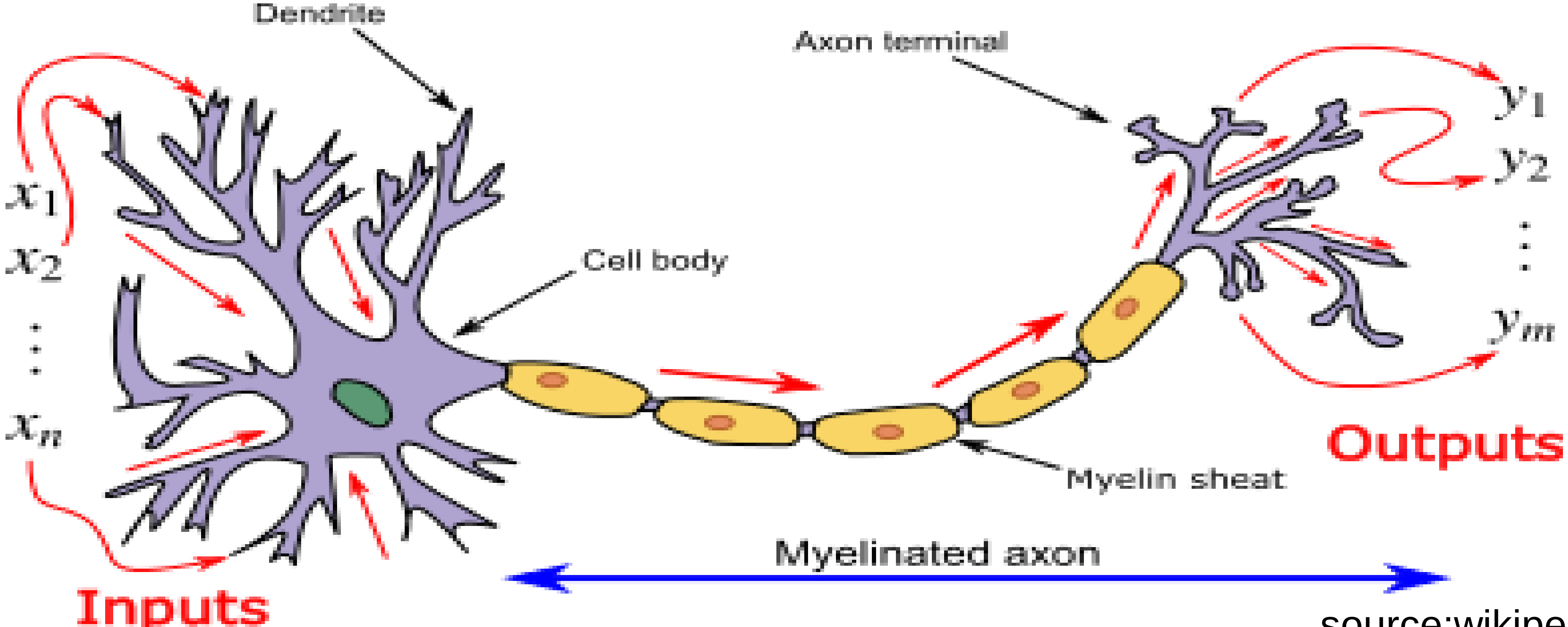
DEEP LEARNING
part I: perceptron

marco milanesio
MS DATA SCIENCE 2020-2021

UNIVERSITÉ CÔTE D'AZUR

neural networks

- recognize patterns
- human brain

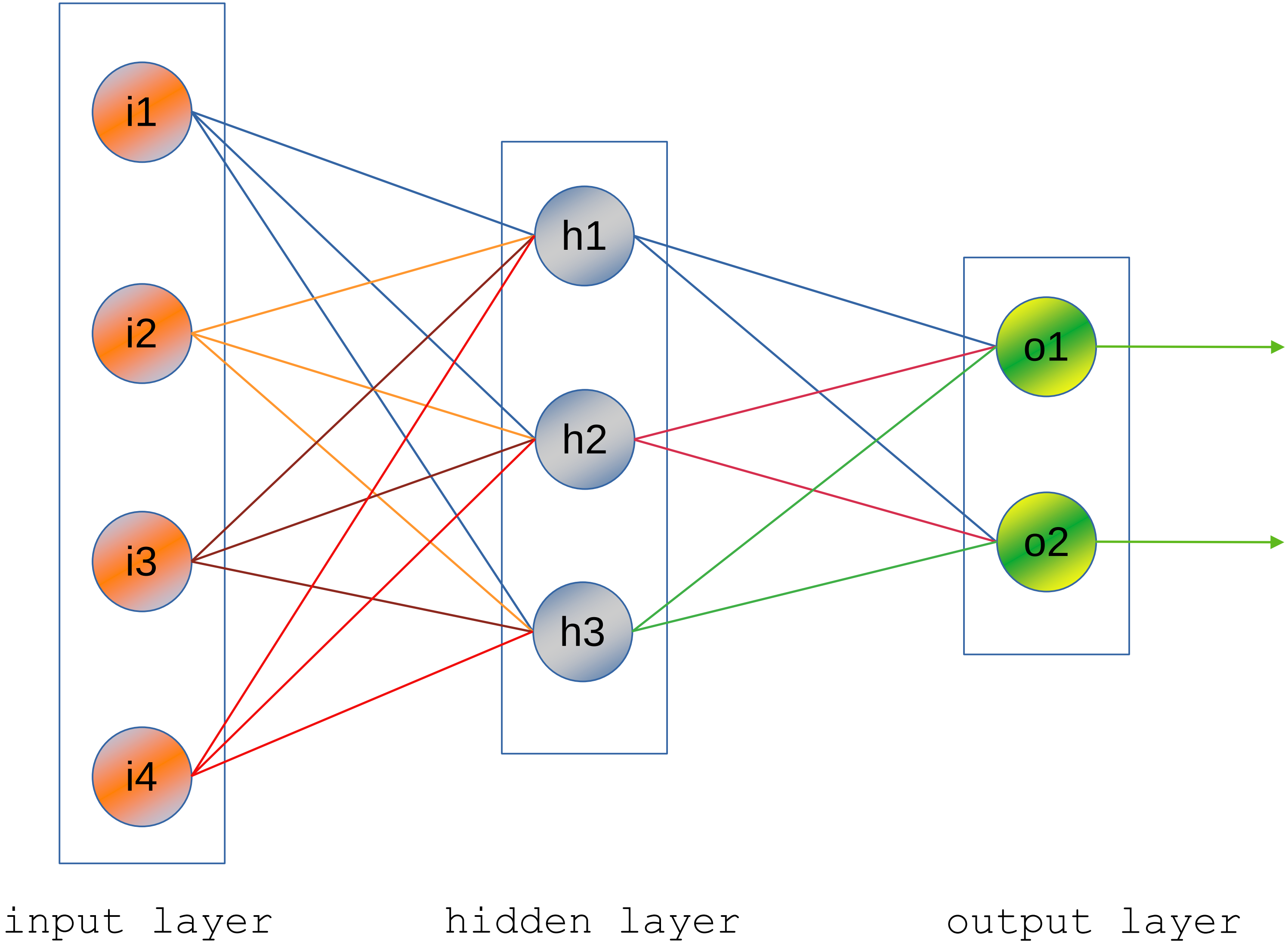


source:wikipedia

artificial neural networks

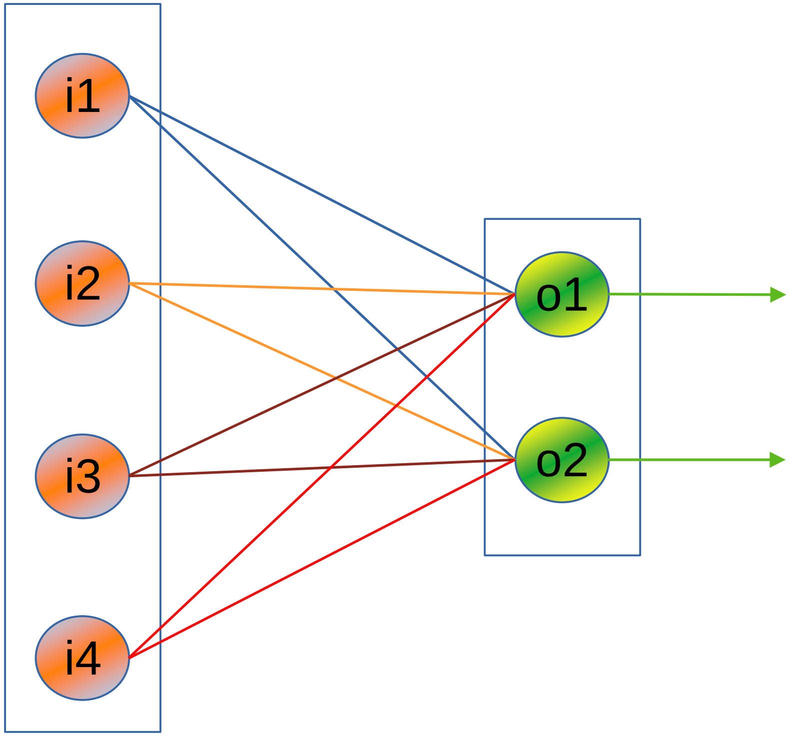
- ANN represents **connections**
- between inputs and outputs
- each connection has a **weight**
- **learning == adjusting these weights**
- to predict the correct output
- applications:
  - classification
  - anomaly detection
  - speech/audio recognition
  - images
  - time series analysis
  - ...

