Generative Adversarial Networks

JAMAL TOUTOUH

jamal@lcc.uma.es, toutouh@mit.edu

jamal.es

@jamtou

Jamal Toutouh, Ph.D.

- Postdoctoral Researcher at **Massachusetts Institute of Technology (MIT)**
  - MIT Computer Science & Artificial Intelligence Lab
  - Marie Sklodowska-Curie Postdoctoral fellowship

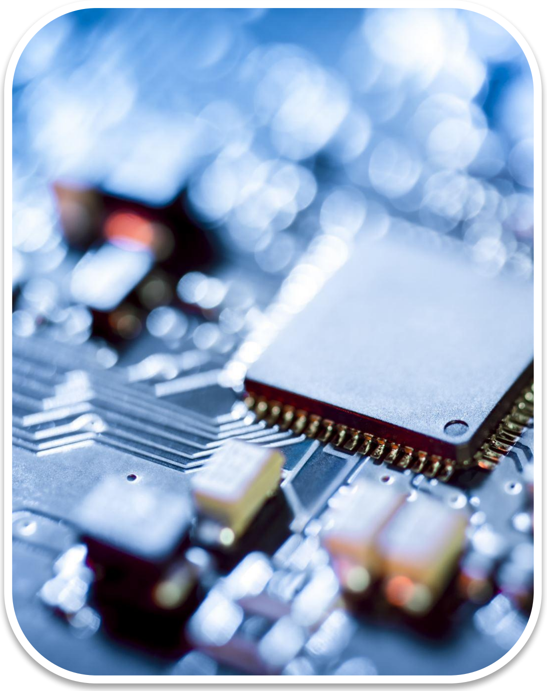
- PhD in Computer Science, **University of Malaga**
- M.Sc. in Software Engineering and Artificial Intelligence, **University of Malaga**
- M.Sc. in Information and Computer Sciences - Intelligent Systems, **University of Luxembourg**

- **Research Interests / Projects**
  - Scalable Machine/Deep Learning
  - Data Science and Large-scale Knowledge Mining
  - Smart Cities / Vehicular Communications
  - Evolutionary Algorithms / Swarm Intelligence

toutouh@mit.edu
www.jamal.es
www.necol.net
@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
  - use Python code to create and use Artificial Neural Networks to address classification 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



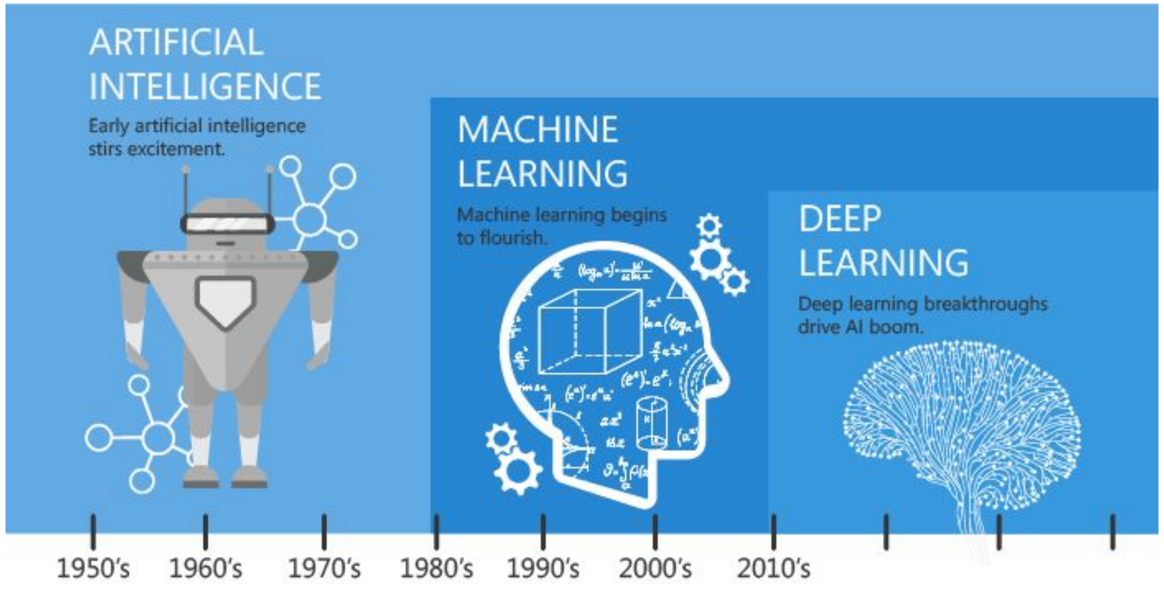
Generative Adversarial Networks for fun

https://thispersondoesnotexist.com/

Main Concepts

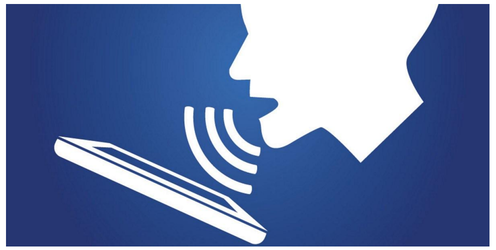
Machine Learning, Deep Learning, AI ....

- **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


ARTIFICIAL INTELLIGENCE
Early artificial intelligence stirs excitement.
MACHINE LEARNING
Machine learning begins to flourish.
DEEP LEARNING
Deep learning breakthroughs drive AI boom.
1950's
1960's
1970's
1980's
1990's
2000's
2010's

Artificial Intelligence Applications

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

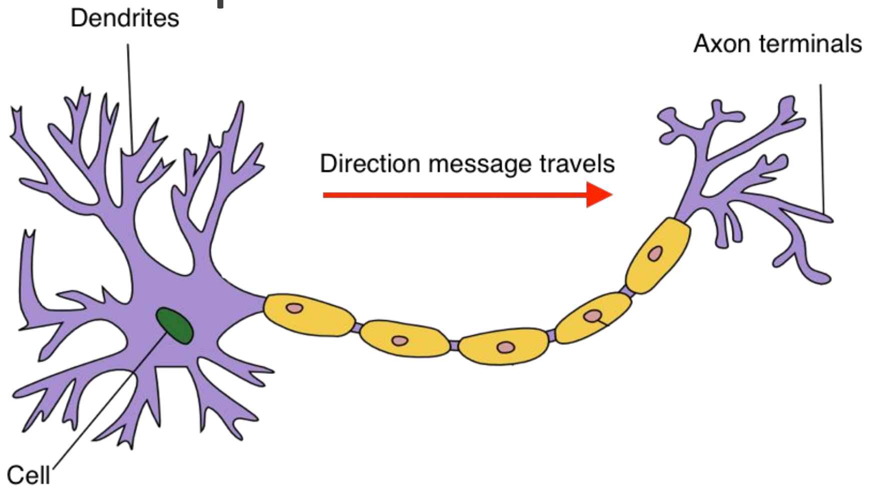


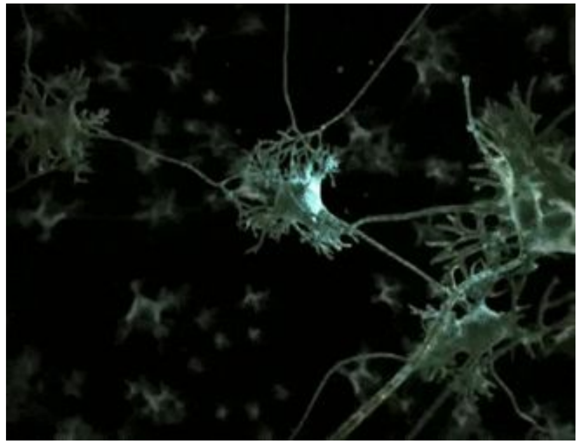
Artificial Neural Networks

Simple Biological Neuron

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.





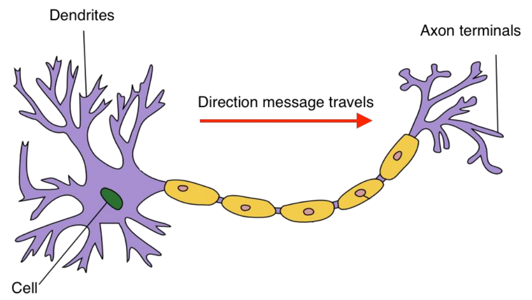
Collective intelligence

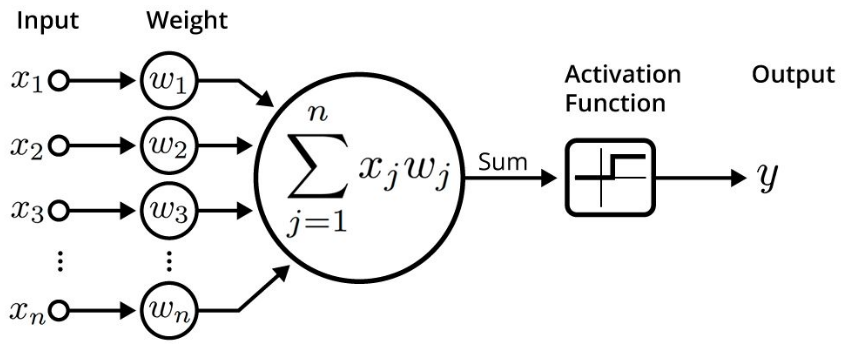
Shared or group intelligence that emerges from collaboration, collective efforts, and/or competition of many agents.

- **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.
- Emergent property from synergies among:
  - data, information, and knowledge;
  - agents;
  - software and hardware.

