Neural Networks

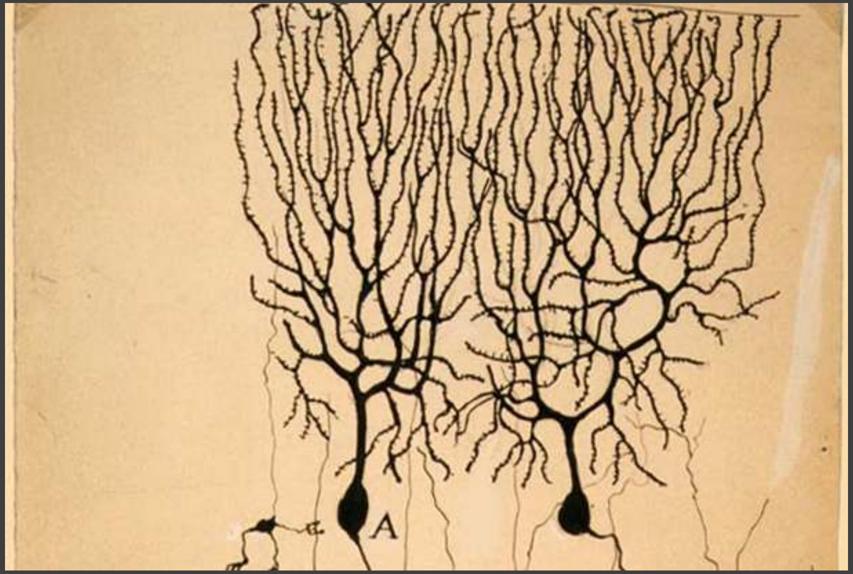
FruitPunch AI Bootcamp

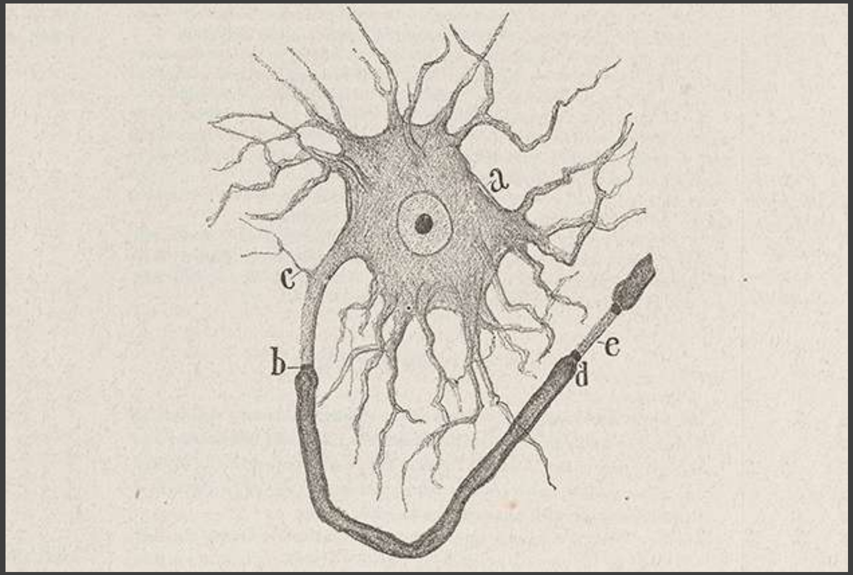
13.12.2020

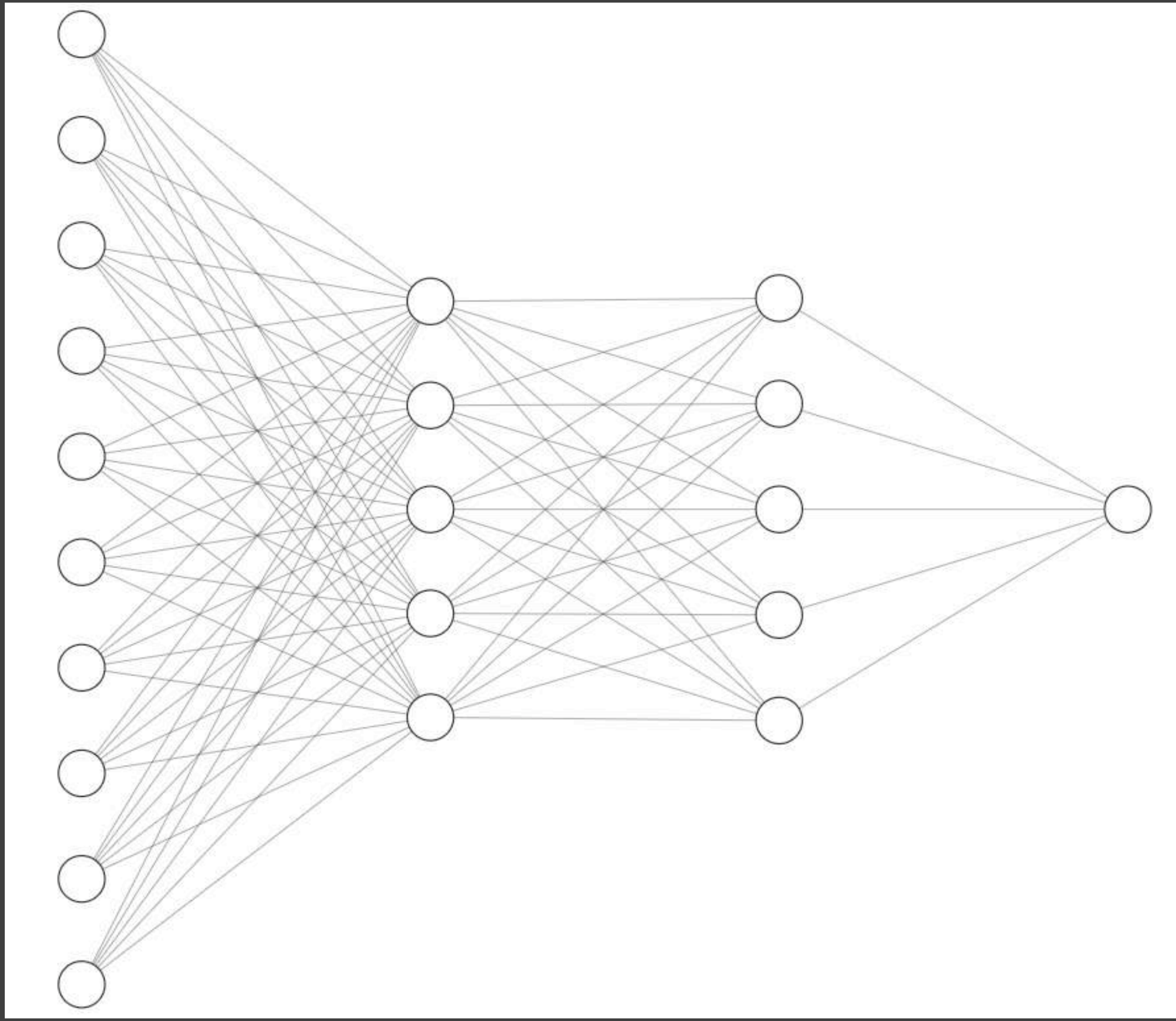
M. Alican Noyan

FruitPunch AI & Ipsumio

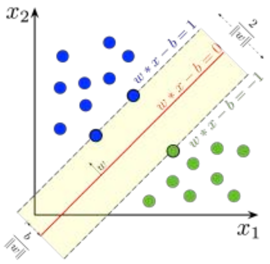
@malicannoyan