general structure



perceptron

- ANN without hidden layers
- only input and output
- applications:
  - decision making
  - logic gates
  - ...

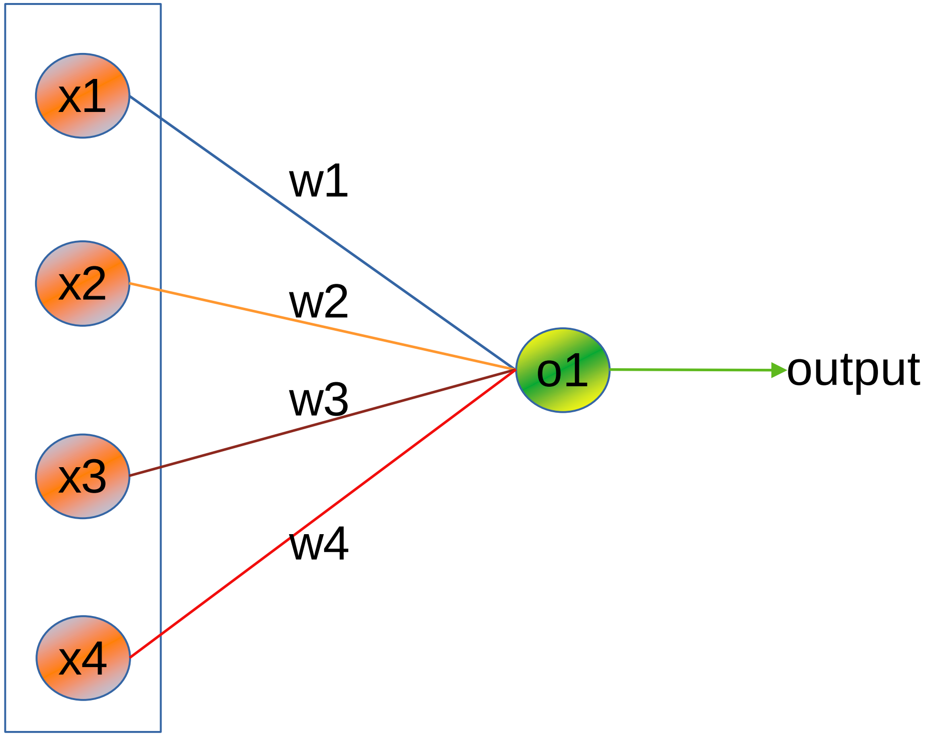


how does it work?

- 2 steps
- Given:
  - a set of input
  - a set of weights (**random!!!**)
- Feed-forward
  - compute output according to weights
- Back-propagation
  - calculate error between predicted and target
  - gradient descent to update the weights