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

$$x_1 \rightarrow x_1 * w_1$$

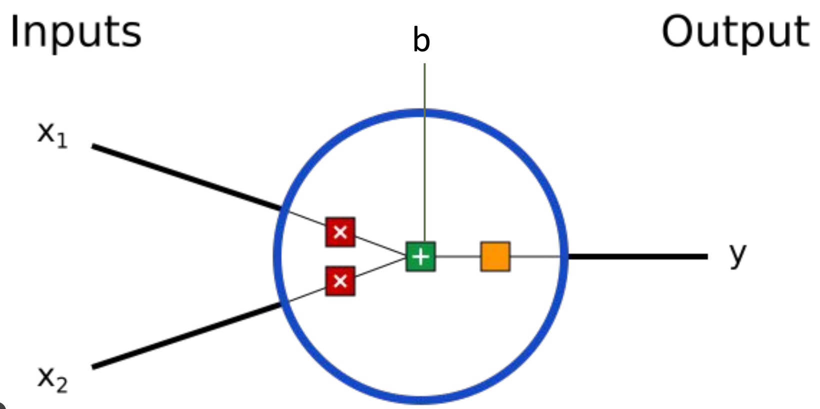
$$x_2 \rightarrow x_2 * w_2$$

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

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



Code: ***basic-neuron.py***

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

$$x_1 \rightarrow x_1 * w_1$$

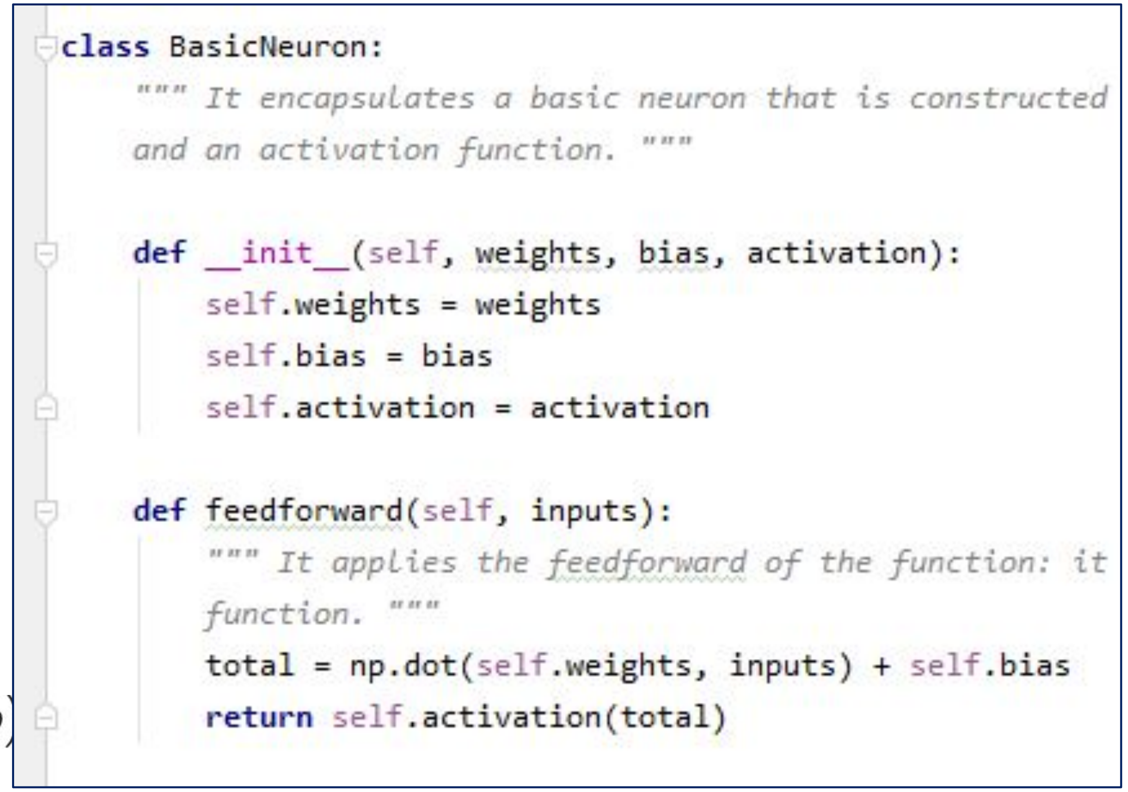
$$x_2 \rightarrow x_2 * w_2$$

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

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

1. The sum is passed through an activation

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

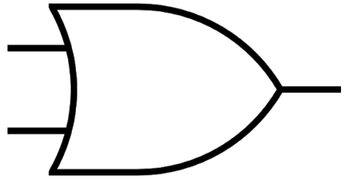


```
class BasicNeuron:
    """ It encapsulates a basic neuron that is constructed
    and an activation function. """

    def __init__(self, weights, bias, activation):
        self.weights = weights
        self.bias = bias
        self.activation = activation

    def feedforward(self, inputs):
        """ It applies the feedforward of the function: it
        function. """
        total = np.dot(self.weights, inputs) + self.bias
        return self.activation(total)
```

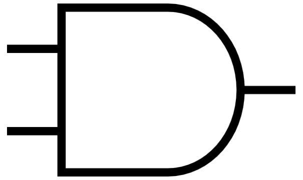
The output of the network depends on the **weights**, the **bias**, and the **activation function**

Artificial Neuron. What can we do?

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



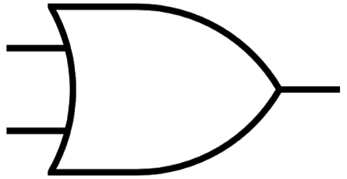
x_1	x_2	x_1 OR x_2
0	0	0
0	1	1
1	0	1
1	1	1

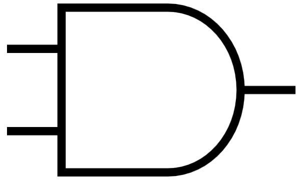


x_1	x_2	x_1 AND x_2
0	0	0
0	1	0
1	0	0
1	1	1

Artificial Neuron. What can we do?

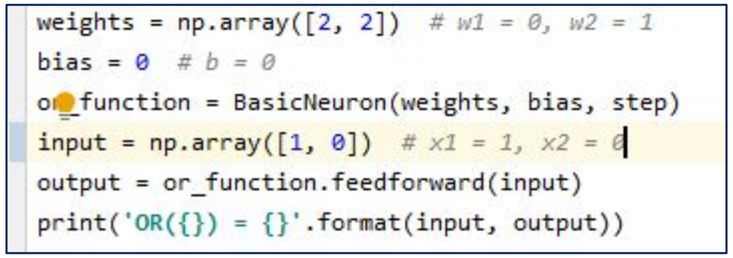
- Try to implement a logic function with the two-input neuron.



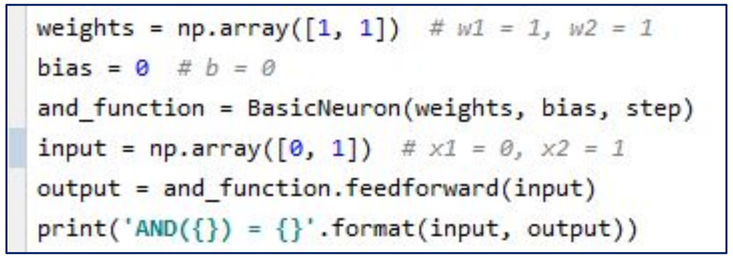


x_1	x_2	x_1 OR x_2
0	0	0
0	1	1
1	0	1
1	1	1

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



```
weights = np.array([2, 2])  # w1 = 0, w2 = 1
bias = 0  # b = 0
or_function = BasicNeuron(weights, bias, step)
input = np.array([1, 0])  # x1 = 1, x2 = 0
output = or_function.feedforward(input)
print('OR({}) = {}'.format(input, output))
```



```
weights = np.array([1, 1])  # w1 = 1, w2 = 1
bias = 0  # b = 0
and_function = BasicNeuron(weights, bias, step)
input = np.array([0, 1])  # x1 = 0, x2 = 1
output = and_function.feedforward(input)
print('AND({}) = {}'.format(input, output))
```

Artificial Neuron. Basic example (2)

- Two-inputs neuron operation:

1. Each input is multiplied by a weight

$$x_1 \rightarrow x_1 * w_1$$

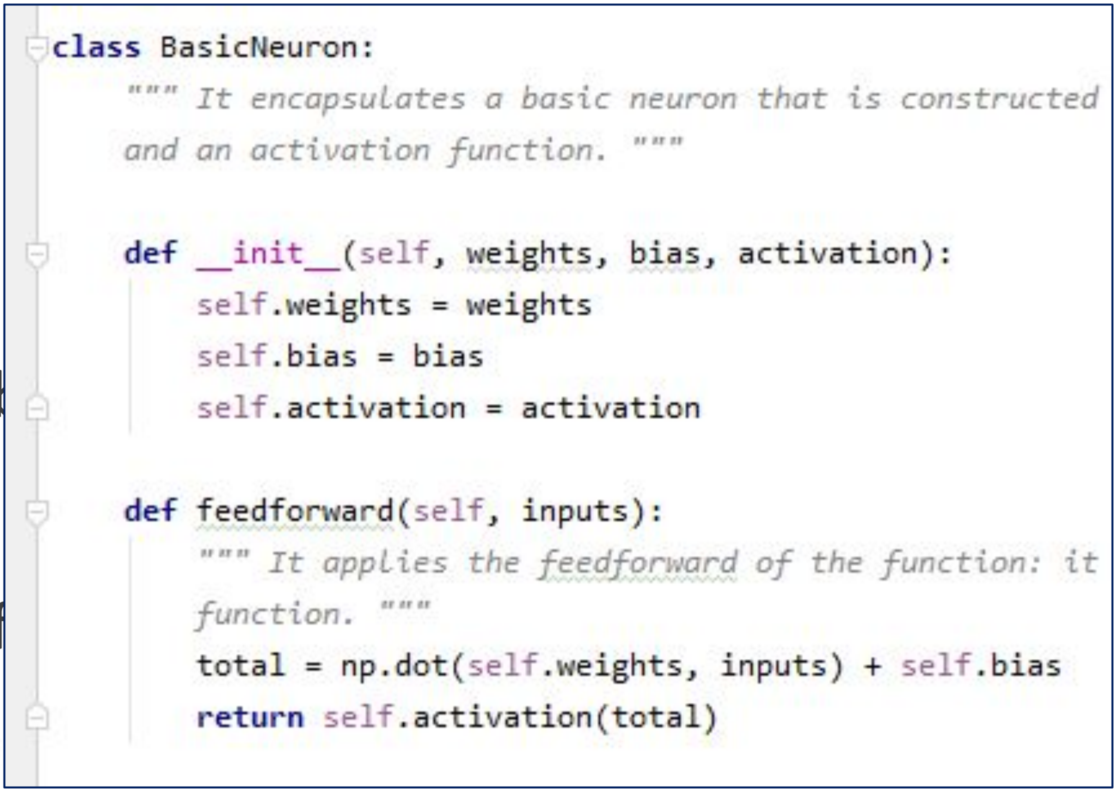
$$x_2 \rightarrow x_2 * w_2$$

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

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

1. The sum is passed through an activation f

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



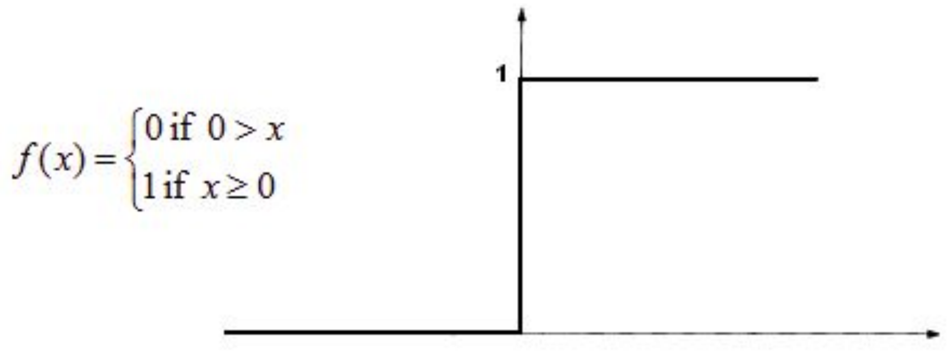
```
class BasicNeuron:
    """ It encapsulates a basic neuron that is constructed
    and an activation function. """

    def __init__(self, weights, bias, activation):
        self.weights = weights
        self.bias = bias
        self.activation = activation

    def feedforward(self, inputs):
        """ It applies the feedforward of the function: it
        function. """
        total = np.dot(self.weights, inputs) + self.bias
        return self.activation(total)
```

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

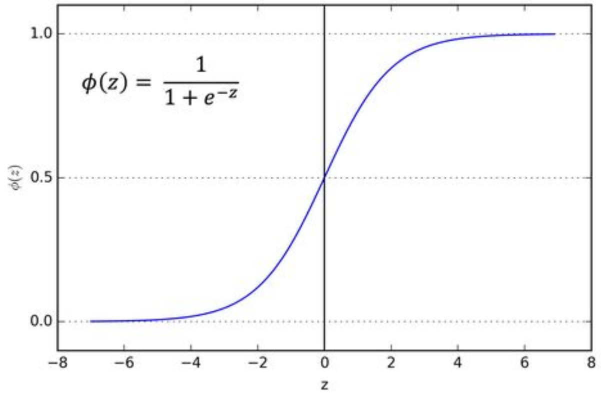
Activation Function

- 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.

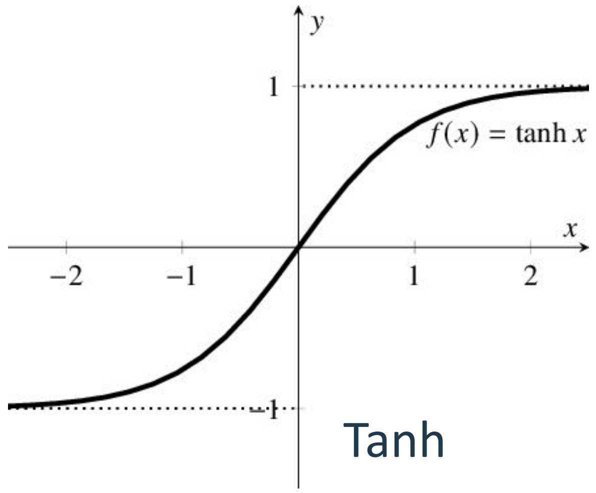
Activation Function. Types



Step

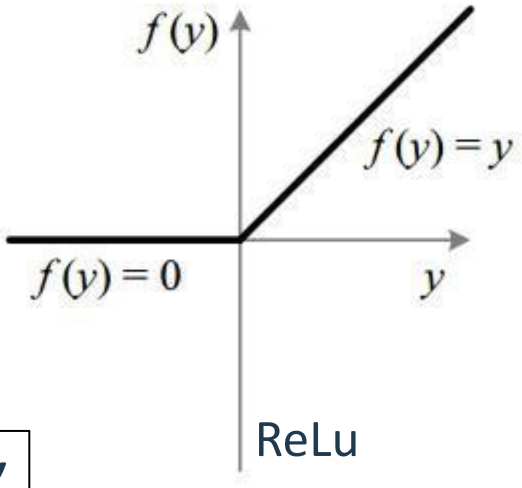


Sigmoid

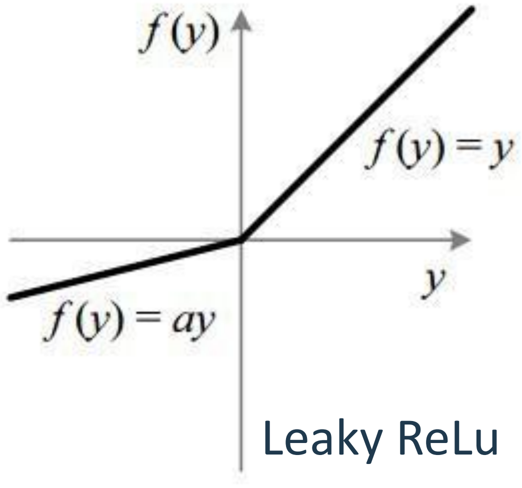


Tanh

Code: ***neuron.py***



ReLu



Leaky ReLu

Artificial Neural Networks

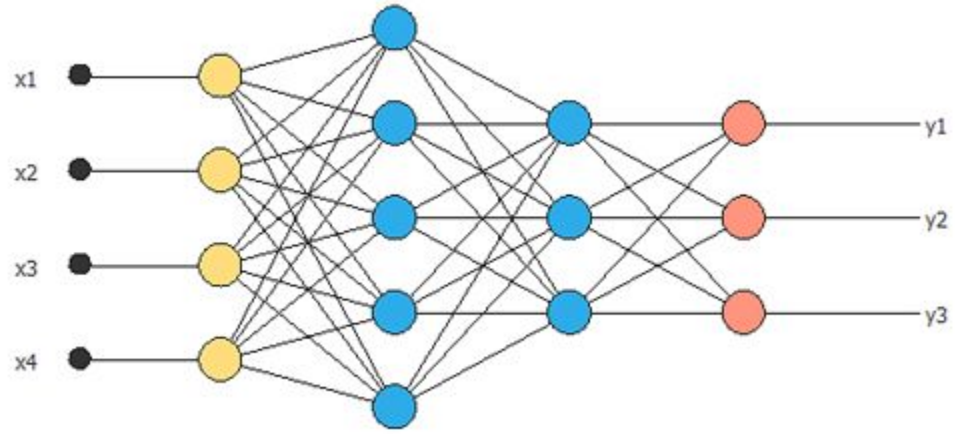
A neural network is a bunch of neurons connected together.

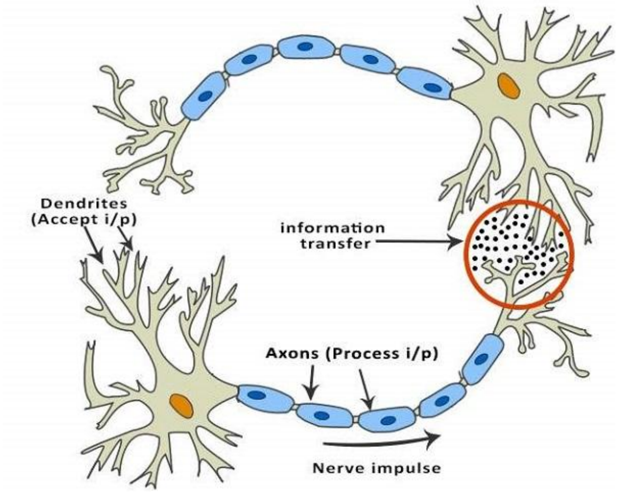
Neural networks are typically **organized in layers**.

