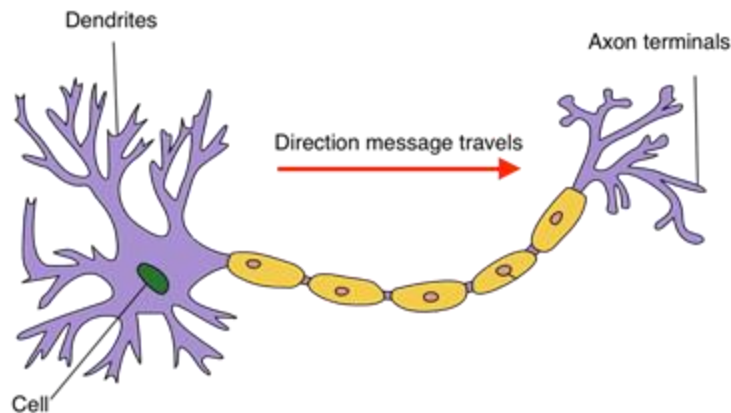
# Simple Biological Neuron

---

The neuron is the fundamental cell responsible for **processing and transmitting information** throughout the nervous system.

A simple biological network has three major parts:

- **Dendrites:** They branch out into a tree around the cell body. They get incoming signals to cell body with their strength as weights.
- **Cell :** Collects input through dendrites and processes to produce output.
- **Axon:** Responsible for transmitting signals to other neurons.



# Collective Intelligence

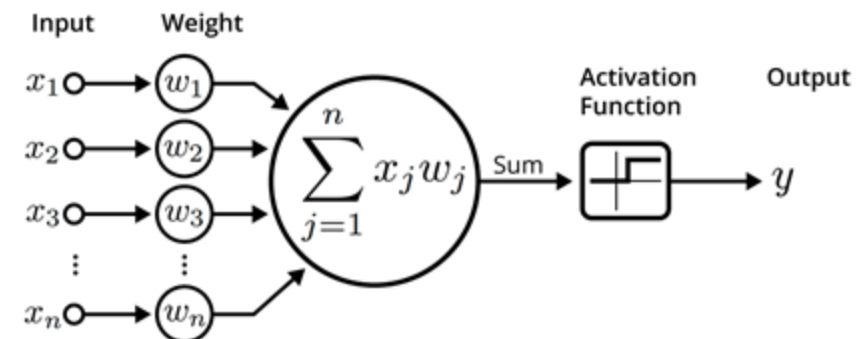
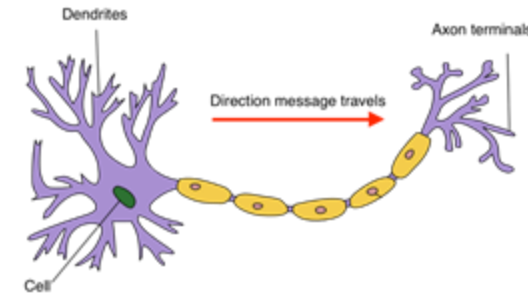
---

Shared or group intelligence that emerges from collaboration, collective efforts, and/or competition of many agents.

- **A single neuron has limited processing capabilities:** response speed is about several milliseconds.
- **However, the human brain is very powerful for problem solving:** it uses the aggregated power of millions of neurons.

# Artificial Neuron

- An Artificial Neuron is a computational model of a biological neuron.
- The idea is that the artificial neuron receives input signals from other connected artificial neurons and via a **non-linear transmission function** emits a signal itself.
- Main operation:
  - Receives *n inputs*
  - Computes the **weighted sum**
  - Passes through an **activation function**
  - Sends the signal to succeeding neurons





# Artificial Neuron. Basic example

Two-inputs neuron operation:

1. Each input is multiplied by a **weight**

$$x_1 \rightarrow x_1 * w_1$$

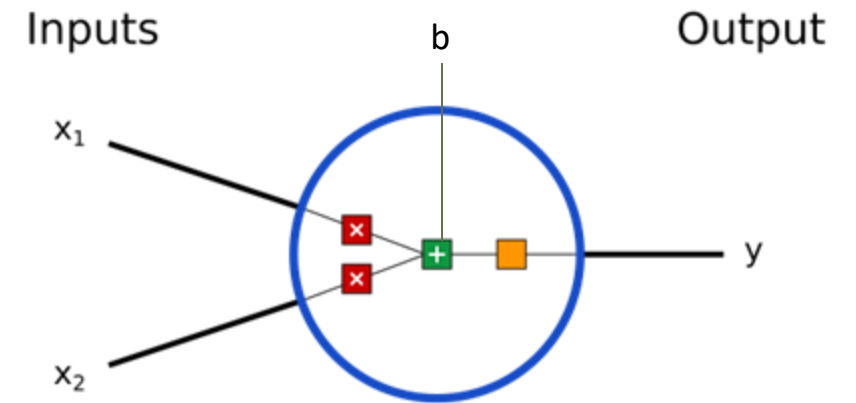
$$x_2 \rightarrow x_2 * w_2$$

1. All weighted sums are added with a **bias**  $b$  (*feedforward*)

$$(x_1 * w_1) + (x_2 * w_2) + b$$

1. The sum is passed through an **activation function**

$$y = f(x_1 * w_1 + x_2 * w_2 + b)$$



[https://colab.research.google.com/drive/1wQWlUvqPBDaYBQJuB5b\\_ss0GPVc-pglU?usp=sharing](https://colab.research.google.com/drive/1wQWlUvqPBDaYBQJuB5b_ss0GPVc-pglU?usp=sharing)

The output of the network depends on the **weights**, the **bias**, and the **activation function**

# Artificial Neuron. Basic example

Two-inputs neuron operation:

1. Each input is multiplied by a **weight**

$$x_1 \rightarrow x_1 * w_1$$

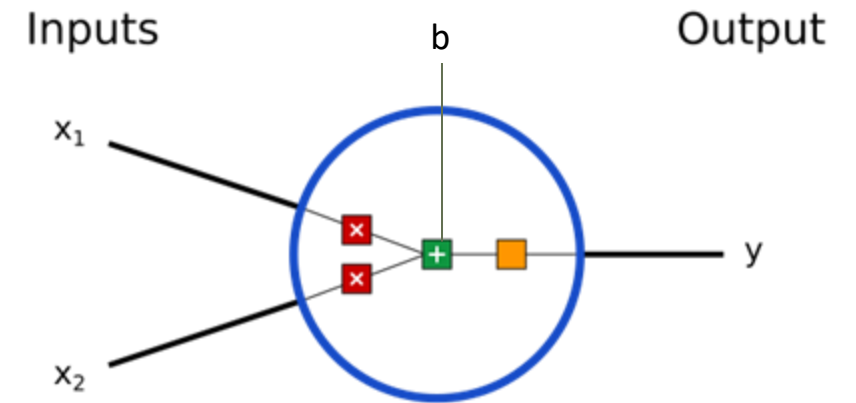
$$x_2 \rightarrow x_2 * w_2$$

1. All weighted sums are added with a **bias** *b* (**feedforward**)

$$(x_1 * w_1) + (x_2 * w_2) + b$$

1. The sum is passed through an **activation function**

$$y = f(x_1 * w_1 + x_2 * w_2 + b)$$

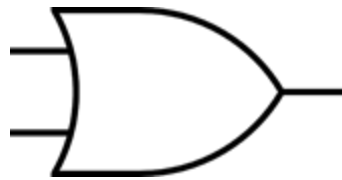


[https://colab.research.google.com/drive/1wQWlUvqPBDAyBQJuB5b\\_ss0GPVc-pglU?usp=sharing](https://colab.research.google.com/drive/1wQWlUvqPBDAyBQJuB5b_ss0GPVc-pglU?usp=sharing)

The output of the network depends on the **weights**, the **bias**, and the **activation function**

# Artificial Neuron. What can we do?

- Try to implement a logic function with the two-input neuron.