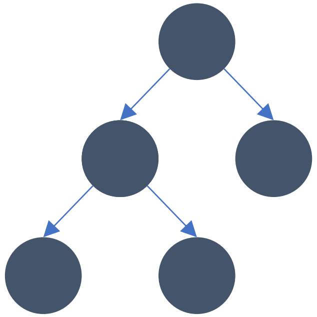

A

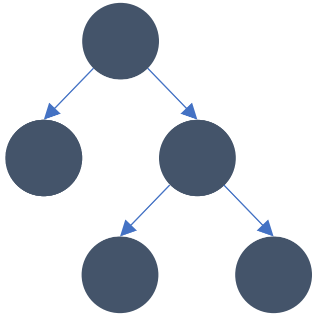

a
c
b
e
d

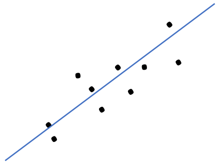


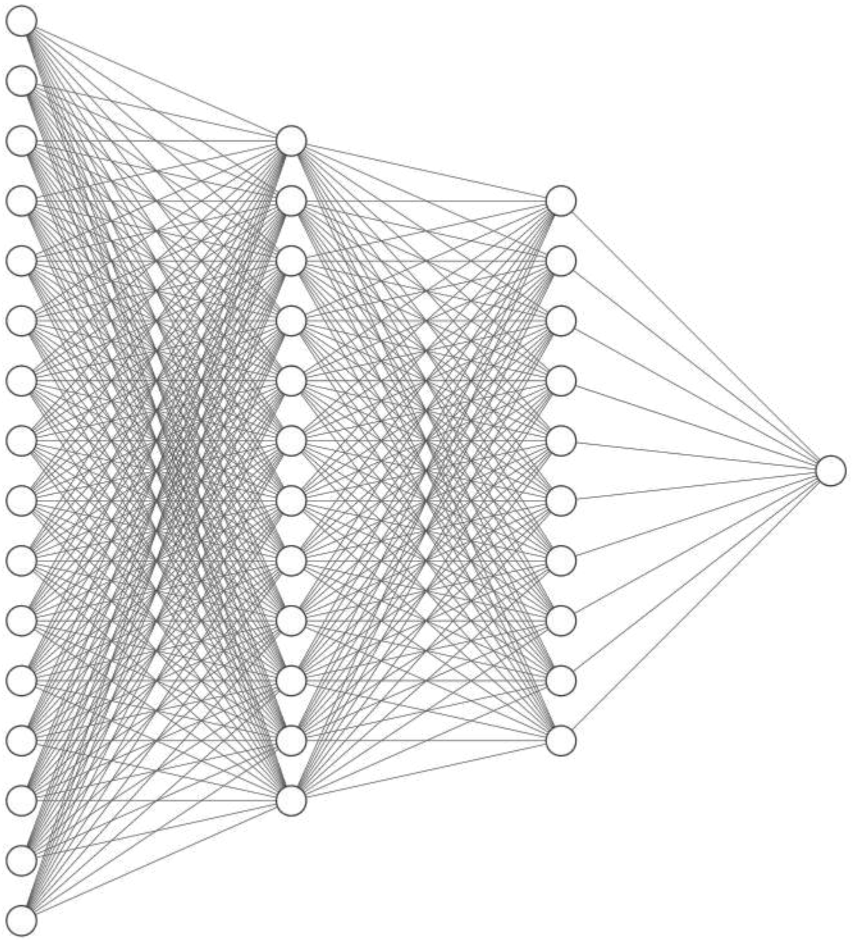
NN vs. other algorithms







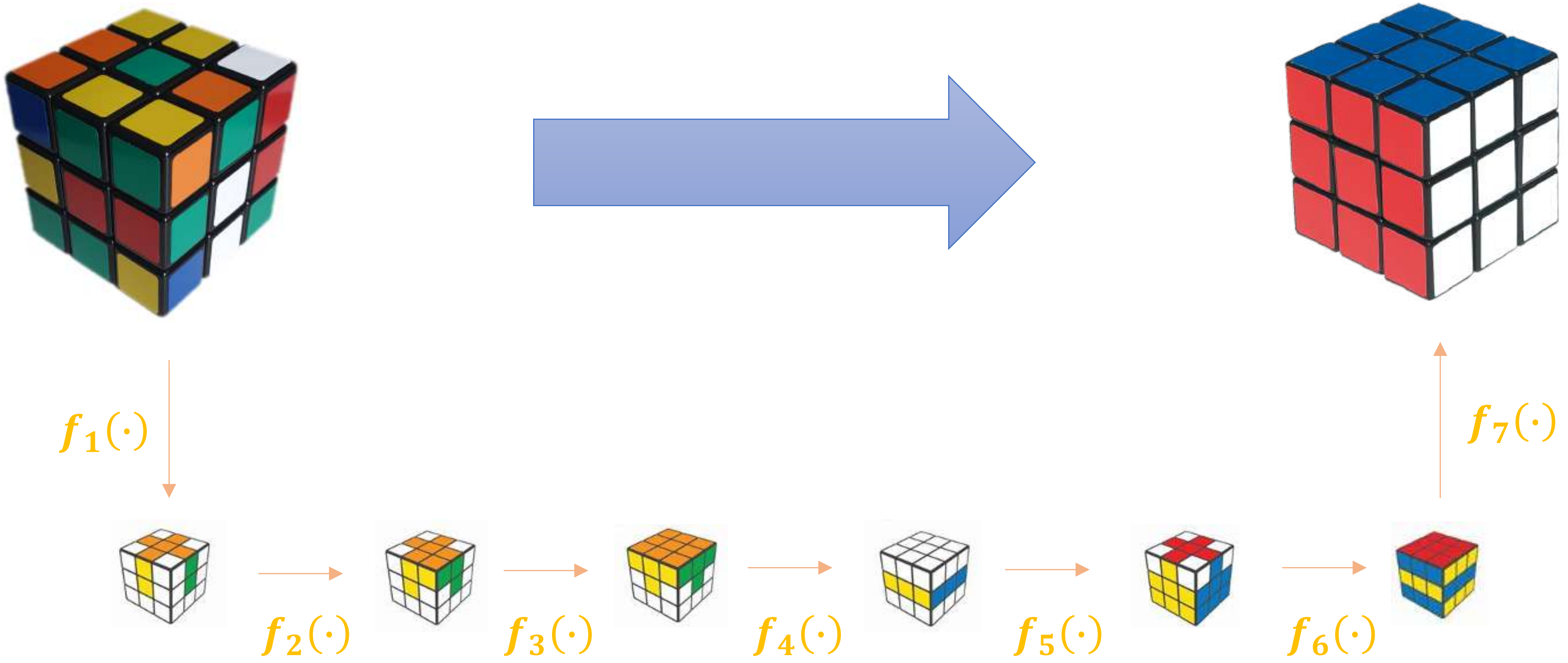




“Astronaut”