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**.

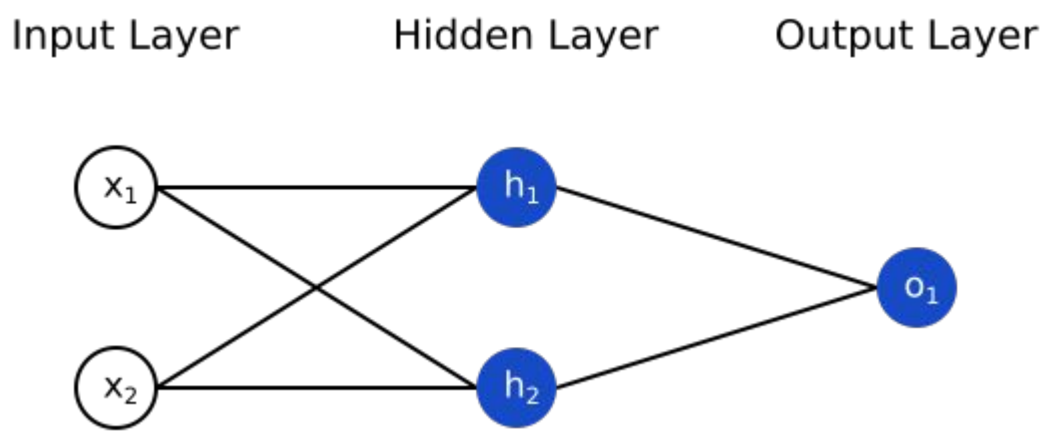
The hidden layers then link to an **output layer**.





Artificial Neural Networks. Code

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



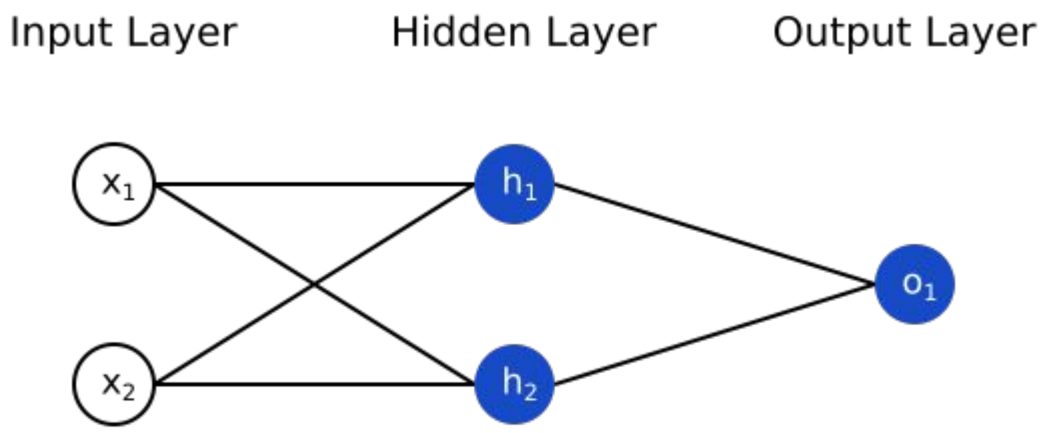
Code: ***basic-two-layer-neural-network.py***

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

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

Artificial Neural Networks. Code

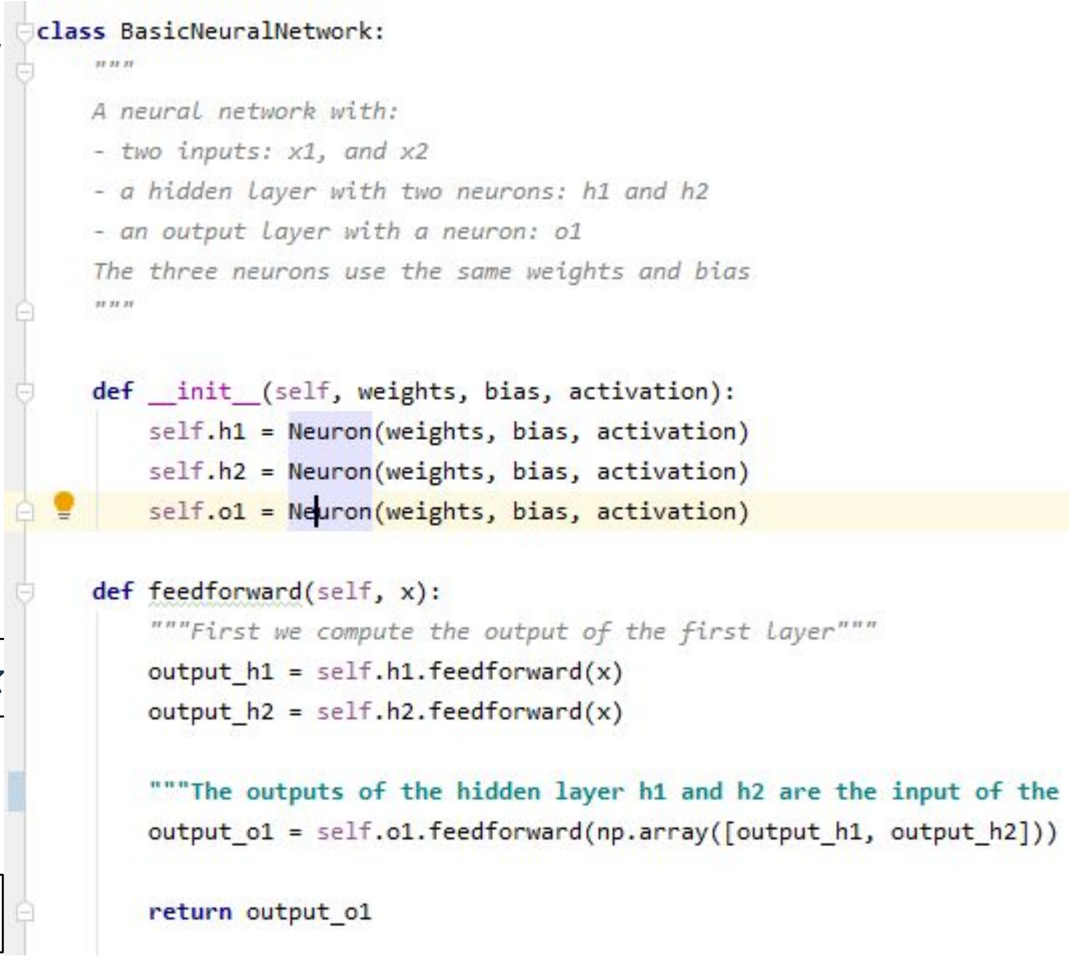
- **Example 1**: Two inputs, two neurons in a hidden lay



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

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

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

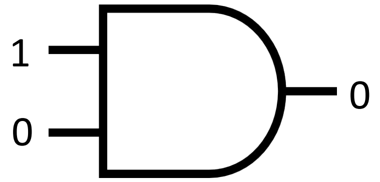


```
class BasicNeuralNetwork:
    """
    A neural network with:
    - two inputs: x1, and x2
    - a hidden layer with two neurons: h1 and h2
    - an output layer with a neuron: o1
    The three neurons use the same weights and bias
    """

    def __init__(self, weights, bias, activation):
        self.h1 = Neuron(weights, bias, activation)
        self.h2 = Neuron(weights, bias, activation)
        self.o1 = Neuron(weights, bias, activation)

    def feedforward(self, x):
        """First we compute the output of the first layer"""
        output_h1 = self.h1.feedforward(x)
        output_h2 = self.h2.feedforward(x)

        """The outputs of the hidden layer h1 and h2 are the input of the
        output_o1 = self.o1.feedforward(np.array([output_h1, output_h2]))

        return output_o1
```

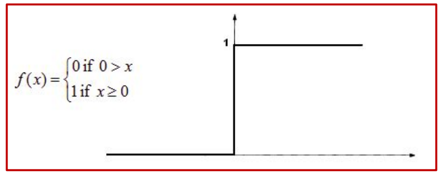
How do ANNs Learn?

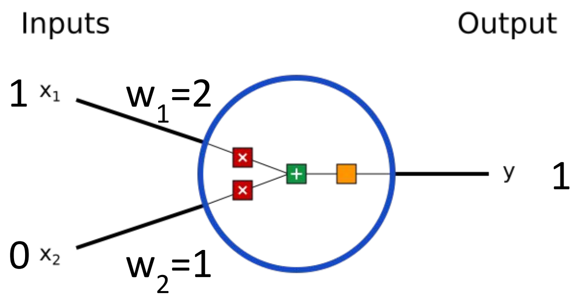
The **output** of the ANN depends on the **weights**

Learning consist on **updating the weights** to get a desired output, i.e., **minimize the error**