example

- consider the following perceptron



data	target
0	0
1	2
2	4
3	6

$$output = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b$$

example

- b = 0 for simplicity

data	target	output $w_i = 3$	error
0	0	0	0
1	2	3	1
2	4	6	2
3	6	9	3

$$\text{output} = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4$$

example

- errors in 3 out of 4 prediction
  - increase or decrease the weights

data	target	output $w_i = 4$	error
0	0	0	0
1	2	4	2
2	4	8	4
3	6	12	6

$$output = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4$$

example

- errors in 3 out of 4 prediction
  - increase or <u>decrease</u> the weights

data	target	output $w_i = 2$	error
0	0	0	0
1	2	2	0
2	4	4	0
3	6	6	0

$$output = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4$$

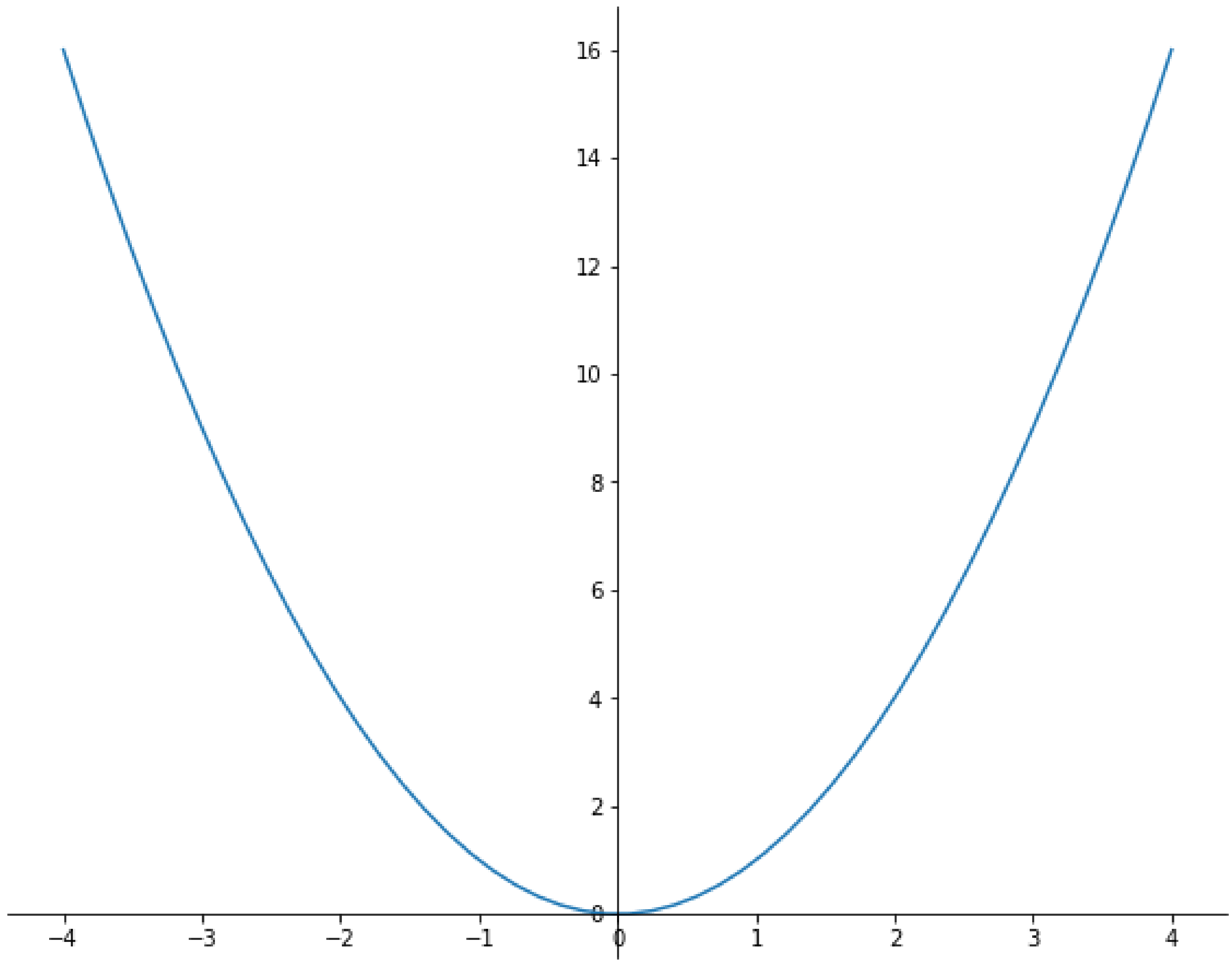
example

- Error minimized
- **Global** minimum

data	target	output $w_i = 2$	error
0	0	0	0
1	2	2	0
2	4	4	0
3	6	6	0

$$output = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4$$

example

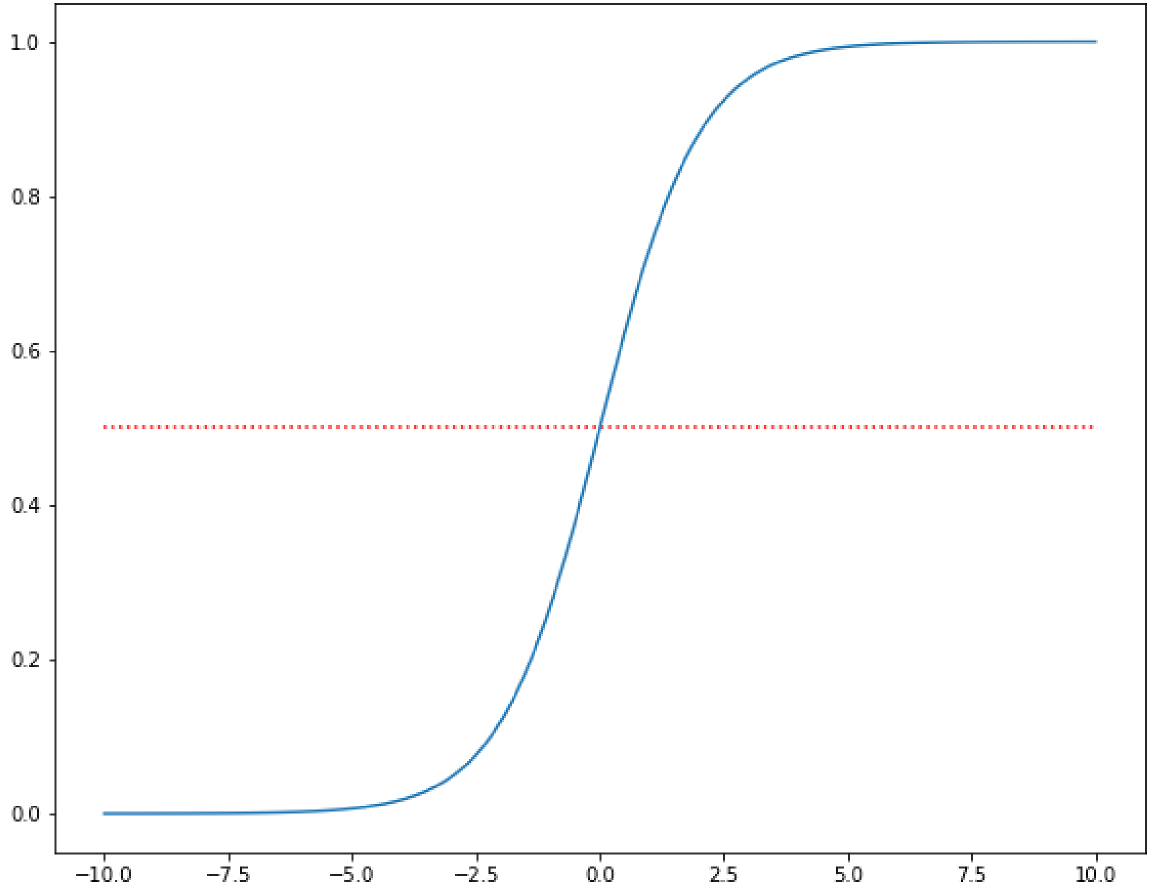


example

- Get the weights so that the error becomes minimum
- Once we figure out if we have to decrease or increase the weights we proceed in the chosen direction
- **STOP** if error increases again
- **GRADIENT DESCENT**