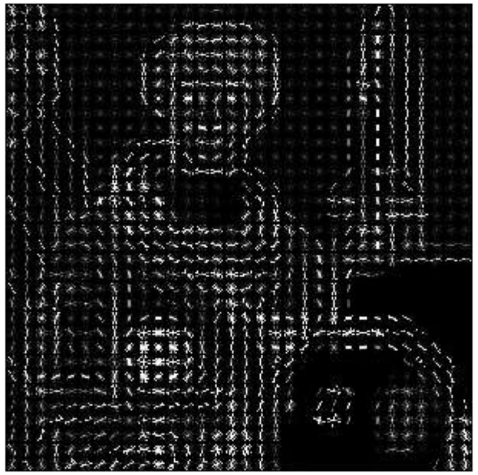
NN vs. other algorithms

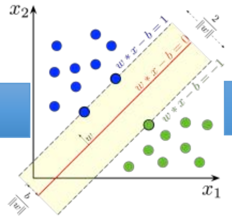
$$f(x) = f_7\left(f_6\left(f_5\left(f_4\left(f_3\left(f_2\left(f_1(x)\right)\right)\right)\right)\right)\right)$$



NN vs. other algorithms

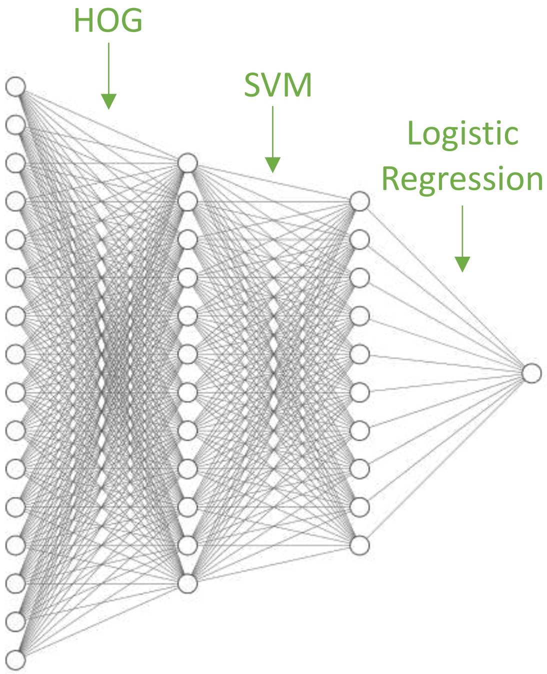
HOG



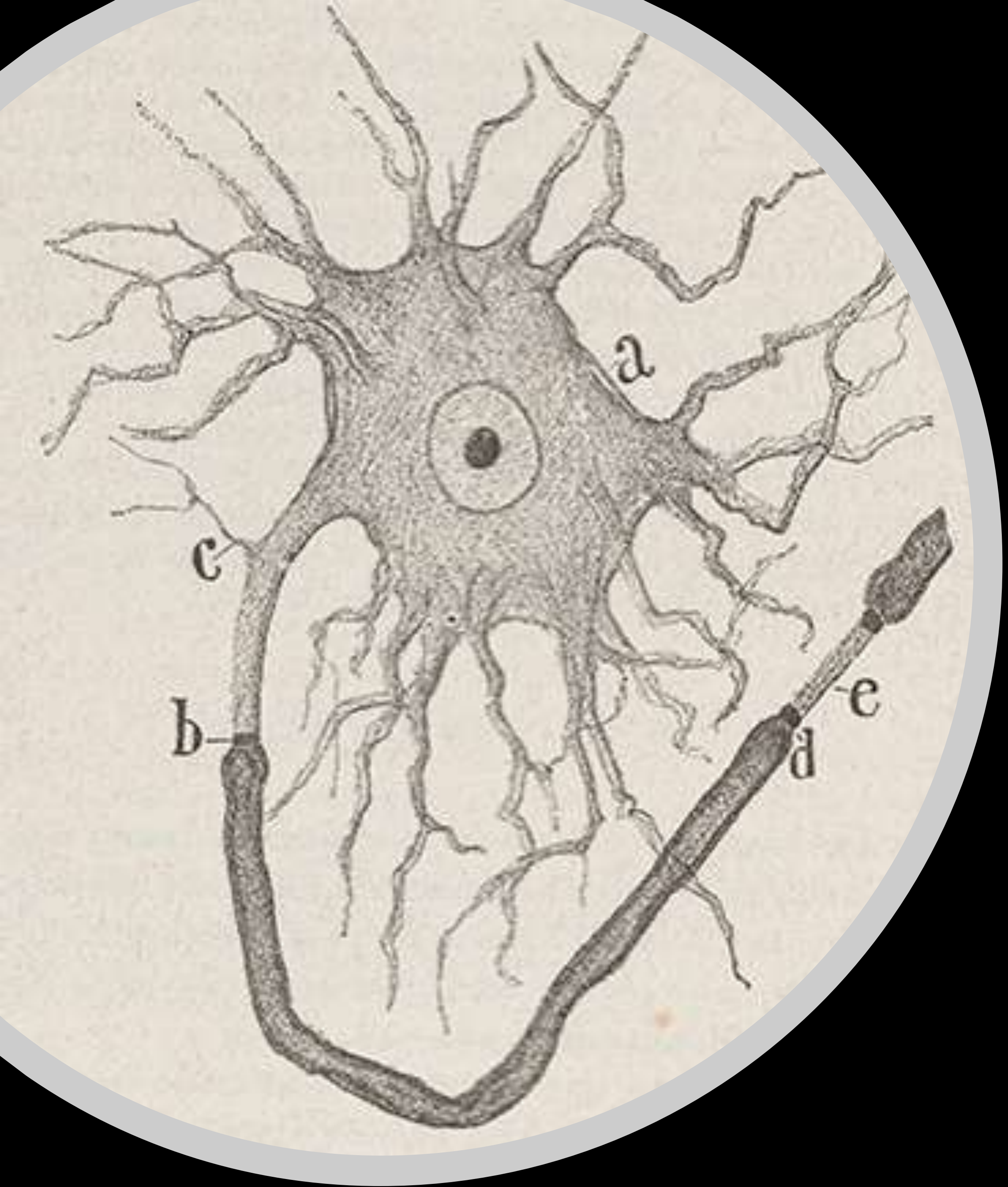


“Astronaut”

“Not an astronaut”