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



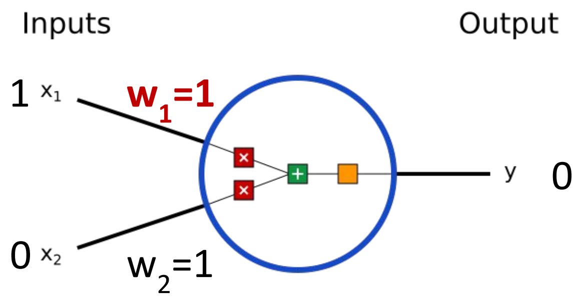


y = step(1 x 2 + 0 x 1, 2)
y = step(2, 2) = 1

Compute error

Error = MSE = 1

Update weights

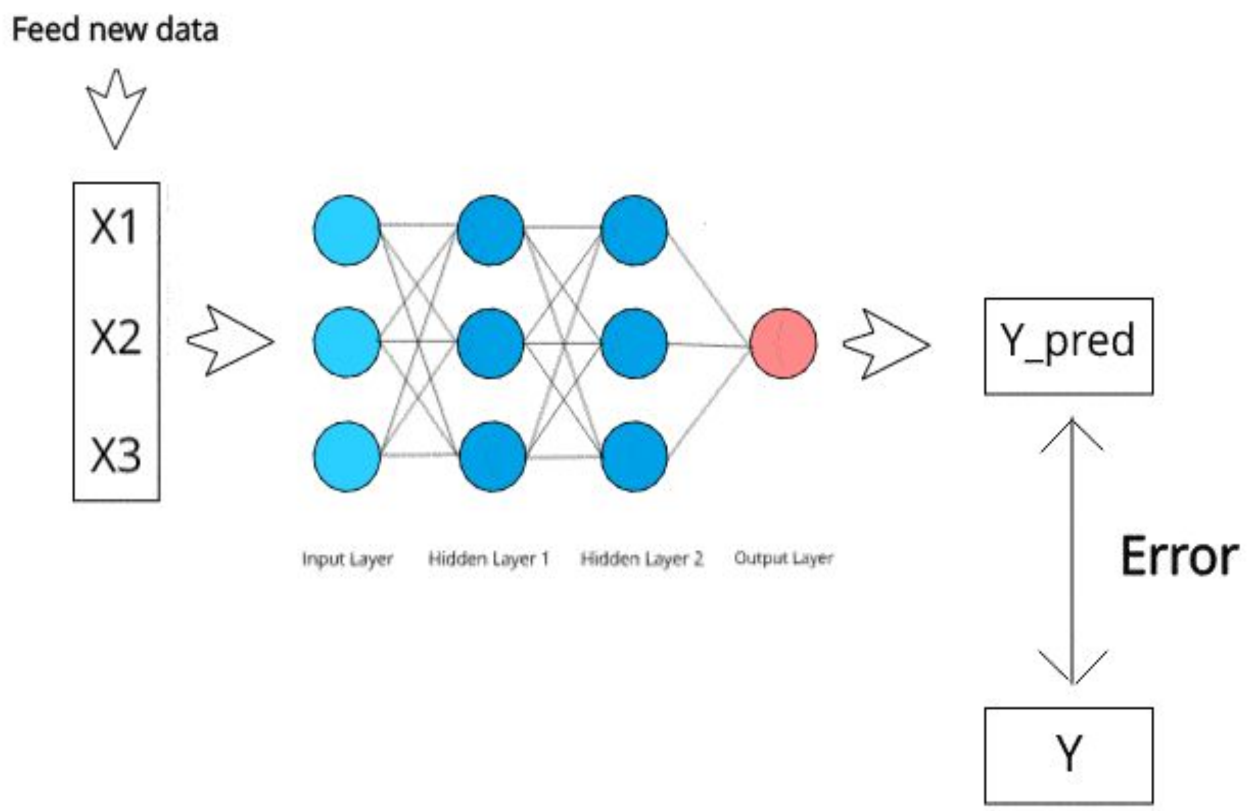


y = step(1 x 1 + 0 x 1, 2)
y = step(1, 2) = 0

How do ANNs Learn?

ANNs learn by solving the **optimization problem** of reducing the error in terms of ***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.



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.

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

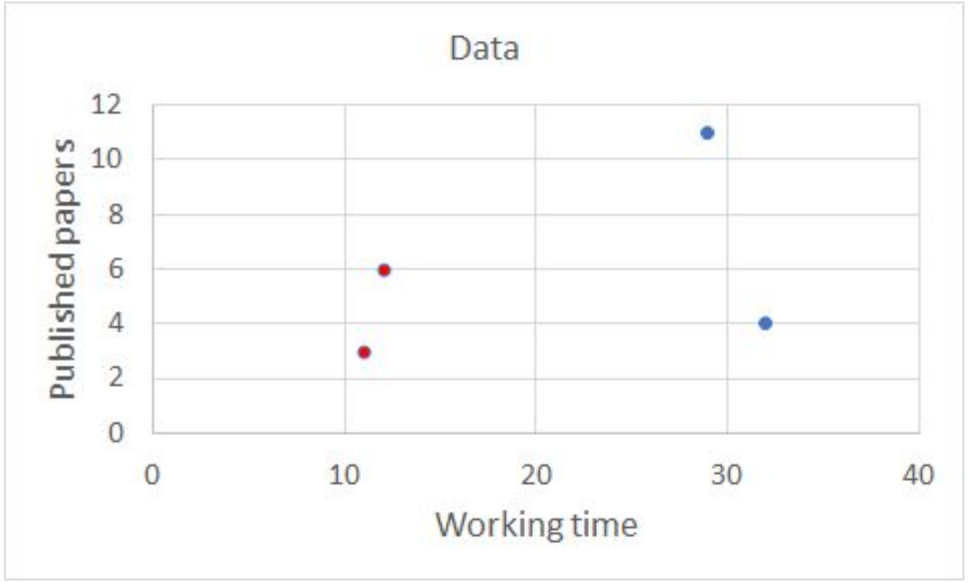
Implementation. Google colab examples

- Generic utilization of Colab: image handling and processing using OpenCV
  https://colab.research.google.com/drive/1EaeyhiUIY0_iw6HMdITUc6JcdaTbf_-E
- ANN examples in Colab
  1. Basic Neuron
     https://colab.research.google.com/drive/12nIfujUWtMMhsykQttKJ8pN8mPoRBjMt
  2. Neuron with different activation functions
     https://colab.research.google.com/drive/1xnEqRbwPxvHy9AowlRUmn8702eUoTkdZ
  3. Two layers ANN
     https://colab.research.google.com/drive/1vxx4rOeDV9LhoABOW_sJz40JnRQihLG6

Implementation. Google colab examples

- ANN examples in Colab
  4. Training two layers ANN
  https://colab.research.google.com/drive/15p4KIxBR5geoy3rDTh1cQq9DI7mmbJJG
  5. Training two layers ANN (using PyTorch library)
  https://colab.research.google.com/drive/1Ri5MF1JYfpQwEqP2IrzEkGx-F3u1uZm5

	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**

Chainer mxnet Caffe2

Microsoft CNTK TensorFlow Keras

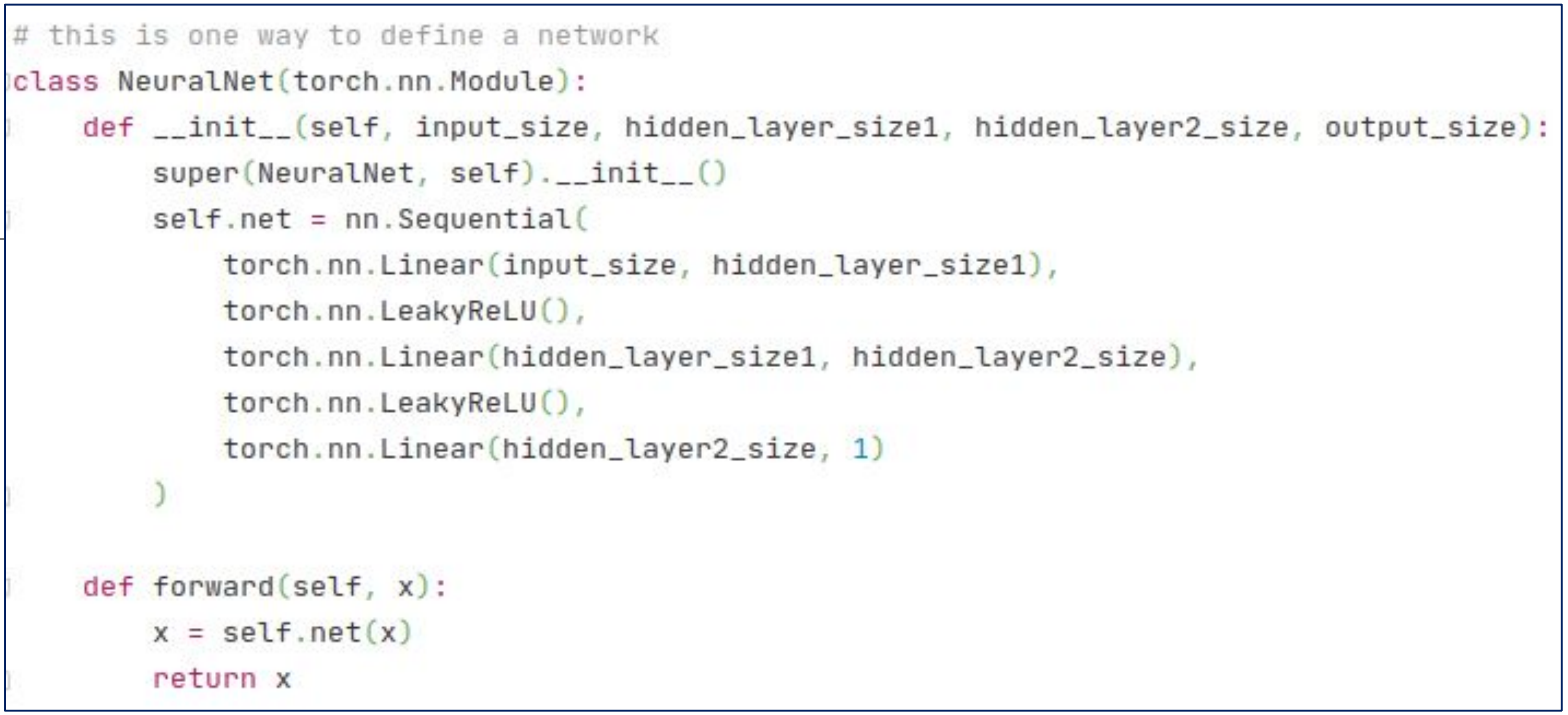
GLUON PYTORCH theano

- Library: PyTorch
  - Deep learning framework/library for python, developed by Facebook.
  - PyTorch has own data structures that provide automatic operations on tensors.