$x_1$	$x_2$	$x_1$ OR $x_2$
0	0	0
0	1	1
1	0	1
1	1	1



$x_1$	$x_2$	$x_1$ AND $x_2$
0	0	0
0	1	0
1	0	0
1	1	1

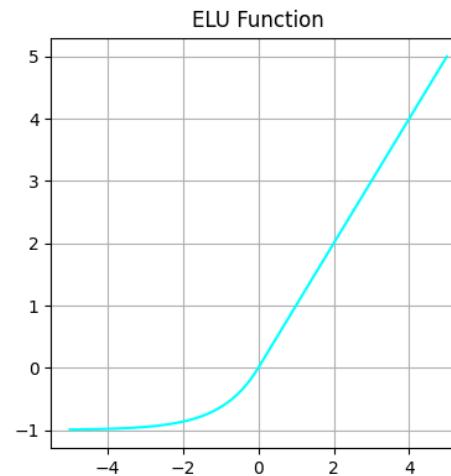
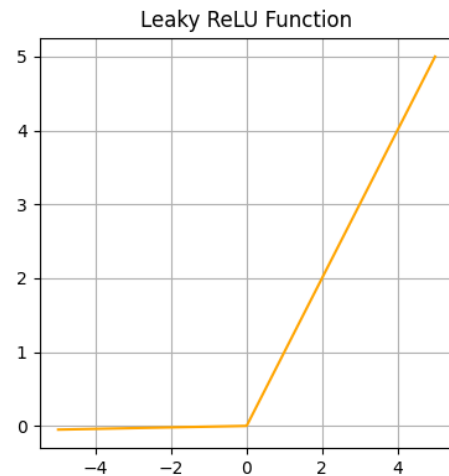
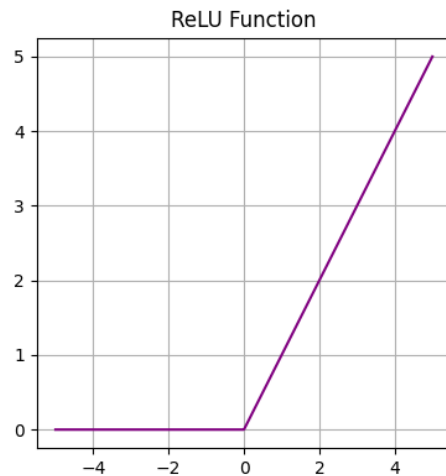
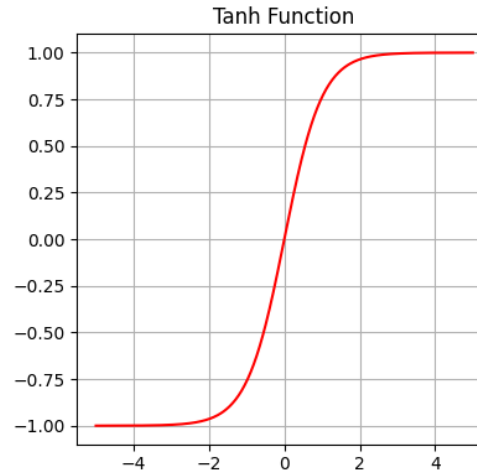
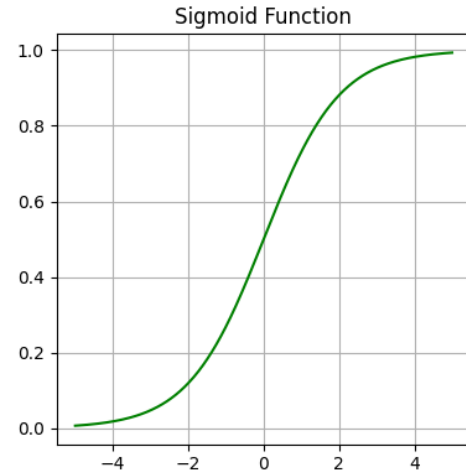
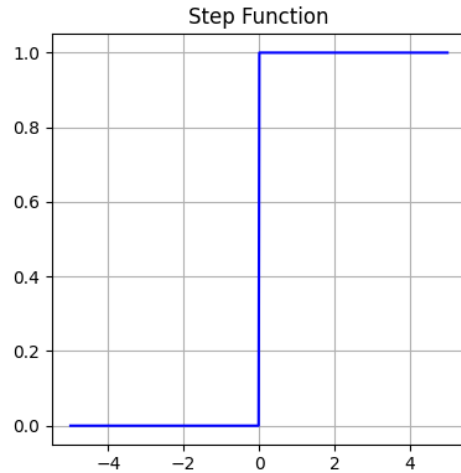
[https://colab.research.google.com/drive/1wQWlUvqPBdaYBQJuB5b\\_ss0GPVc-pglU?usp=sharing](https://colab.research.google.com/drive/1wQWlUvqPBdaYBQJuB5b_ss0GPVc-pglU?usp=sharing)

# Activation Function

---

- Activation function decides whether a neuron should be **activated or not** by calculating the weighted sum and further adding bias to it. The motive is to **introduce non-linearity** into the output of a neuron.
- If we do not apply activation function then the output signal would be **simply linear function** (one-degree polynomial).
- Linear functions are limited in their complexity, have **less power**. Without activation function, our model cannot learn and model complicated data such as images, videos, audio, speech, etc.

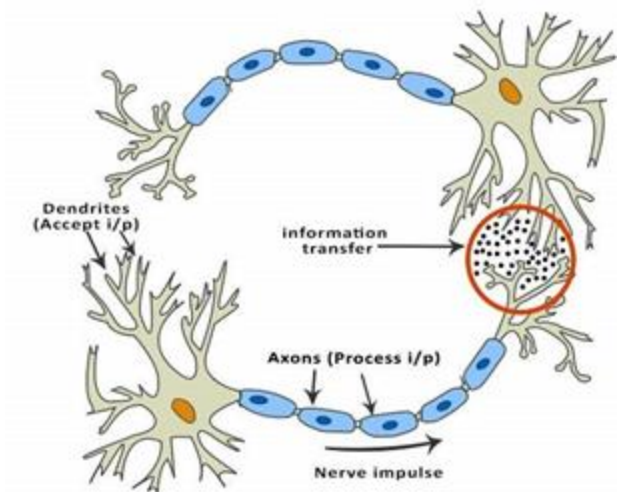
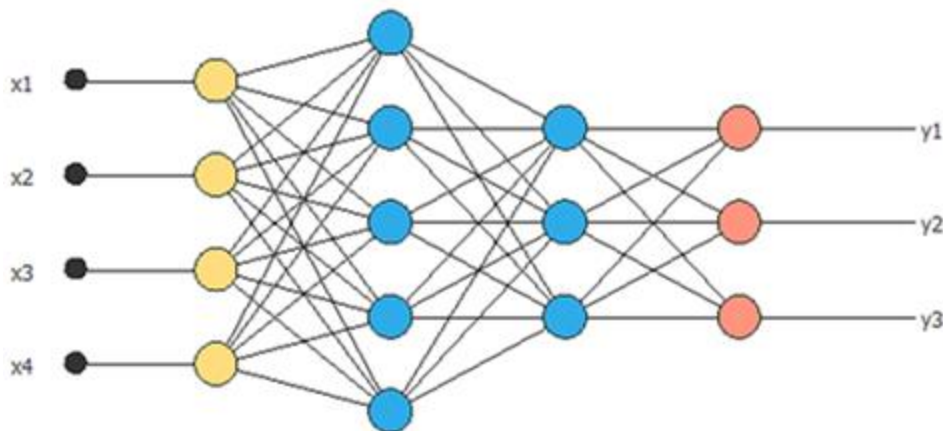
# Activation Function. Types



[https://colab.research.google.com/drive/1\\_HBmqEihSNQirwOX7Gy6455ySYDneZc2?usp=sharing](https://colab.research.google.com/drive/1_HBmqEihSNQirwOX7Gy6455ySYDneZc2?usp=sharing)

# Artificial Neural Networks

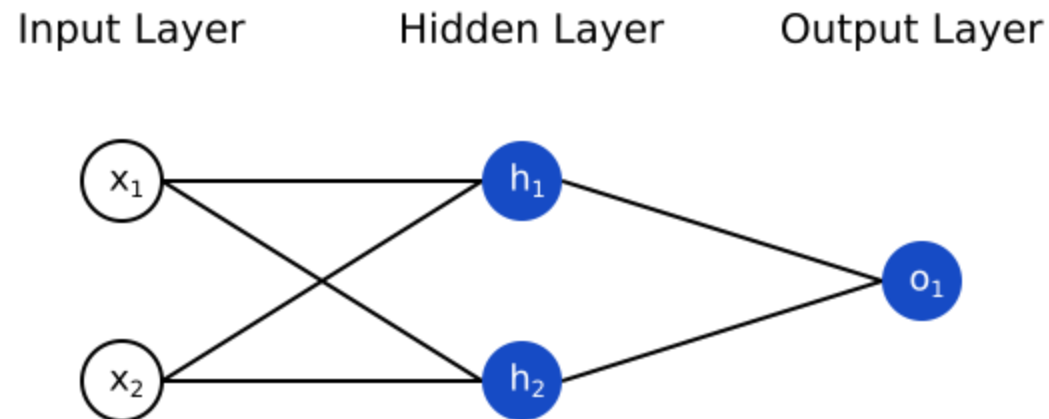
- A neural network is a bunch of neurons connected together.
- Neural networks are typically **organized in layers**.
- Layers are made up of a number of **interconnected neurons**.
- Inputs are presented to the network via the **input layer**, which communicates to **one or more hidden layers** through **weighted connections**.
- The hidden layers then link to an **output layer**.