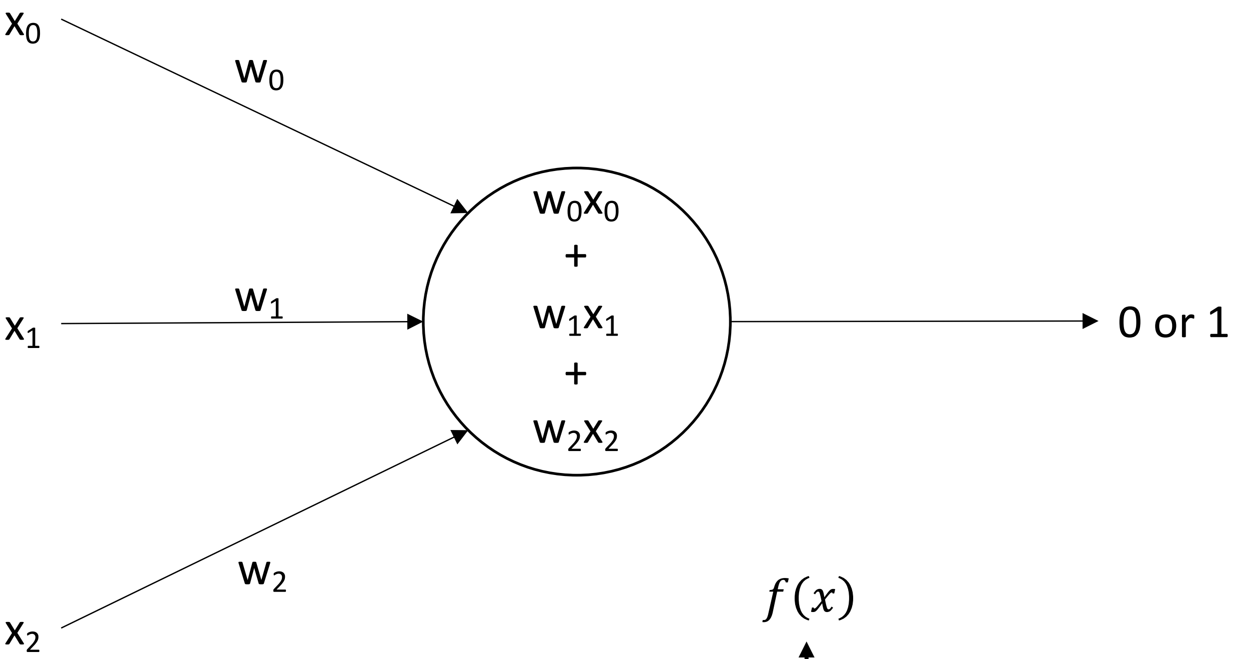


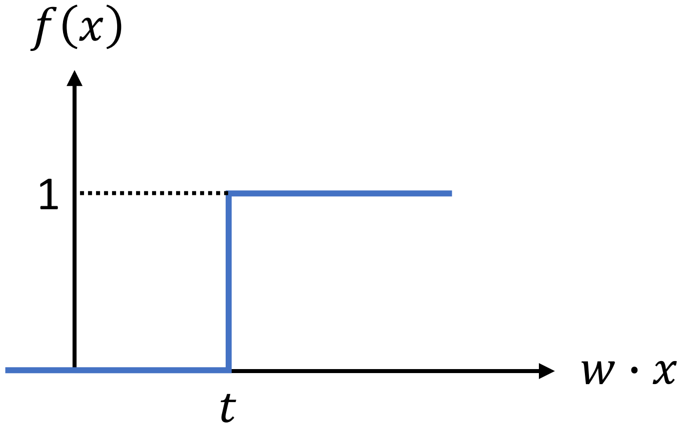
“Astronaut”



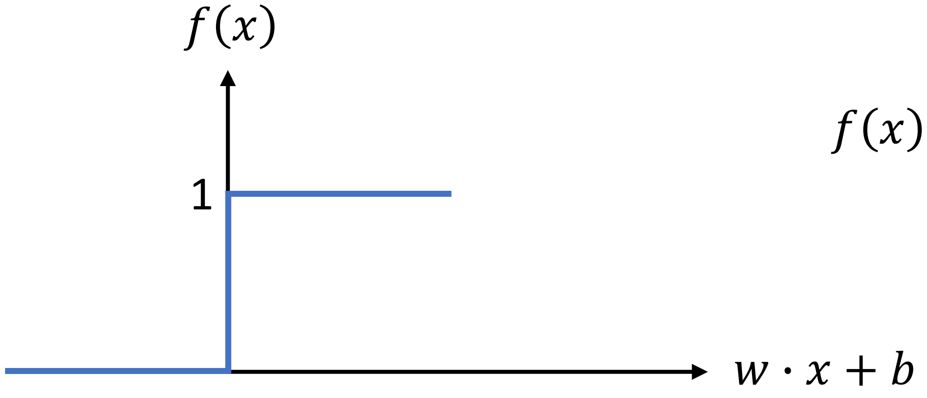
Neurons

Perceptron unit



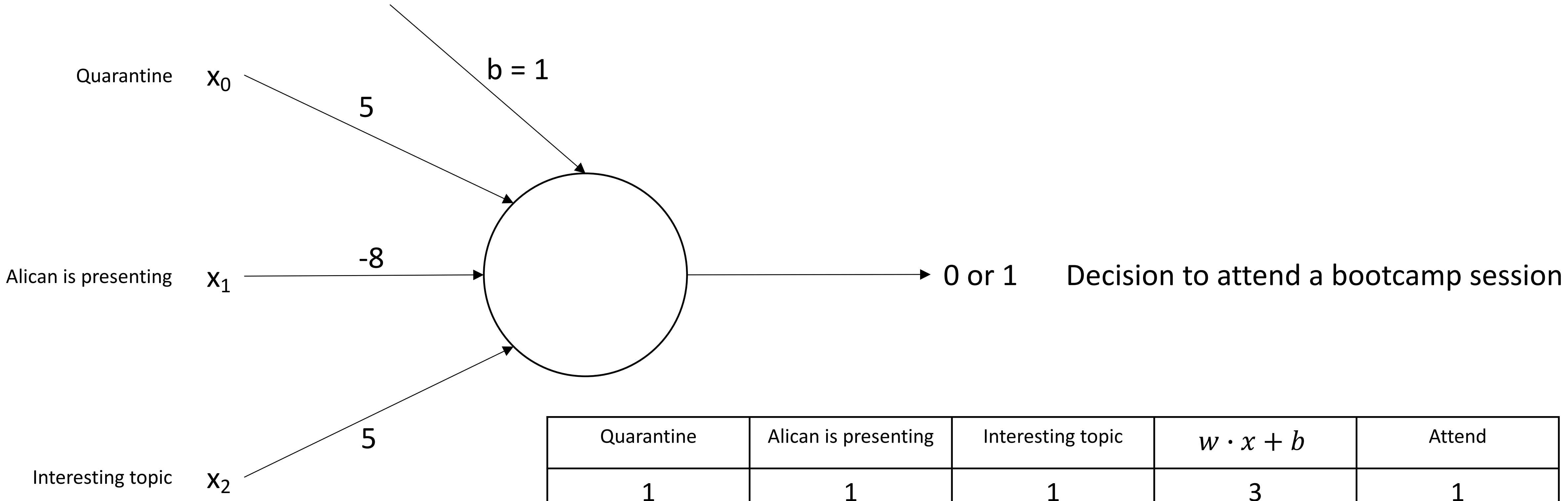


$$f(x) = \begin{cases} 1, & w \cdot x > threshold \\ 0, & w \cdot x \leq threshold \end{cases}$$



$$f(x) = \begin{cases} 1, & w \cdot x + b > 0 \\ 0, & w \cdot x + b \leq 0 \end{cases}$$