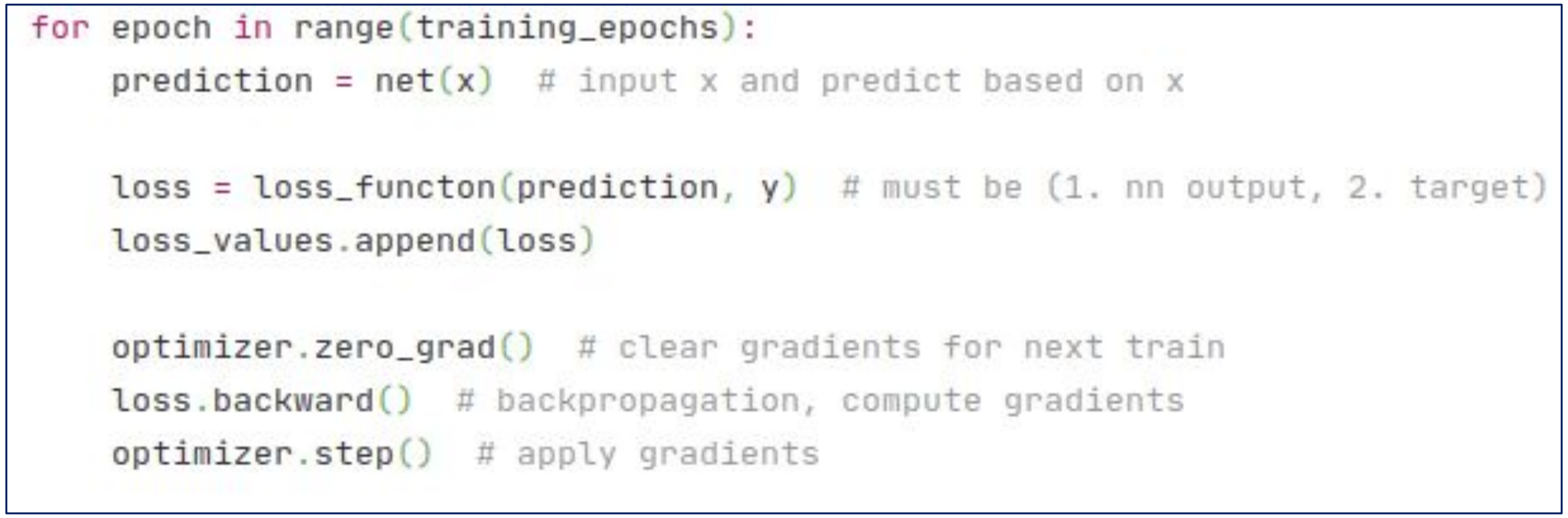
Deep Learning

ANN results get better with

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



```python
# this is one way to define a network
class NeuralNet(torch.nn.Module):
    def __init__(self, input_size, hidden_layer_size1, hidden_layer2_size, output_size):
        super(NeuralNet, self).__init__()
        self.net = nn.Sequential(
            torch.nn.Linear(input_size, hidden_layer_size1),
            torch.nn.LeakyReLU(),
            torch.nn.Linear(hidden_layer_size1, hidden_layer2_size),
            torch.nn.LeakyReLU(),
            torch.nn.Linear(hidden_layer2_size, 1)
        )

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



```python
for epoch in range(training_epochs):
    prediction = net(x)  # input x and predict based on x

    loss = loss_functon(prediction, y)  # must be (1. nn output, 2. target)
    loss_values.append(loss)