# Artificial Neural Networks. Code

---

- **Example 1:** Two inputs, two neurons in a hidden layer, and one output



**Code:** *basic-two-layer-neural-network.py*

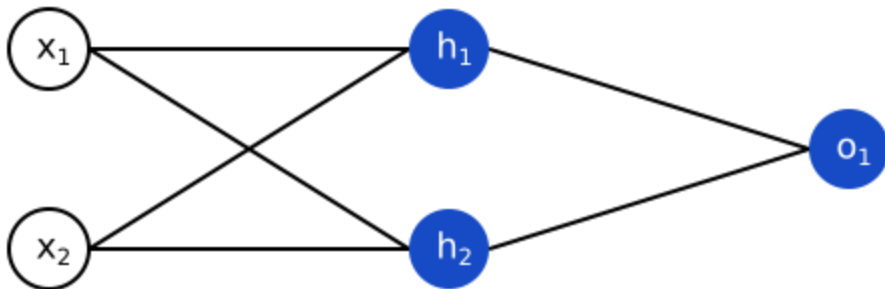
- **Example 2:** X inputs, H neurons in the hidden layer, and one output

**Code:** *two-layer-neural-network.py*

# Artificial Neural Networks. Code

- **Example 1:** Two inputs, two neurons in a hidden layer, and one output

Input Layer                  Hidden Layer                  Output Layer



```
class NeuralNetwork:
    def __init__(self):
        """
        Initializes a neural network to model XOR logic:
        - Input layer: 2 neurons (inputs directly used).
        - Hidden layer: 2 neurons.
        - Output layer: 1 neuron.
        """
        # Hidden layer: 2 neurons
        self.hidden_neurons = [
            Neuron(weights=[1.0, 1.0], bias=-0.5, activation='step'), # Acts as OR gate
            Neuron(weights=[-1.0, -1.0], bias=1.5, activation='step') # Acts as NAND gate
        ]
        # Output layer: 1 neuron
        self.output_neuron = Neuron(weights=[1.0, 1.0], bias=-1.5, activation='step') # Ac
```

<https://colab.research.google.com/drive/1SlgAgzjxcKD-J8OzkXRr--e0ljtNldrT?usp=sharing>

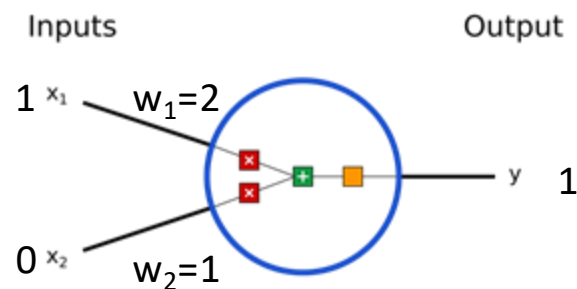
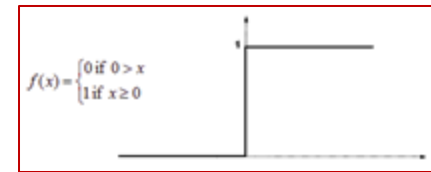


# How do ANNs Learn?

- The **output** of the ANN depends on the **weights**
- **Learning** consist in **updating the weights** to get a desired output, i.e., **minimize the error**



$$y = f(x_1 * w_1 + x_2 * w_2 + b)$$

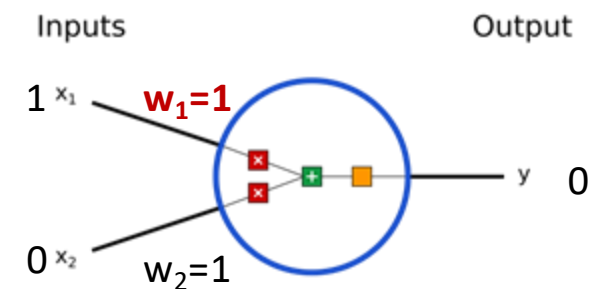


$$y = \text{step}(1 \times 2 + 0 \times 1, 2)$$
$$y = \text{step}(2, 2) = 1$$

Compute error

$$\text{Error} = \text{MSE} = 1$$

Update weights



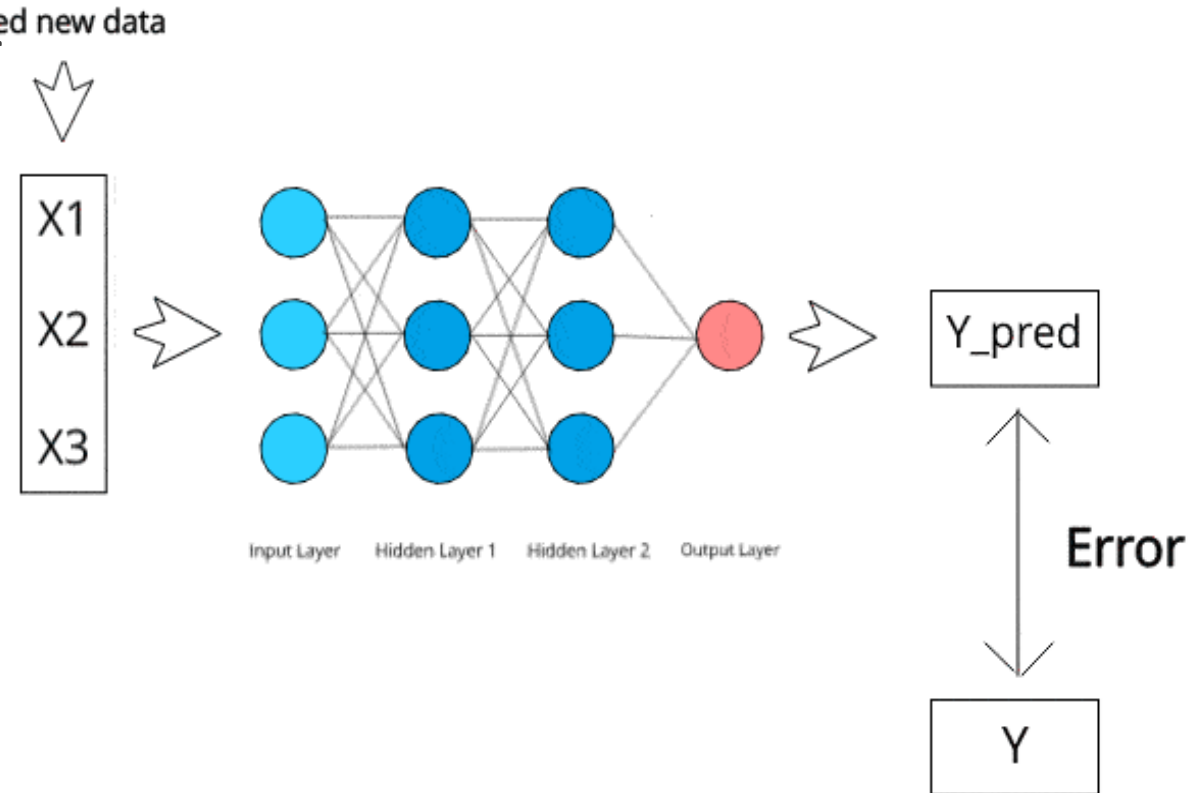
$$y = \text{step}(1 \times 1 + 0 \times 1, 2)$$
$$y = \text{step}(1, 2) = 0$$

# 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.



# How do ANNs Learn? A simple example

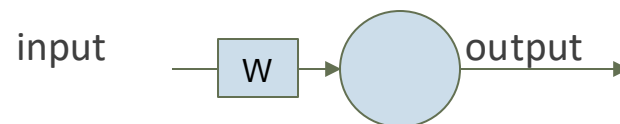
- **Objective:** Learn the following data

Input	Desired output
0	0
1	2
2	4
3	6
4	8

$$\text{output} = 2 \times \text{input}$$

- **Error function:** mean squared error (MSE)

- **Neural network model:**  $\text{output} = W \times \text{input}$  ( $W$  represents the weight)



# How do ANNs Learn? A simple example

---

**First step:** Random weights initialization  $\rightarrow W = 3 \rightarrow \text{output} = 3 \times \text{input}$

**Second step:** Get actual output  $\rightarrow$  Forward propagate input

Input	Actual output
0	0
1	3
2	6
3	9
4	12

# How do ANNs Learn? A simple example

---

**Third step:** Get loss values  $\rightarrow$  loss = f(actual output, desired output)  
In our example f is mean squared error (MSE)

Input	Actual output	Desired output	Loss = Square error
0	0	0	0
1	3	2	1
2	6	4	4
3	9	6	9
4	12	8	16

Total loss is **30**

# How do ANNs Learn? A simple example

---

## Fourth step: Differentiation

In our numerical example:  $-1000.0 < W < 1000.0$

We can move with steps of 0.0001

Optimization problem  $\rightarrow$  finding  $W$  that minimizes loss

**Differentiation** allows us to address the problem

Remember  $\rightarrow$  the derivative of a function at a certain point, gives the rate or the speed of which this function is changing its values at this point