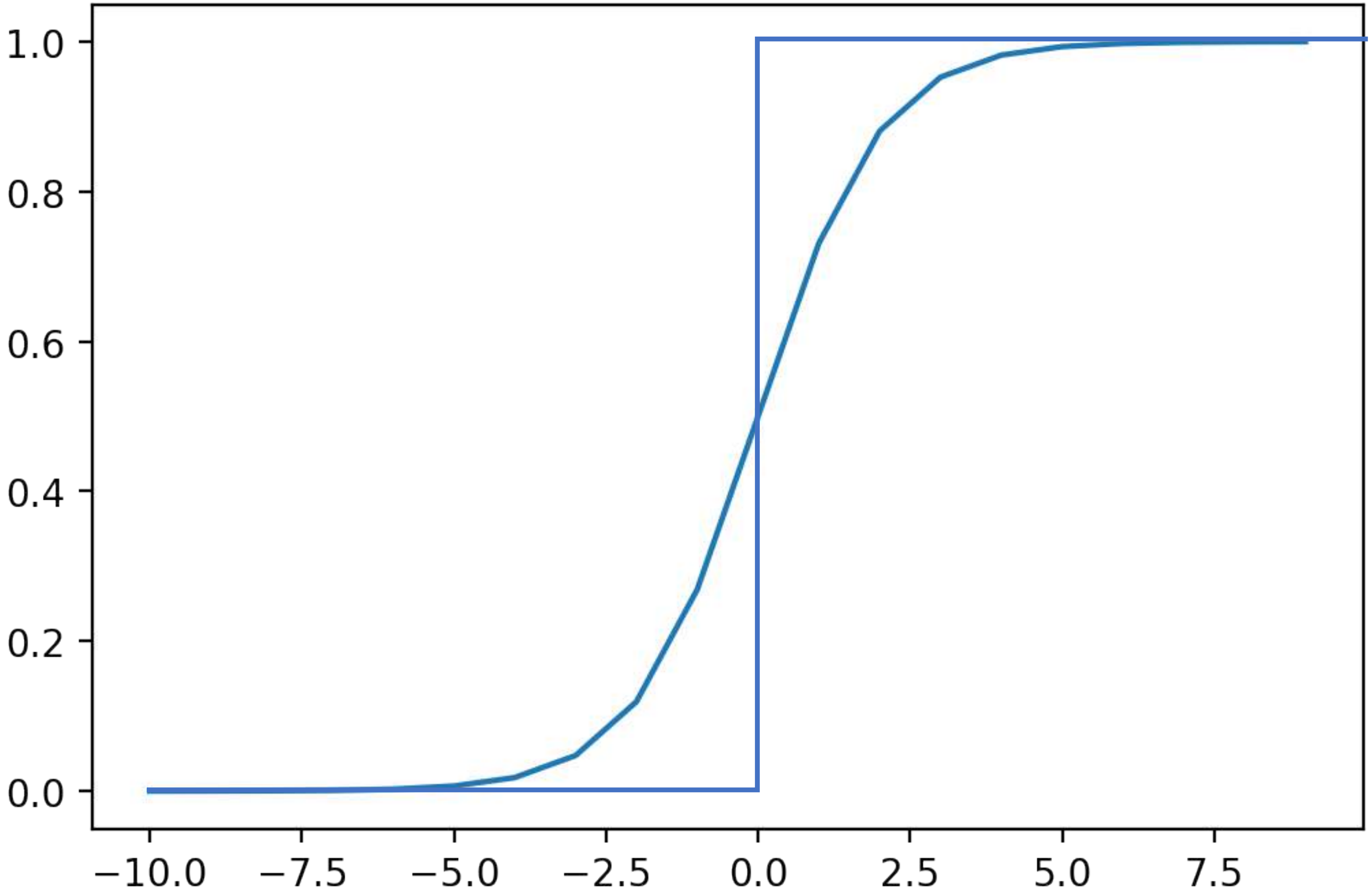
Perceptron unit



Quarantine	Alican is presenting	Interesting topic	$w \cdot x + b$	Attend
1	1	1	3	1
0	1	1	-2	0
1	1	0	-2	0
0	1	0	-7	0
0	0	0	1	1
1	0	1	11	1