sigmoid function

- activation function in a ANN

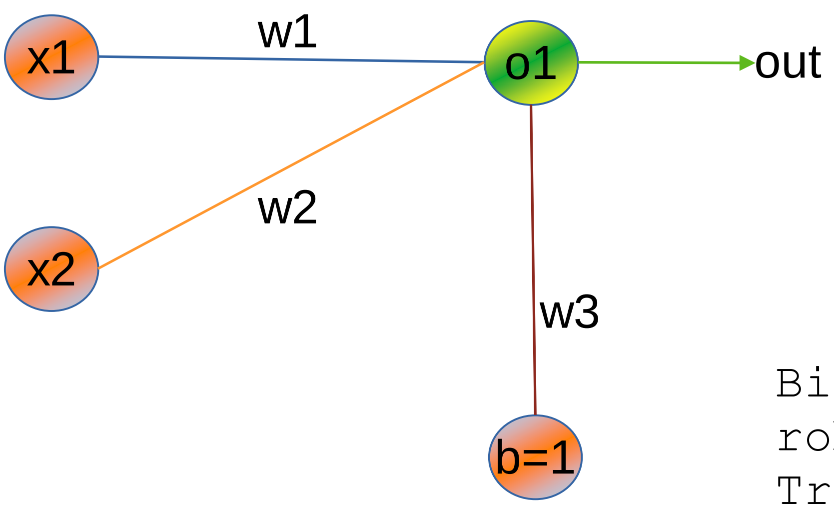


ANN from scratch

- Implementation of a ANN
- **From scratch**
- Pure python
- ANN → **OR GATE**

OR GATE

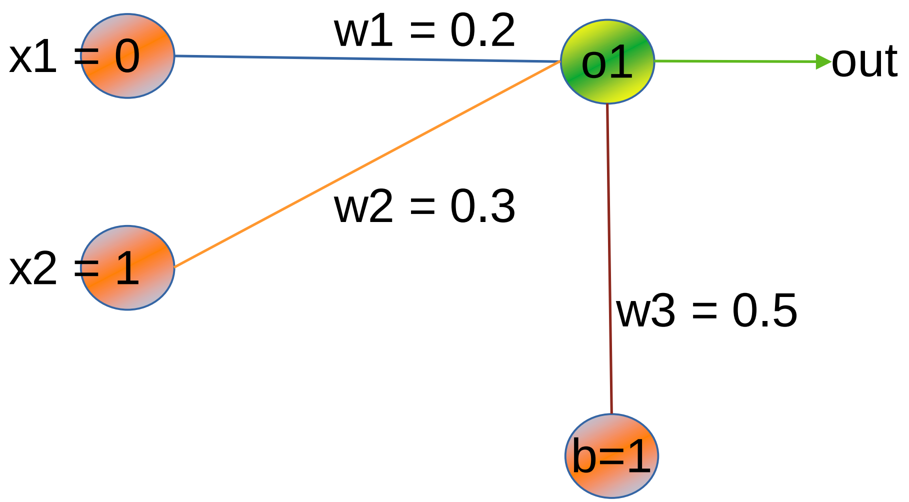
X1	X2	Out
0	0	0
0	1	1
1	0	1
1	1	1



Bias = 1 to make the network more robust. Will be clear at the end. Trust me for now.

OR GATE

- init the weights at random
- calculate input and output (and error)



Input for o1 = w1x1 + w2x2 + w3b = 0.8

out = sigmoid(o1) = 1 / (1 + e^{-o1}) = 0.68997

MSE = SUM(1/2 * (target - output)^2) = 0.0480593

OR GATE

- Have to compute this for all inputs
- Compute global MSE
- Then, update the weights to minimize the error
- → **GRADIENT DESCENT**

Gradient descent

- iterative algorithm
- find optimal values for its parameters
- inputs = parameters + learning rate (lr)

- Loop:
  - start with initial values
  - calculate costs
  - update values using an update function
  - return min costs for cost function

- X = X - lr * f′(X)
- where f′(X) = d/dX f(X)

Gradient descent

- Need to find the derivatives…
- Let’s switch to the notebook

- [LIVE CODING]

OR GATE

- pretty good!

Prediction for (1,0) --> Target value = 1

In [37]:
```
point = np.array([1,0])
res1 = np.dot(point, weights) + bias  # step1
res2 = sigmoid(res1)      # step2
print(res2)
```

```
[0.9793702]
```

Prediction for (1,1) --> Target value = 1

In [38]:
```
point = np.array([1,1])
res1 = np.dot(point, weights) + bias  # step1
res2 = sigmoid(res1)      # step2
print(res2)
```

```
[0.99998097]
```

Prediction for (0,0) --> Target value = 0

In [39]:
```
point = np.array([0,0])
res1 = np.dot(point, weights) + bias  # step1
res2 = sigmoid(res1)      # step2
print(res2)
```

```
[0.04112867]
```

OR GATE

- Why bias?
- Suppose we have input (0,0)
- The sum of the products will always be 0
- **INDEPENDENTLY** of the weights
- Then the result will always be 0
- **INDEPENDENTLY** of how long we train
- Bias affect the shape of the sigmoid function (Live coding)

THE END

- For now.
- Next time: pytorch