In order to see the effect of the derivative, we can ask ourselves the following question: how much the total error will change if we change the internal weight of the neural network with a certain small value  $\delta W = 0.0001$

**Loss with  $W = 3.0001 \rightarrow 30.006$**

**Loss with  $W = 2.9999 \rightarrow 29.994$**

# How do ANNs Learn? A simple example

## Fourth step: Differentiation

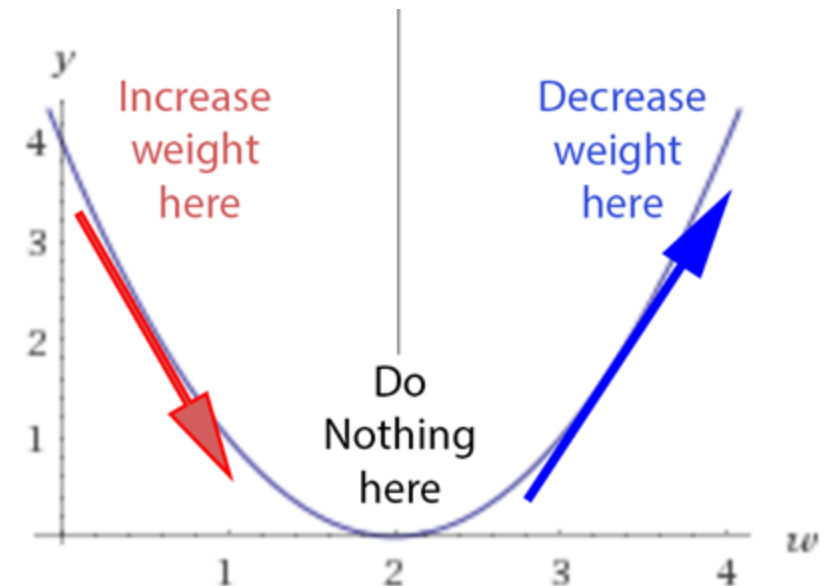
We could guess this rate by calculating directly the derivative of the loss function

Here is what our loss function looks like:

If  $W=2$ , we have a **loss of 0**, since the neural network actual output fits perfectly the training set

If  $W<2$ , we have a **positive loss** function, but the **derivative is negative**, meaning that an increase of weight will decrease the loss function

If  $W>2$ , we have a **positive loss**, but the **derivative is positive**, meaning that any more increase in the weight, will increase the losses even more



Arrows represent the derivative at the corresponding points

# How do ANNs Learn? A simple example

---

## **Fifth step:** Backpropagation

In this example, we used only one layer neural network  
→ No backpropagation is needed!!!

In the case there are more layers, the process is the same but each layer (as the output layer with the loss function) requires to provide the function of its derivative.

Thus, we only need to keep a stack of the function calls during the forward pass and their parameters, in order to know the way back to back-propagate the errors using the derivatives of these functions. This can be done by **de-stacking** through the function calls. This technique is called **auto-differentiation**.



# How do ANNs Learn? A simple example

---

## Sixth step: Weight update

Thus as a general rule of weight updates is the **delta rule**:

New weight = old weight - derivative x learning rate

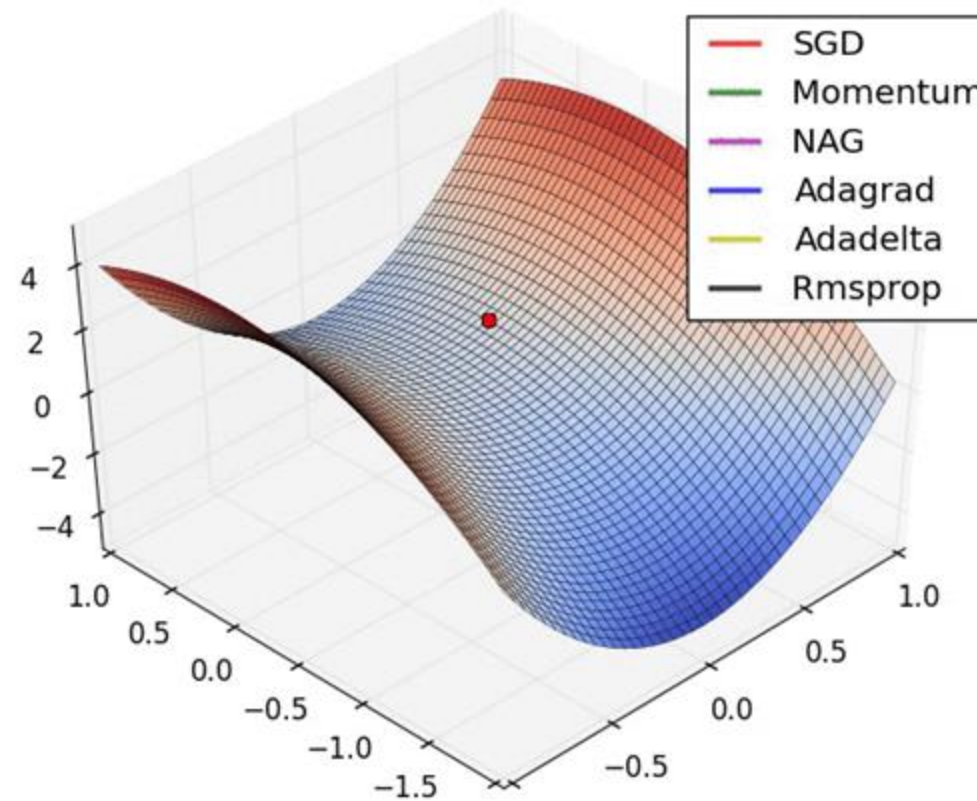
If the **derivative rate is positive**, it means that an increase in weight will increase the error, thus the new weight should be smaller.

If the **derivative rate is negative**, it means that an increase in weight will decrease the error, thus we need to increase the weights.

If the **derivative is 0**, it means that we are in a stable minimum. Thus, no update on the weights is needed -> we reached a stable state.

# How do ANNs Learn? A simple example

The importance of the optimization method applied



<https://towardsdatascience.com/overview-of-various-optimizers-in-neural-networks-17c1be2df6d5>

# Deep Learning

---

ANN results get better with

- more/better **data**
- bigger **models**
- more **computation**

# Deep Learning

ANN results get better with

- more/better **data**
- bigger **models**
- more **computation**

## Example:

Deep Learning for regression

[https://colab.research.google.com/drive/1HIU\\_2w2ELvC8tzPkJXjtwqdoVJWIQghO](https://colab.research.google.com/drive/1HIU_2w2ELvC8tzPkJXjtwqdoVJWIQghO)



# Artificial Neural Networks

---

- **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. Types

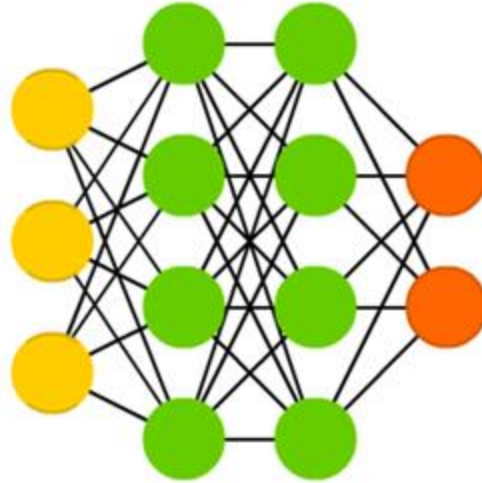
Perceptron (P)



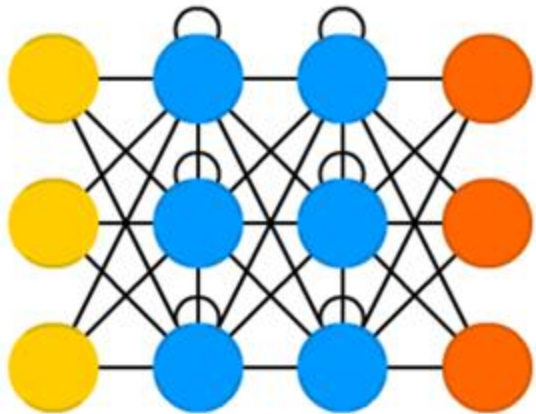
Feed Forward (FF)