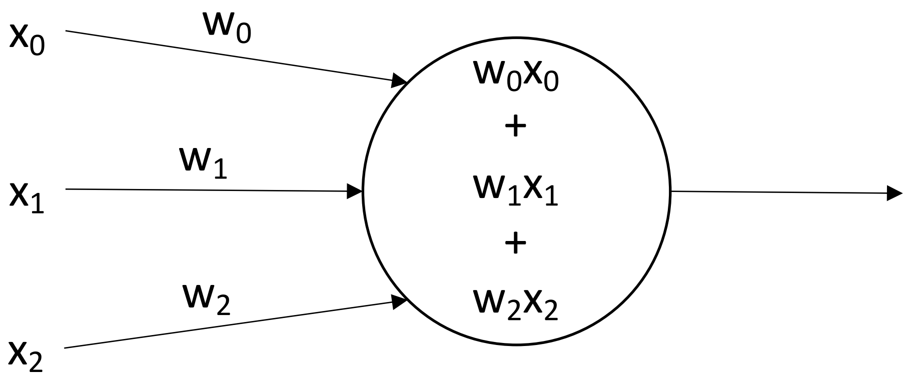
Sigmoid unit

$$\sigma(z) = \frac{1}{1+\ e^{-z}}$$



$$z = w \cdot x + b$$

Perceptron unit

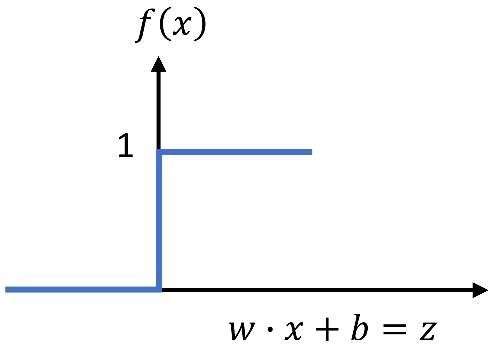


0 or 1

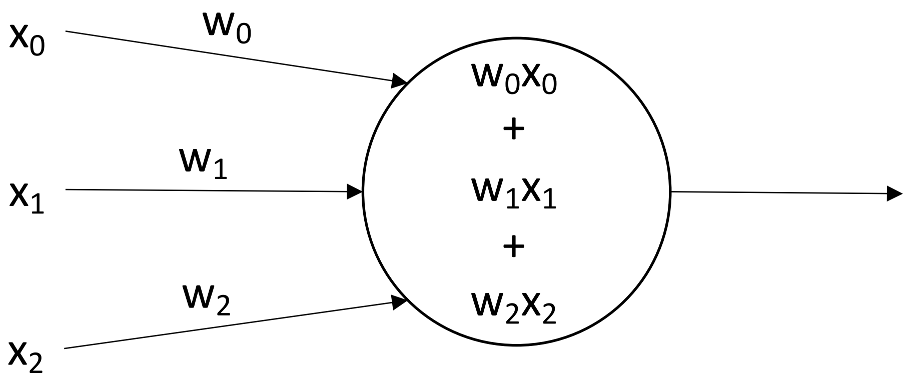
$$w \cdot x + b = z$$

$$f(z) = \begin{cases} 1, & z > 0 \\ 0, & z \leq 0 \end{cases}$$

Activation function



Sigmoid unit

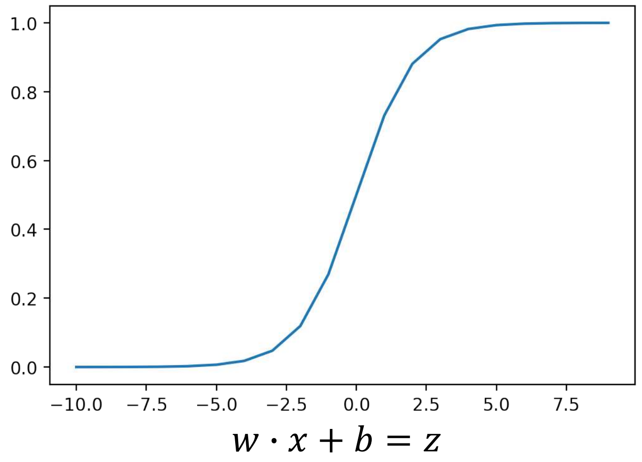


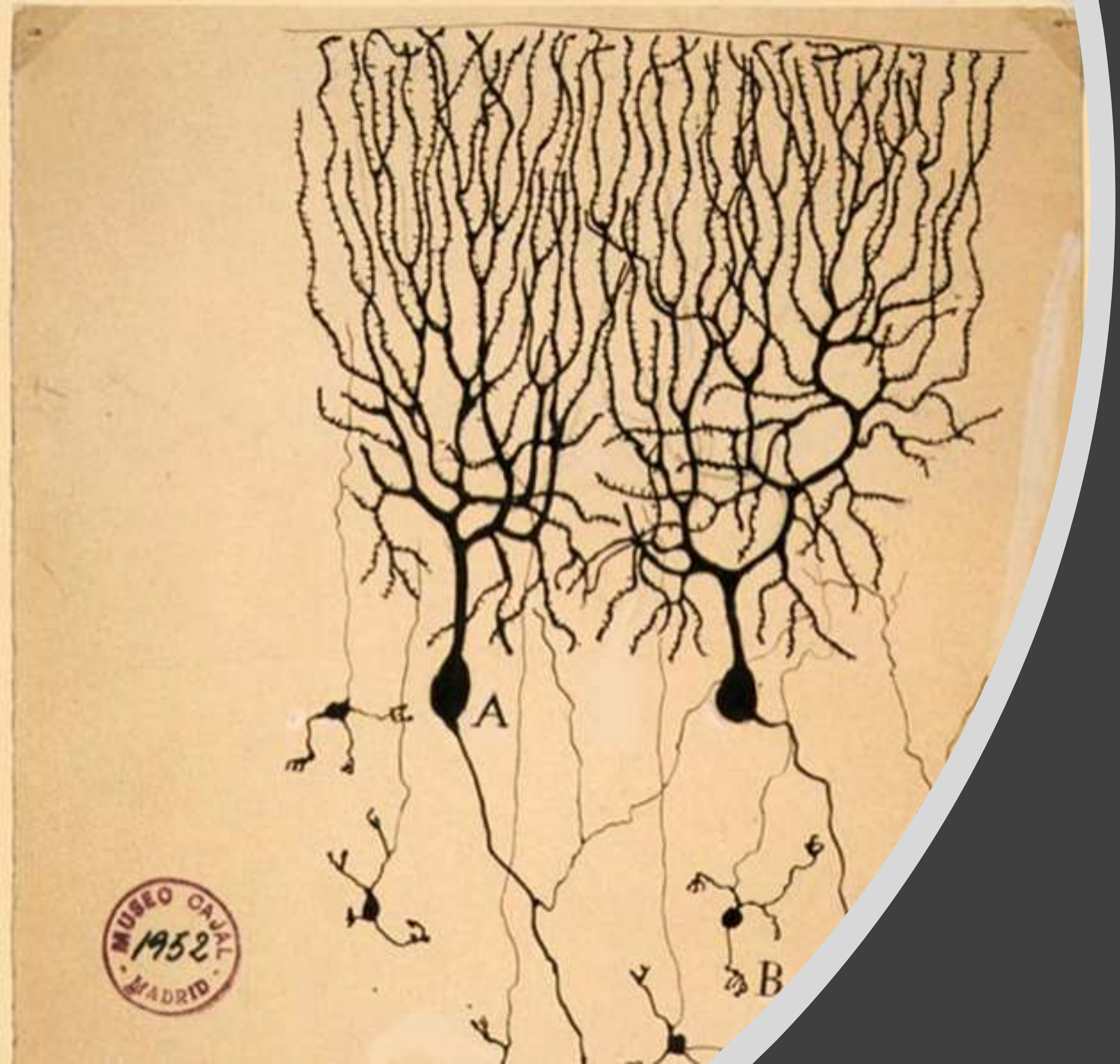
0 - 1

$$w \cdot x + b = z$$

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

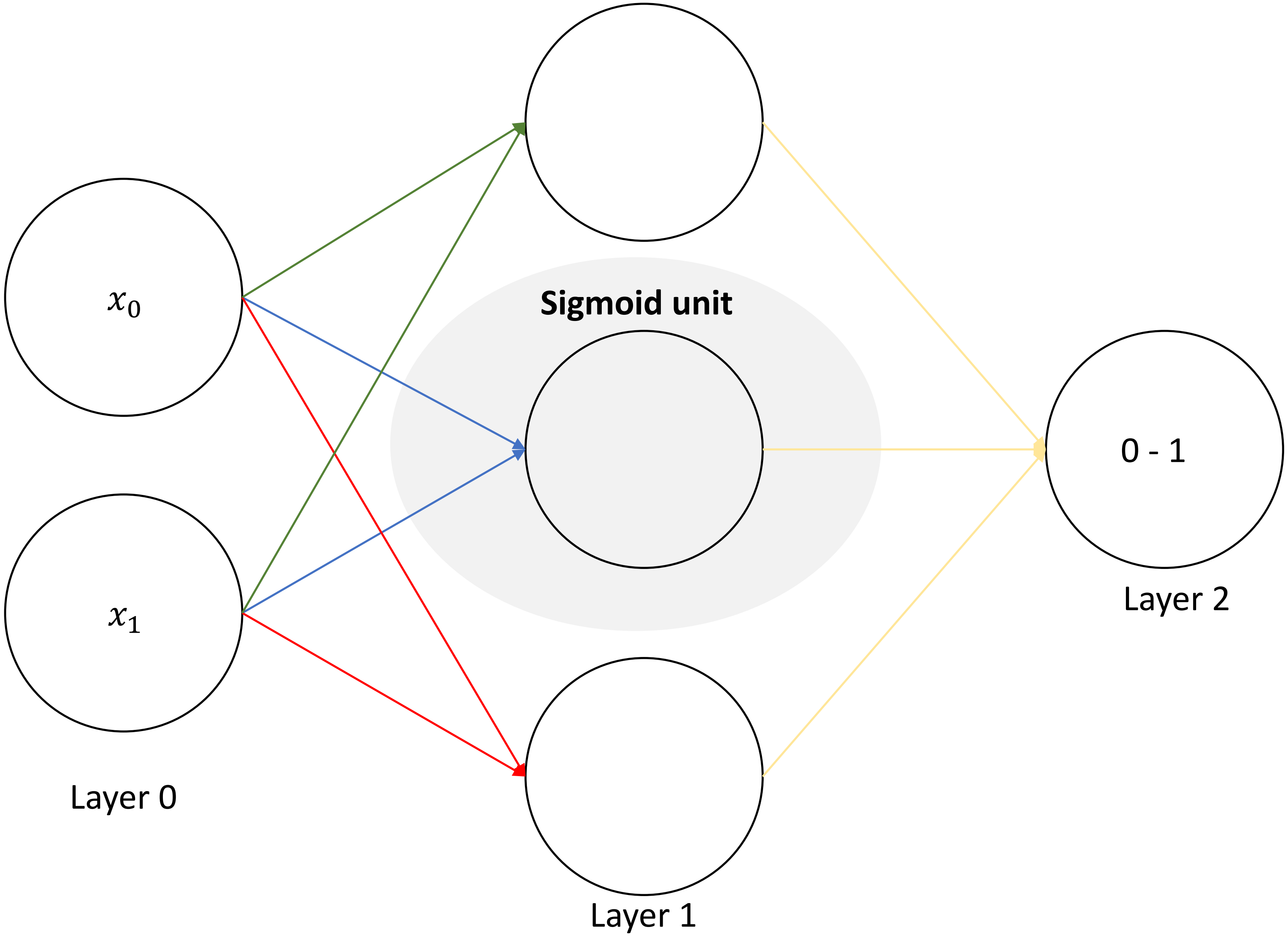
Activation function



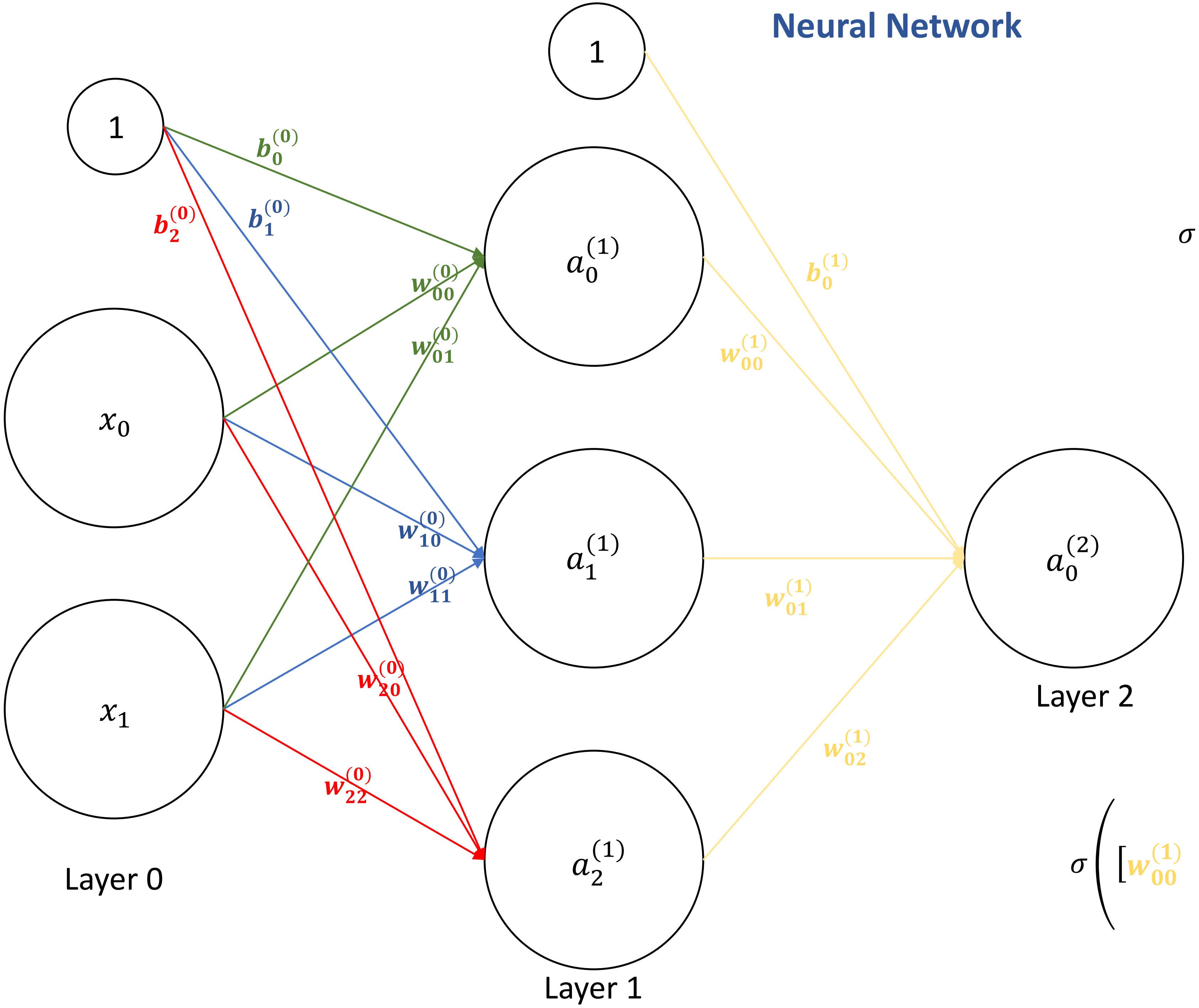


Combining neurons to build neural networks

Neural Network



Neural Network



$$\boldsymbol{w}_{00}^{(0)} x_0 + \boldsymbol{w}_{01}^{(0)} x_1 + \boldsymbol{b}_0^{(0)} = \boldsymbol{z}$$

$$\sigma(z) = a_0^{(1)}$$

$$\sigma\left(\begin{bmatrix} \boldsymbol{w}_{00}^{(0)} & \boldsymbol{w}_{01}^{(0)} \\ \boldsymbol{w}_{10}^{(0)} & \boldsymbol{w}_{11}^{(0)} \\ \boldsymbol{w}_{20}^{(0)} & \boldsymbol{w}_{21}^{(0)} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} + \begin{bmatrix} \boldsymbol{b}_0^{(0)} \\ \boldsymbol{b}_1^{(0)} \\ \boldsymbol{b}_2^{(0)} \end{bmatrix}\right) = \begin{bmatrix} a_0^{(1)} \\ a_1^{(1)} \\ a_2^{(1)} \end{bmatrix}$$