    optimizer.zero_grad()  # clear gradients for next train
    loss.backward()  # backpropagation, compute gradients
    optimizer.step()  # apply gradients
```

Deep Learning

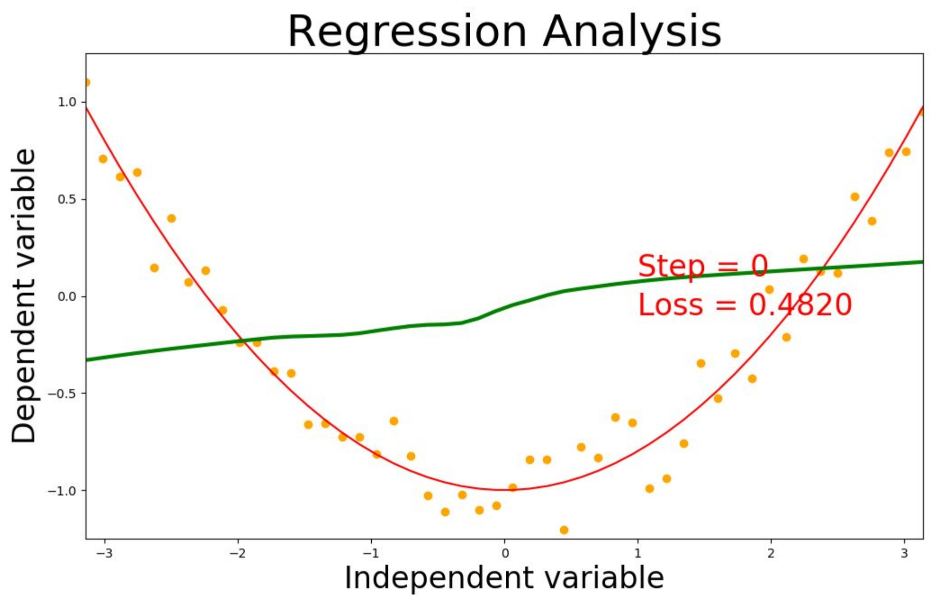
ANN results get better with

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

Example:

Deep Learning for regression

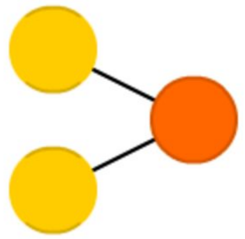
https://colab.research.google.com/drive/1HIU_2w2ELvC8tzPkJXjtwqdoVJWIQghO



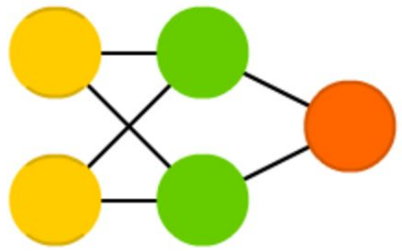
Regression Analysis - Loss
Loss
Training epoch
Loss = 0.4820
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100

Artificial Neural Networks. Types

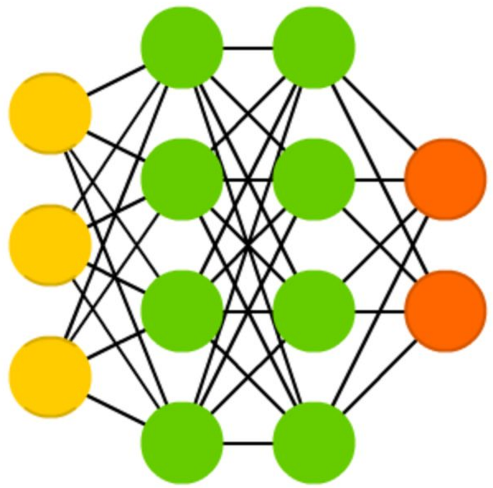
Perceptron (P)



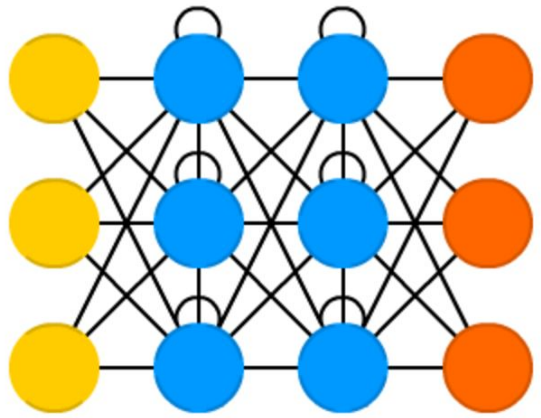
Feed Forward (FF)



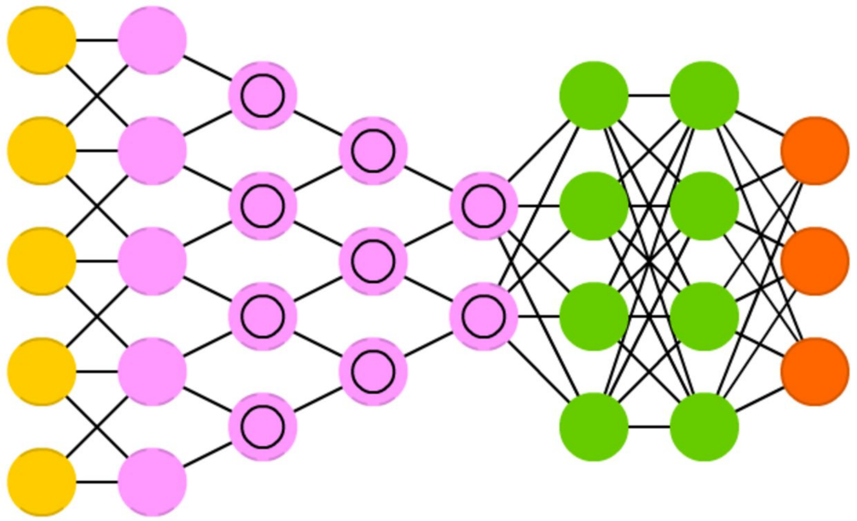
Deep Feed Forward (DFF)



Recurrent Neural Network (RNN)

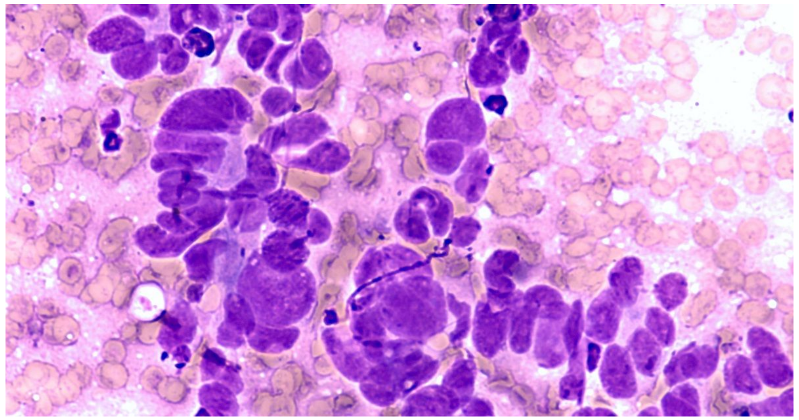


Deep Convolutional Network (DCN)



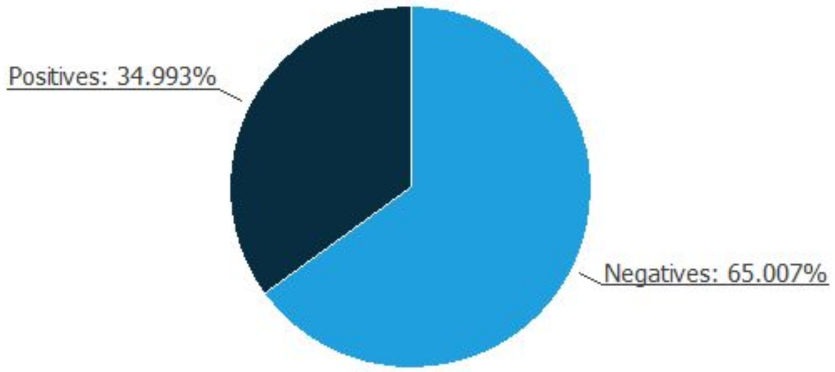
Deep learning. Sample applications (1)

- 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.



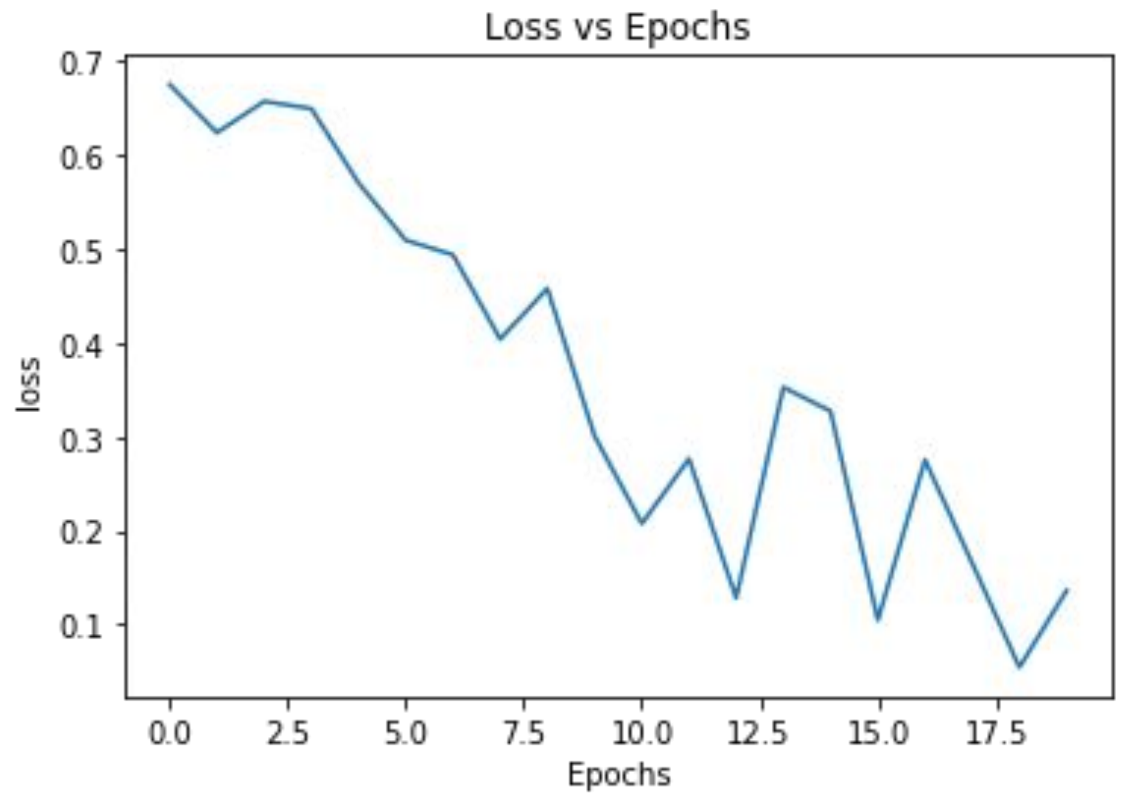
- Training dataset, to train the ANN
  - malignant or benign cases.
- Testing dataset, never seen by the ANN during the training phase:
  - guarantee not over-fitting the ANN to training dataset