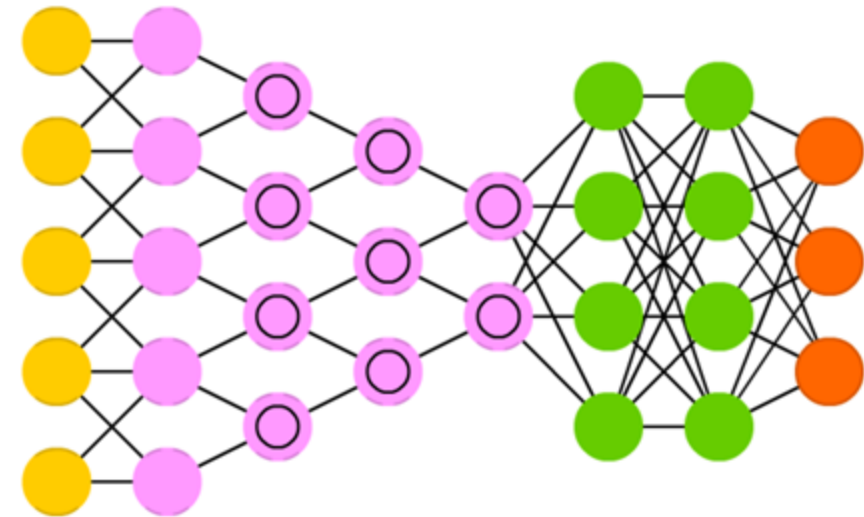
Deep Feed Forward (DFF)



Recurrent Neural Network (RNN)

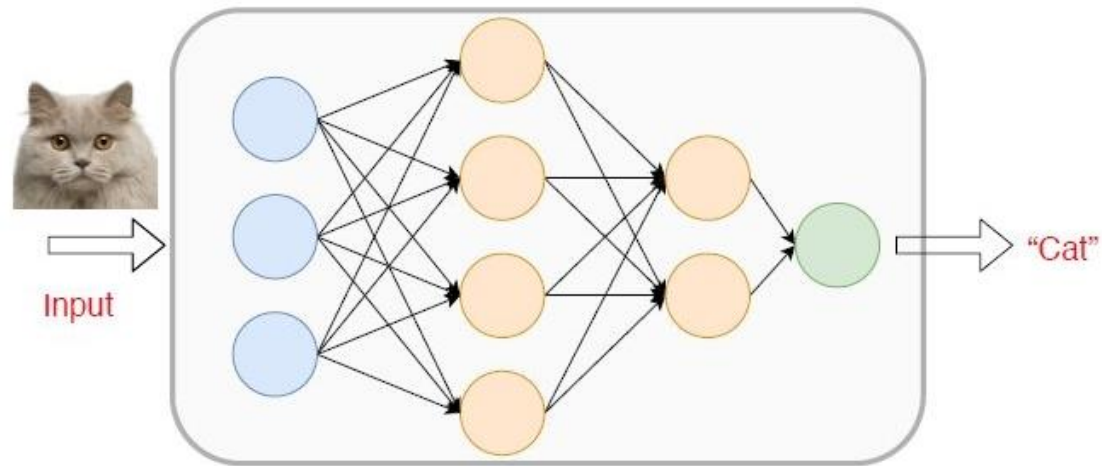


Deep Convolutional Network (DCN)



# Artificial Neural Networks. MLPs

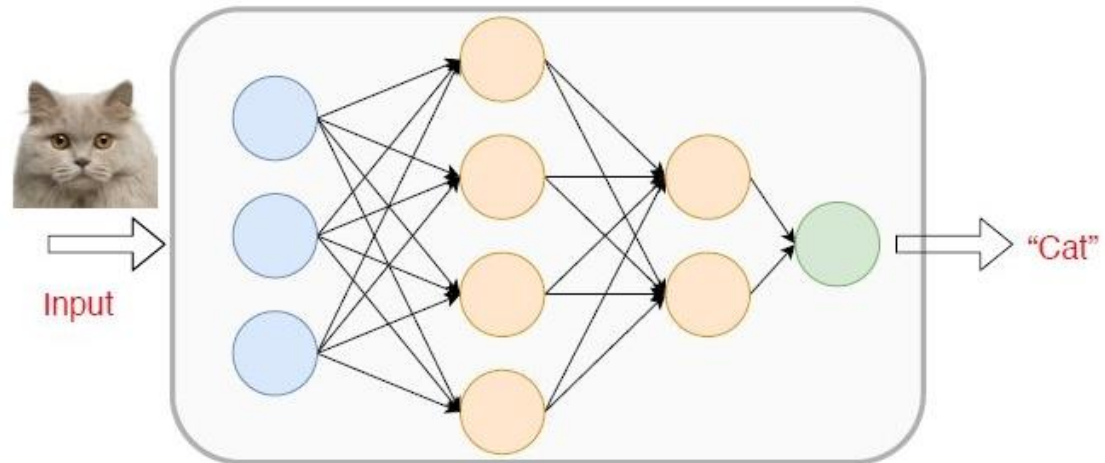
- They are comprised of one or more layers of neurons. Data is fed to the **input layer**, there may be **one or more hidden layers** providing levels of abstraction, and final predictions are made on the **output layer**, also called the visible layer.



- 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

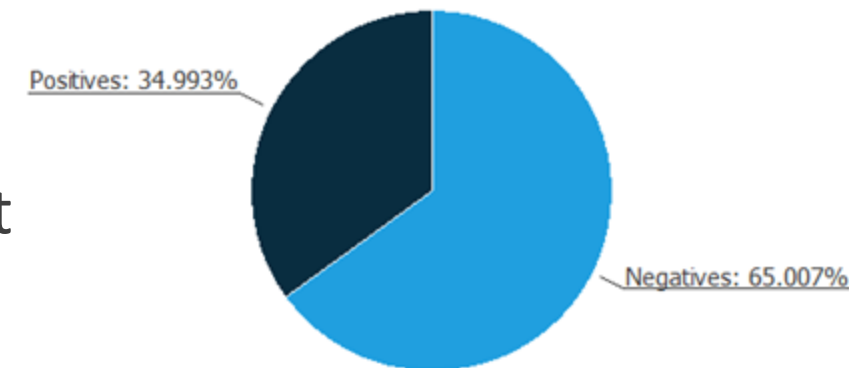
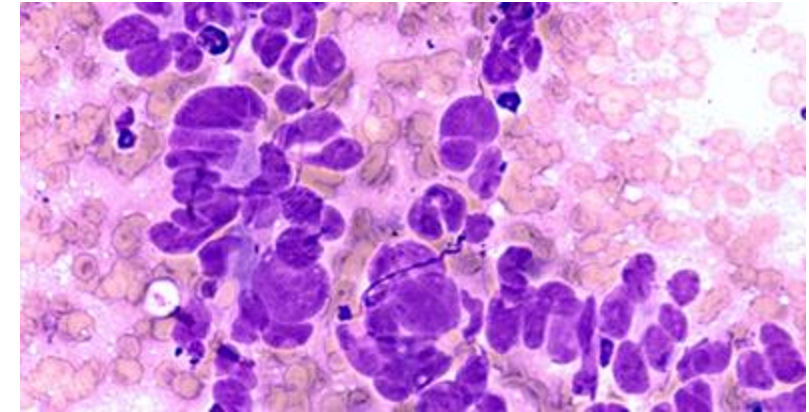
- They are very flexible and can be used to learn a mapping from inputs to outputs.
  - This flexibility allows them to be applied to many different types of data.
- MLP are suitable for:
  - Tabular datasets
  - Classification prediction problems
  - Regression prediction problems
- They can be used aslo for:
  - Image data
  - Text Data
  - Time series data
  - Other types of data





# Deep learning. Sample applications (1)

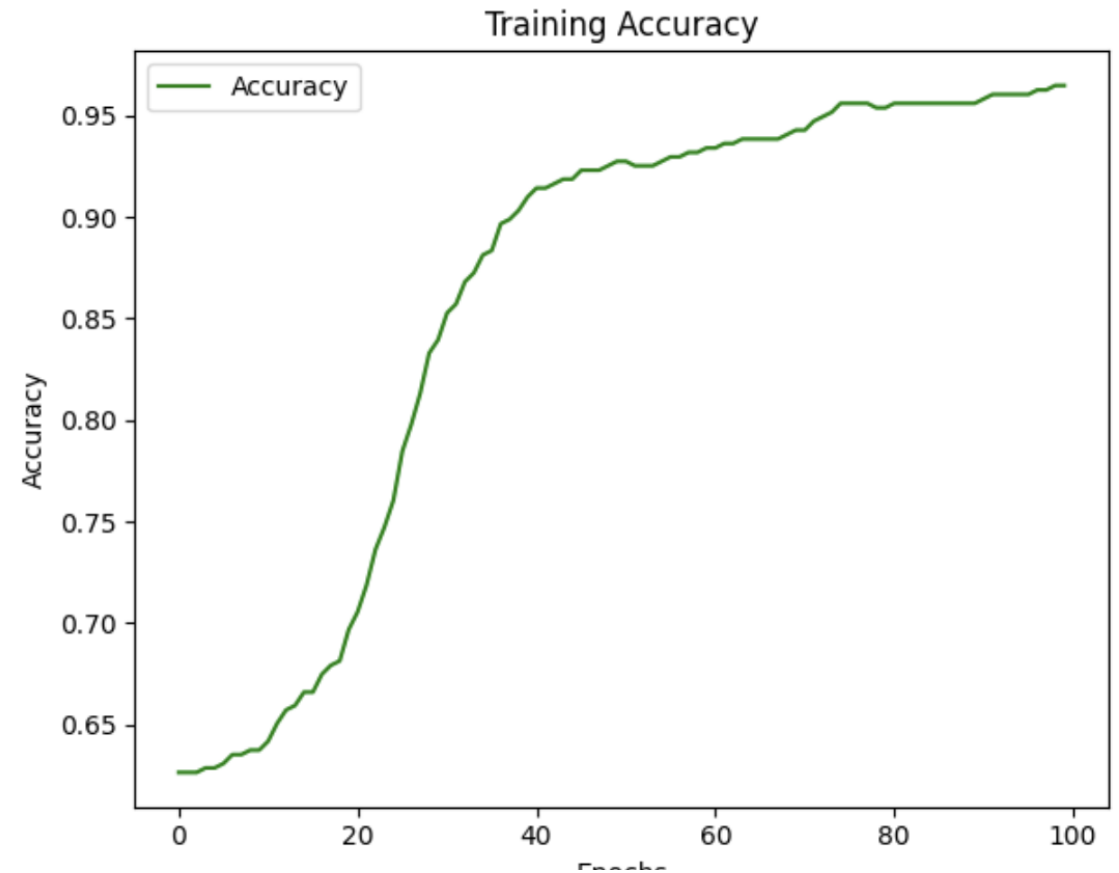
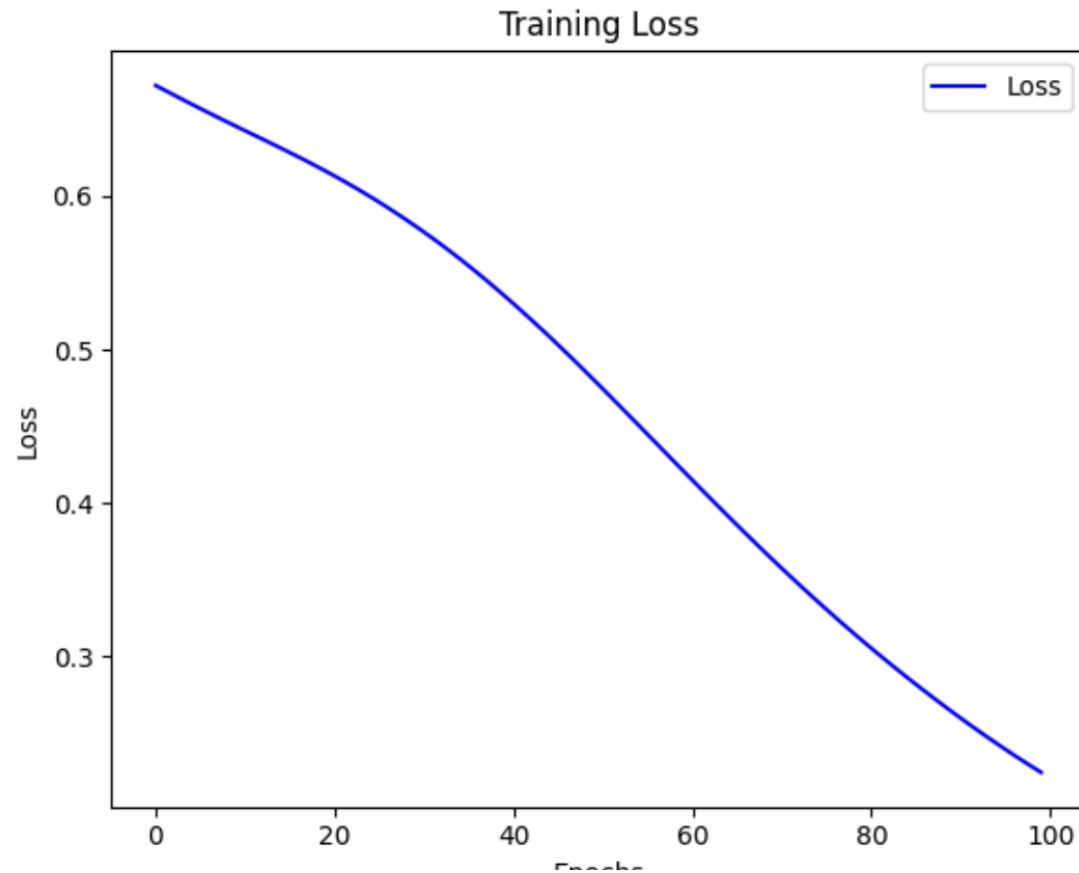
- Breast cancer prediction
  - Assess whether a lump in a breast is malignant (**cancerous**) or benign (**non-cancerous**) from digitized images of a fine-needle aspiration biopsy.
  - The dataset contains 30 features from the images.
- Training dataset, to train the ANN
  - malignant or benign cases.
- Testing dataset, never seen by the ANN during the training phase:
  - guarantee not over-fitting the ANN to training dataset



<https://colab.research.google.com/drive/18Pwt986XRAwdHIJdLjQu27Fw1X8r0eVT?usp=sharing>

# Deep learning. Sample applications (1)

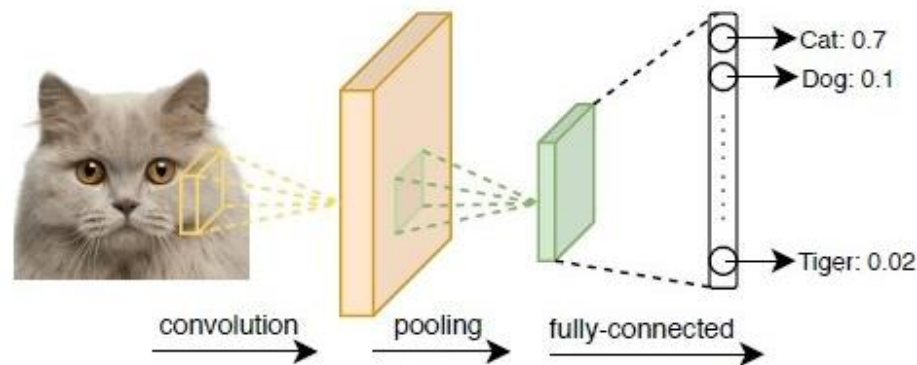
- Breast cancer prediction: results



# Artificial Neural Networks. CNNs

---

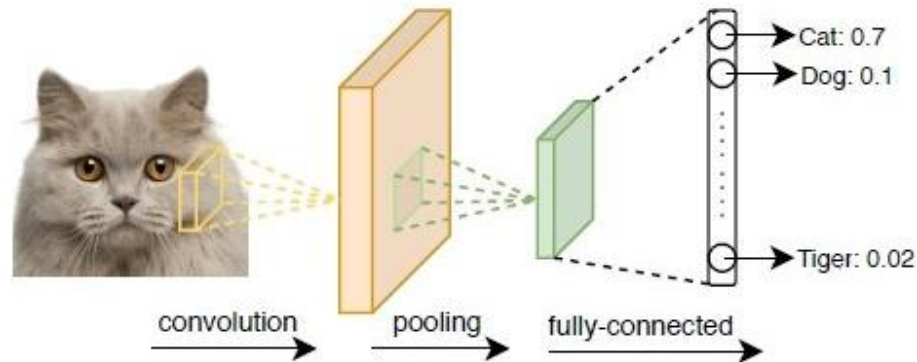
- CNN: A specialized neural network for processing data with a grid-like topology (e.g., images).
  - Convolution layers that apply filters to local regions of the input.
- In 1989, LeCun proposed CNN that was trained by backpropagation



# Artificial Neural Networks. CNNs

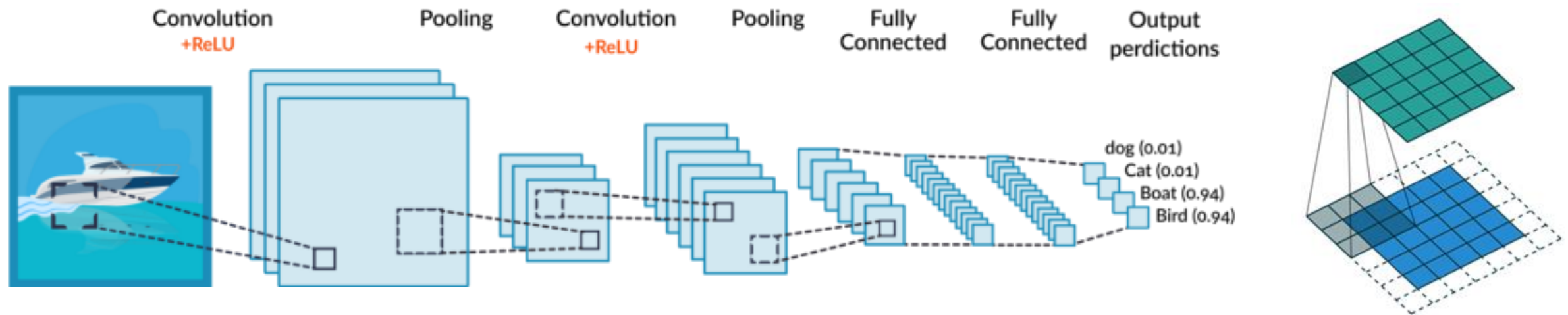
---

- 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)