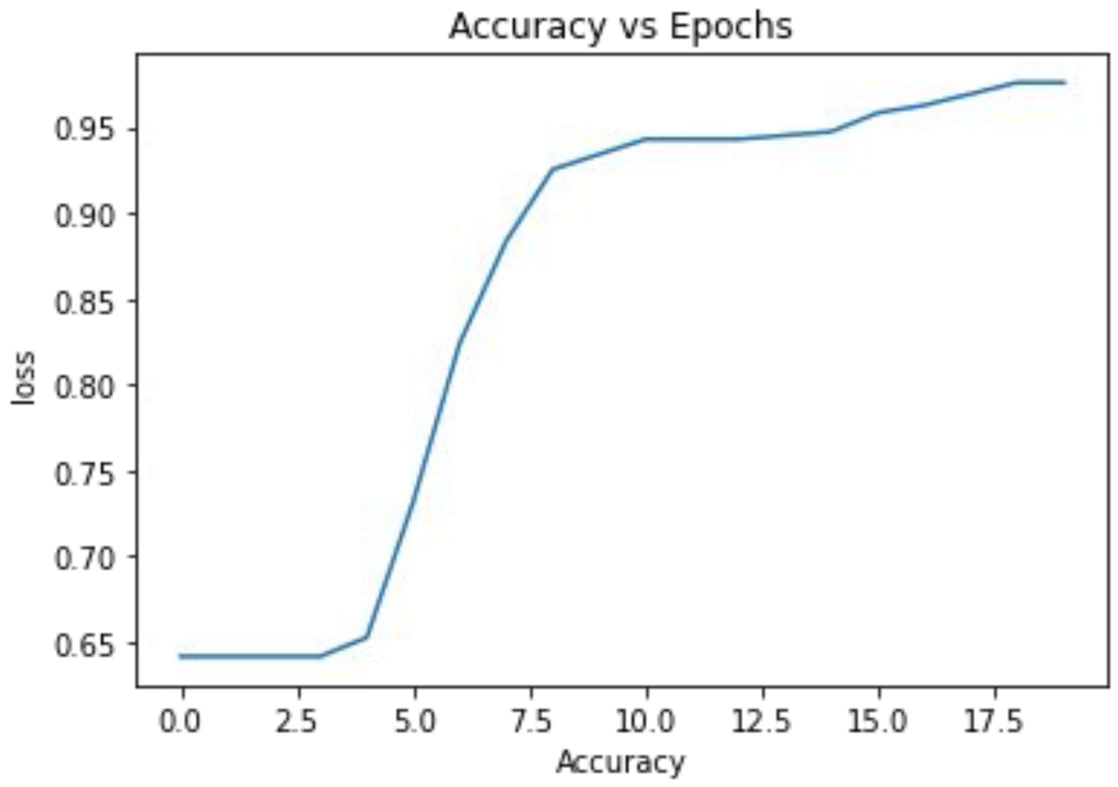
https://colab.research.google.com/drive/1jjS5aDG-GW9y4vkCim6OH_EyLHMXizVT



Deep learning. Sample applications (1)

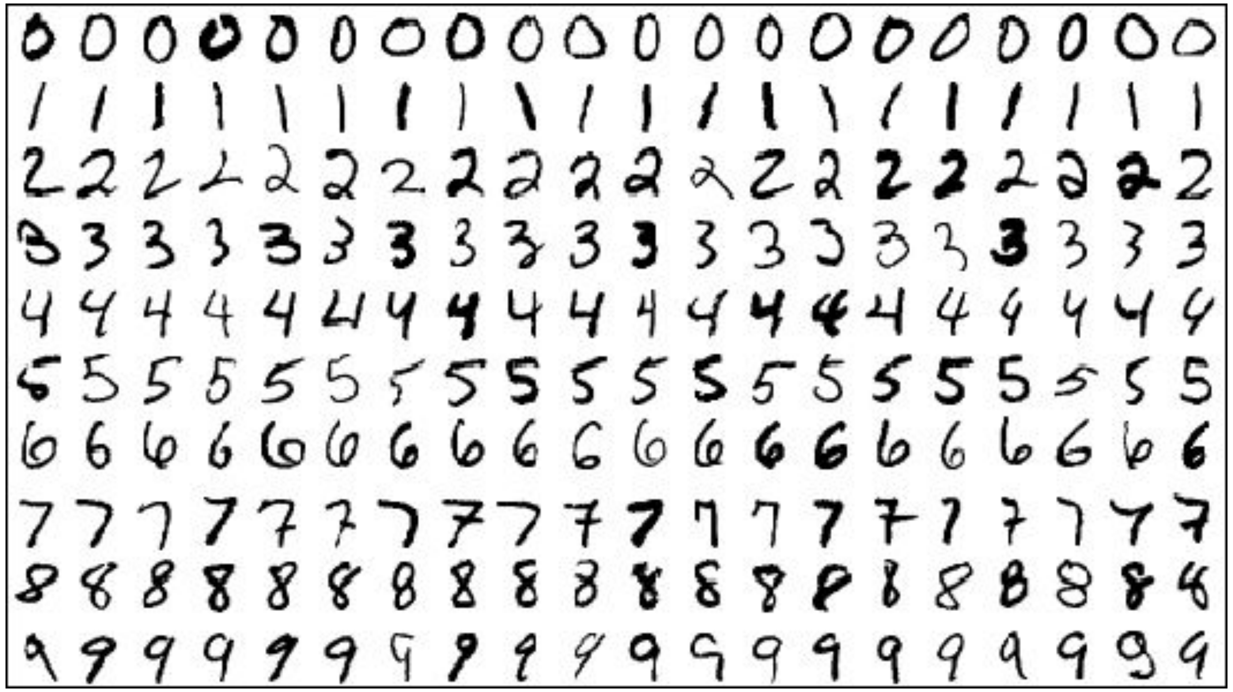
- Breast cancer prediction: results





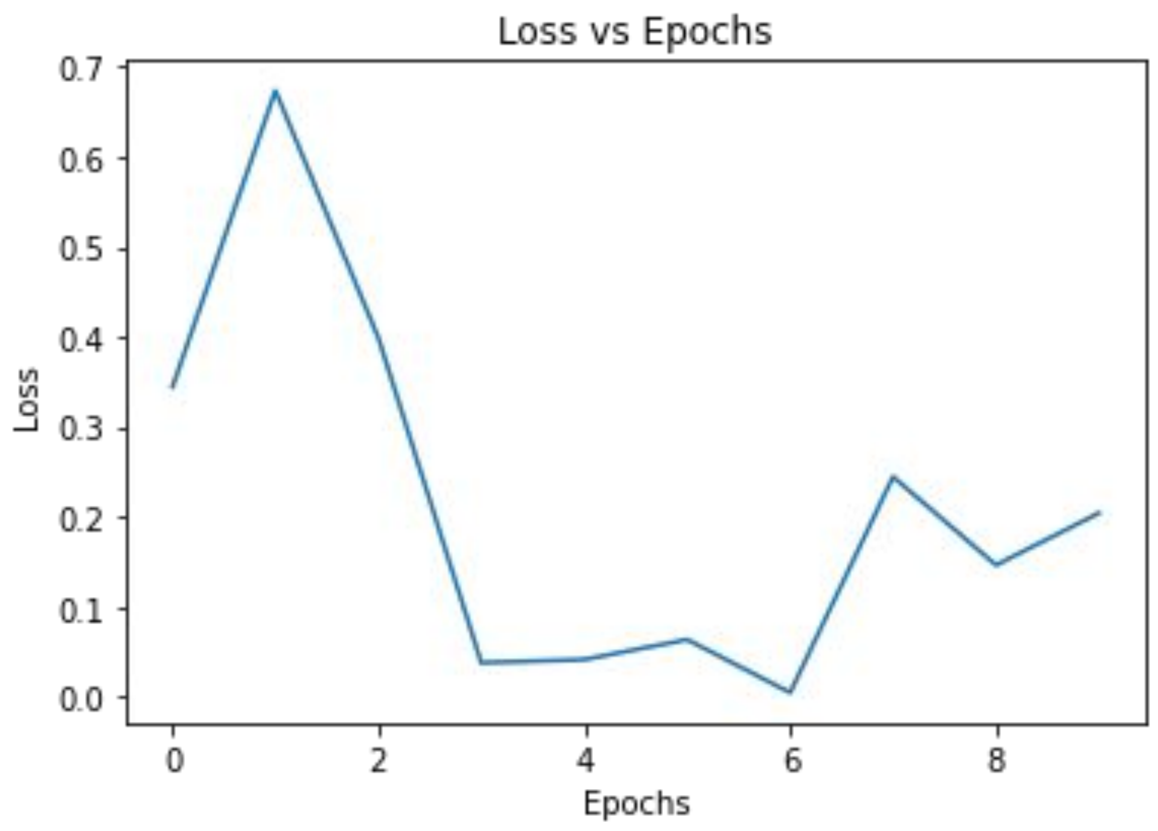
Deep learning. Sample applications (2)

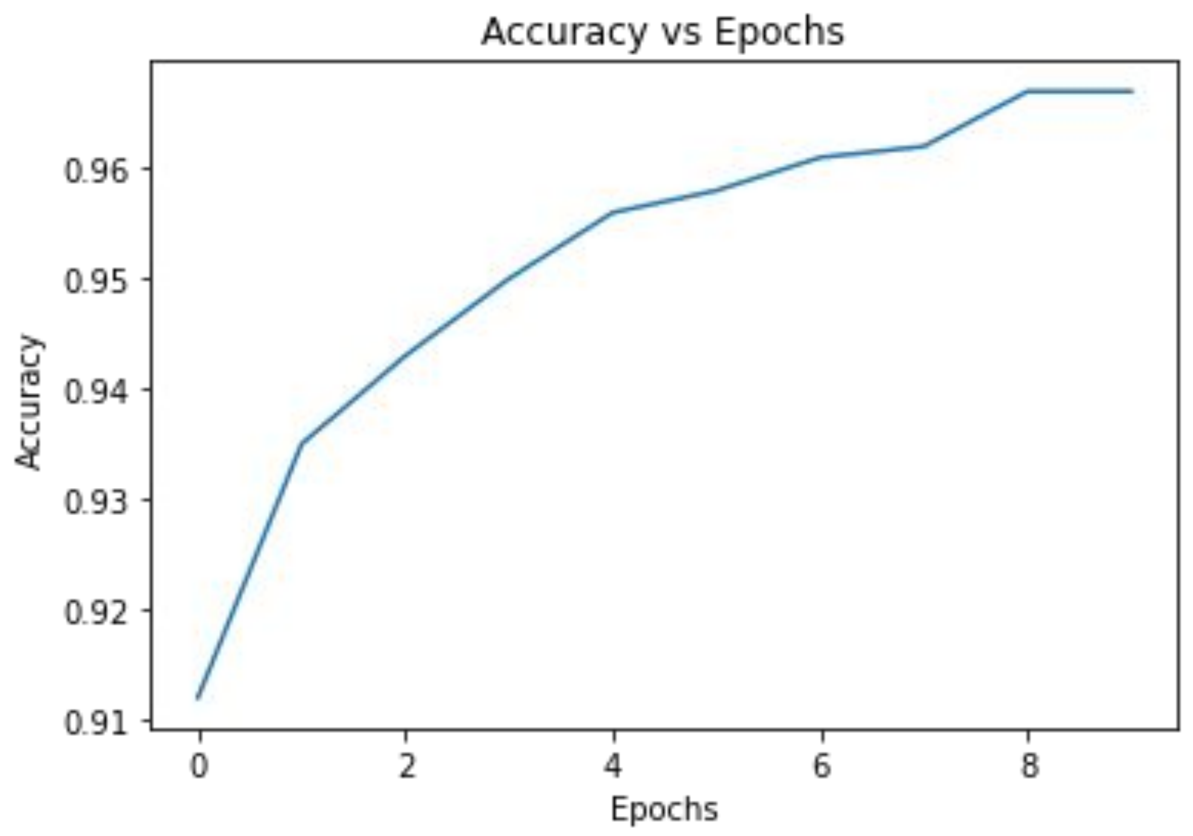
- Handwritten digits classification (MNIST repository).
- MLP to assign each image a label in the set {0,1,...,9}.
- Training dataset, to train the ANN
  - ‘what numbers 0 through 9 look like’.
- Testing dataset, never seen by the ANN during the training phase:
  - guarantee the ANN is not over-fitted to the training dataset,
  - assure the ANN can label independent items properly.
- colab.research.google.com/drive/1liK0JOZjuNLpZG5wsfJ6RQq1R5f3iHFi



Deep learning. Sample applications (2)

- MLP for handwritten digits classification: results





Deep learning. Sample applications (3)

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

- colab.research.google.com/drive/1cdq6ABdnKDgDpfz0RP_Kx0THSniMbz9A

Learning

- Supervised learning: labeled datasets/classification and regression problems
- Unsupervised learning: unlabeled datasets/clustering and dimensionality reduction problems.

- Next talk.