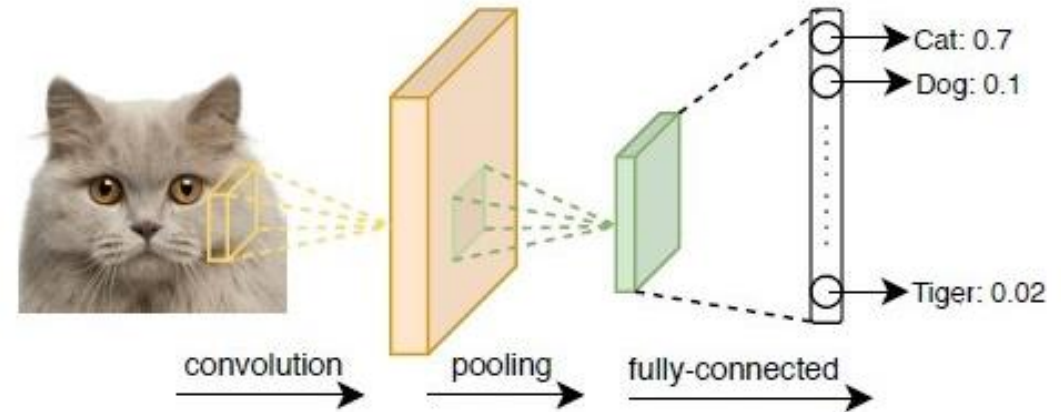
# Artificial Neural Networks. CNNs

- A typical CNN has 4 layers:
  - **Input layer**
  - **Convolution layer:** Applies convolution operations with filters (kernels) to extract features.
  - **Pooling layer:** Reduces dimensionality
  - **Fully connected layer:** Similar to MLP, for classification or regression tasks.



# Artificial Neural Networks. CNNs

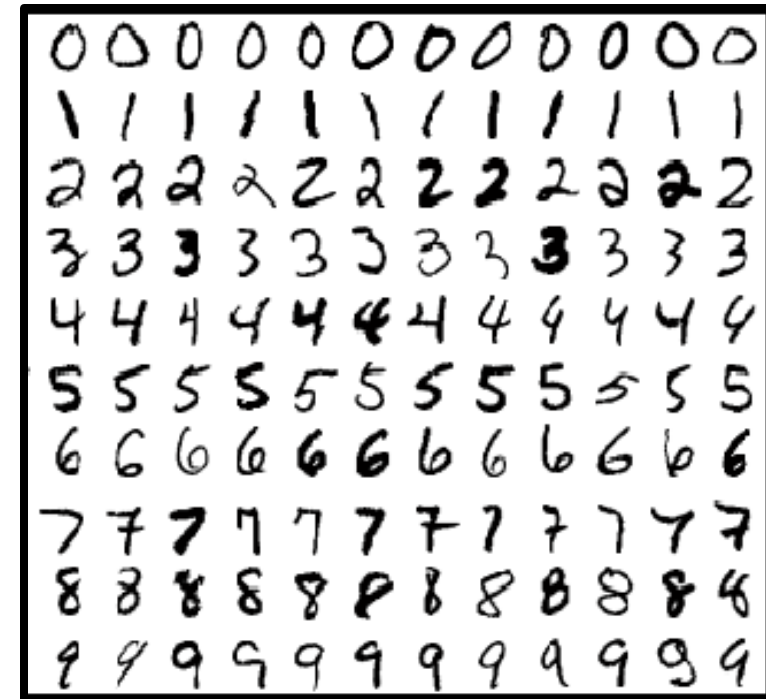
- The benefit of using CNNs is their ability to develop an internal representation of a two-dimensional image. This allows the model to learn position and scale in variant structures in the data, which is important when working with images.
- CNN are suitable for:
  - Image data
  - Classification prediction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



# Deep learning. Sample applications (2)

- Handwritten digits classification (MNIST repository).
- MLP to assign each image a label in the set  $\{0,1,\dots,9\}$ .
- 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.

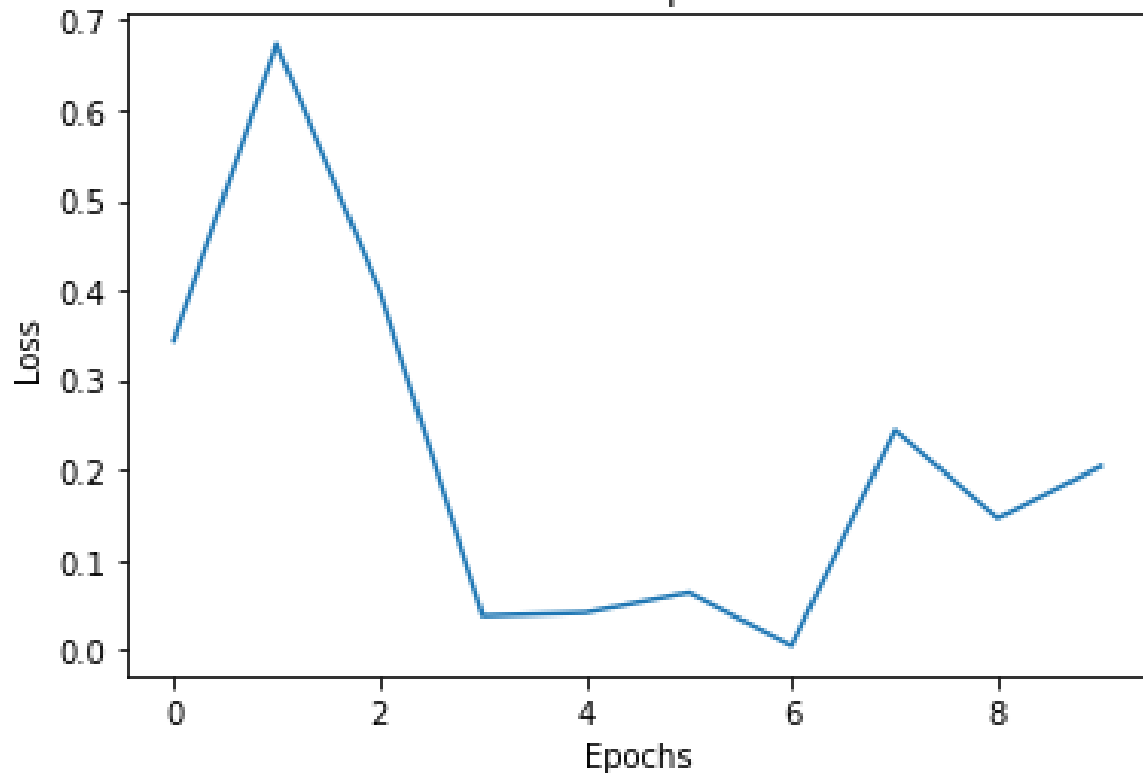
<https://colab.research.google.com/drive/1rYdsv3mHhyoqejbT0R8TBTCa8wcUQH8B?usp=sharing>



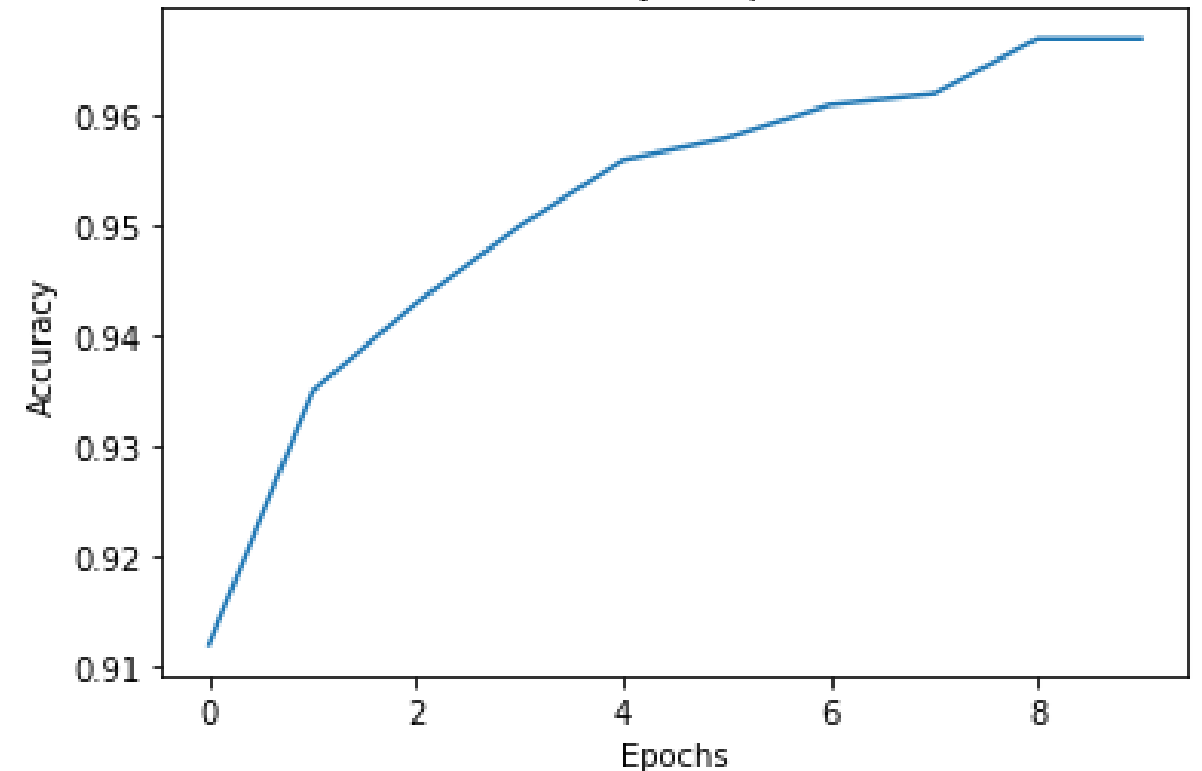
# Deep learning. Sample applications (2)

- MLP for handwritten digits classification: results

Loss vs Epochs



Accuracy vs Epochs





# Deep learning. Sample applications (3)

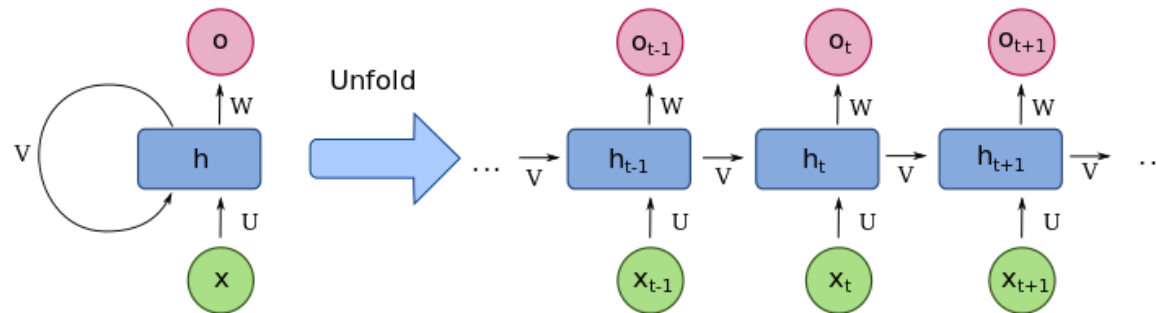
---

- Handwritten digits classification (MNIST repository) with a Convolutional Neural Network

[https://colab.research.google.com/drive/1SRW5OzrbzS0DB17W0VG-FG\\_ykg2Q2nfk?usp=sharing](https://colab.research.google.com/drive/1SRW5OzrbzS0DB17W0VG-FG_ykg2Q2nfk?usp=sharing)

# 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

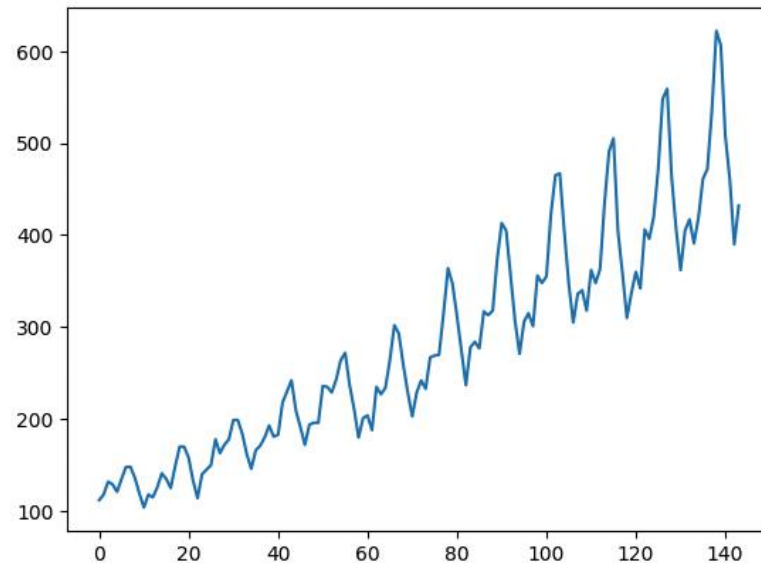
---

- 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. 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)

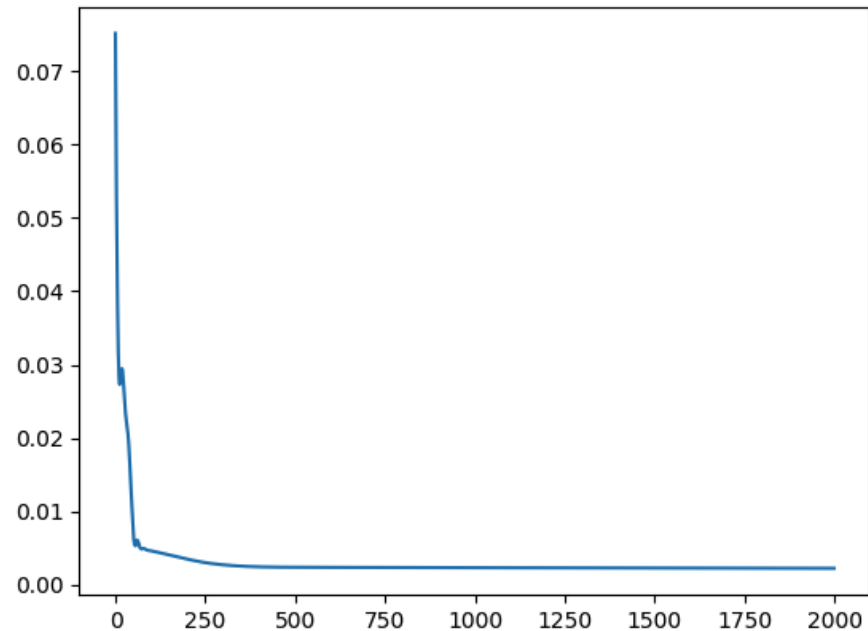


<https://colab.research.google.com/drive/1OhfxhkPWJRfIF0312I5yQ-3CjxhK2QLd?usp=sharing>

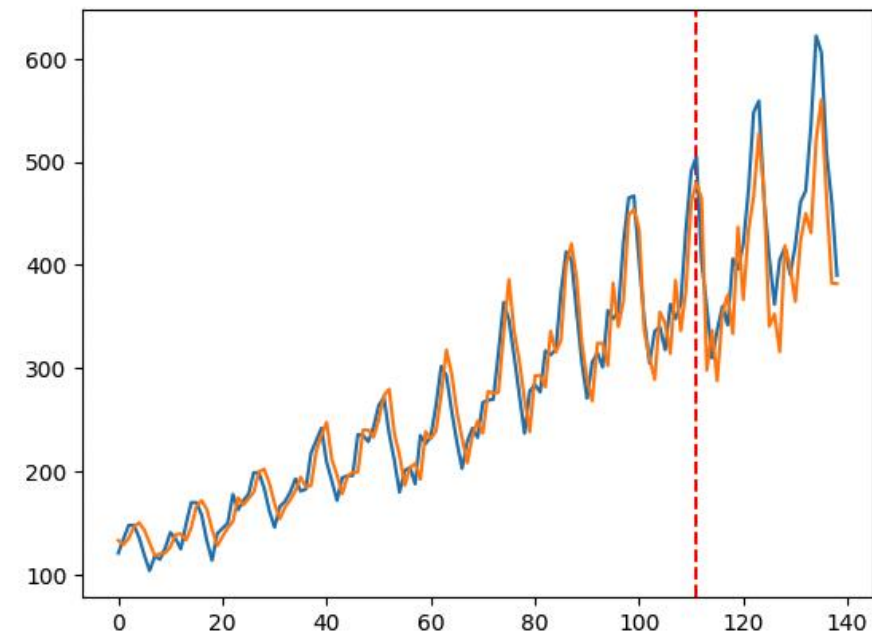
# Deep learning. Example: Basic use of RNN

- RNN for sequence prediction [https://colab.research.google.com/drive/1sNDE114j7xAllxaYbs3G2A8TtpC\\_DyNc?usp=sharing](https://colab.research.google.com/drive/1sNDE114j7xAllxaYbs3G2A8TtpC_DyNc?usp=sharing)

Loss evolution



Time-Series Prediction



Thanks!      Comments?

JAMAL TOUTOUH

toutouh@mit.edu

jamal.es

necol.net

@jamtou