$$\sigma\left(w^{(0)} x + b^{(0)}\right) = a^{(1)}$$

Forward pass

$$w = \left[w^{(0)}, w^{(1)}\right] \qquad b = \left[b^{(0)}, b^{(1)}\right]$$

input

$$\sigma\left(w^{(0)} x + b^{(0)}\right) = a^{(1)}$$

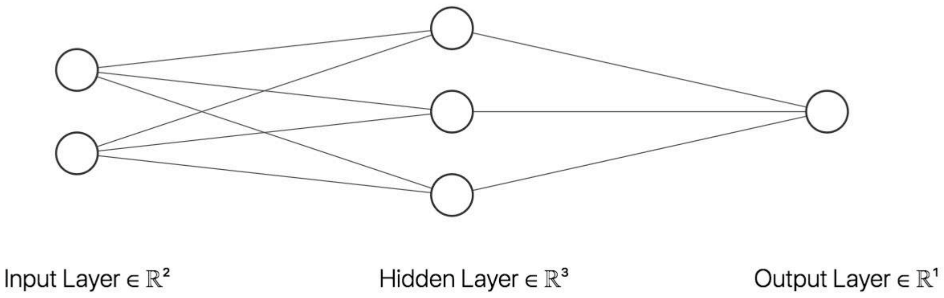
output

$$\sigma\left(w^{(1)} a^{(1)} + b^{(1)}\right) = a^{(2)}$$

$$\sigma\left(\begin{bmatrix} \boldsymbol{w}_{00}^{(1)} & \boldsymbol{w}_{01}^{(1)} & \boldsymbol{w}_{02}^{(1)} \end{bmatrix} \begin{bmatrix} a_0^{(1)} \\ a_1^{(1)} \\ a_2^{(1)} \end{bmatrix} + \begin{bmatrix} \boldsymbol{b}_0^{(1)} \end{bmatrix}\right) = \begin{bmatrix} a_0^{(2)} \end{bmatrix}$$

$$\sigma\left(w^{(1)} a^{(1)} + b^{(1)}\right) = a^{(2)}$$

Challenge 1 – Given any size NN, initialize weights and biases




```python
w, b = nn_parameter_maker([2, 3, 1])
```

$w = [w^{(0)}, w^{(1)}]$ $\quad b = [b^{(0)}, b^{(1)}]$

```python
w[0], w[0].shape
```

```
(array([[0.32576672, 0.63369041],
        [0.24036744, 0.4580126 ],
        [1.25499253, 0.39859241]]), (3, 2))
```

```python
w[1], w[1].shape
```

```
(array([[2.41678702, 0.32569961, 0.41721439]]), (1, 3))
```

```python
b[0], b[0].shape
```

```
(array([[-0.15655603],
        [ 1.63128929],
        [ 0.56478083]]), (3, 1))
```

```python
b[1], b[1].shape
```

```
(array([[-0.58297318]]), (1, 1))
```

$$w = \begin{bmatrix} w_{00}^{(0)} & w_{01}^{(0)} \\ w_{10}^{(0)} & w_{11}^{(0)} \\ w_{20}^{(0)} & w_{21}^{(0)} \end{bmatrix}, \begin{bmatrix} w_{00}^{(1)} & w_{01}^{(1)} & w_{02}^{(1)} \end{bmatrix}$$

$$b = \begin{bmatrix} b_0^{(0)} \\ b_1^{(0)} \\ b_2^{(0)} \end{bmatrix}, \begin{bmatrix} b_0^{(1)} \end{bmatrix}$$

```python
# Challenge 2
def sigmoid(z):

    return
```

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Forward pass

$w = [w^{(0)}, w^{(1)}]$ $\quad b = [b^{(0)}, b^{(1)}]$

input

$$\sigma(w^{(0)}x + b^{(0)}) = a^{(1)}$$

output

$$\sigma(w^{(1)}a^{(1)} + b^{(1)}) = a^{(2)}$$

```python
# Challenge 3
def forward_pass(inp, w, b):

    return out
```

input $\longrightarrow$ output

Forward pass

Side note

$$\begin{array}{cc} x_0 & x_1 \end{array}$$

$$X = \begin{bmatrix} & \\ \vdots & \vdots \end{bmatrix} \qquad y = \begin{bmatrix} \\ \vdots \end{bmatrix}$$

X^T

$$\sigma\left(\begin{bmatrix} w_{00}^{(0)} & w_{01}^{(0)} \\ w_{10}^{(0)} & w_{11}^{(0)} \\ w_{20}^{(0)} & w_{21}^{(0)} \end{bmatrix}\begin{bmatrix} x_0 \cdots \\ x_1 \cdots \end{bmatrix} + \begin{bmatrix} b_0^{(0)} \\ b_1^{(0)} \\ b_2^{(0)} \end{bmatrix}\right) = \begin{bmatrix} a_0^{(1)} \\ a_1^{(1)} \\ a_2^{(1)} \end{bmatrix}$$

y^T

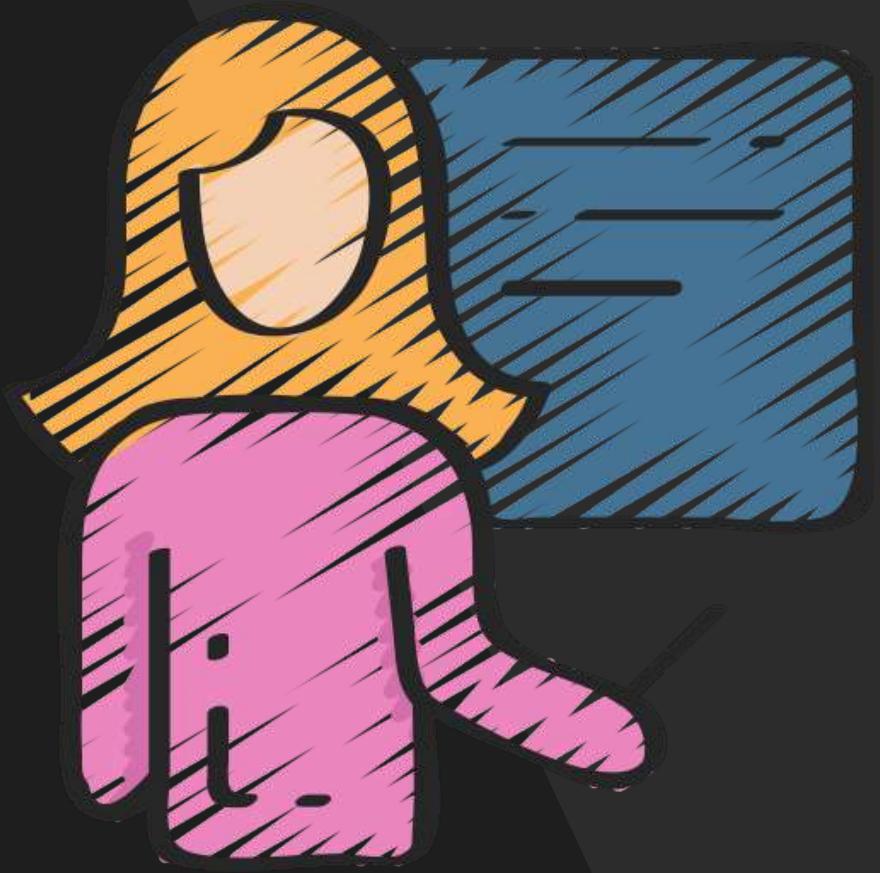
$$\sigma\left(\begin{bmatrix} w_{00}^{(1)} & w_{01}^{(1)} & w_{02}^{(1)} \end{bmatrix}\begin{bmatrix} a_0^{(1)} \\ a_1^{(1)} \\ a_2^{(1)} \end{bmatrix} + \begin{bmatrix} b_0^{(1)} \end{bmatrix}\right) = \begin{bmatrix} a_0^{(2)} \cdots \end{bmatrix}$$

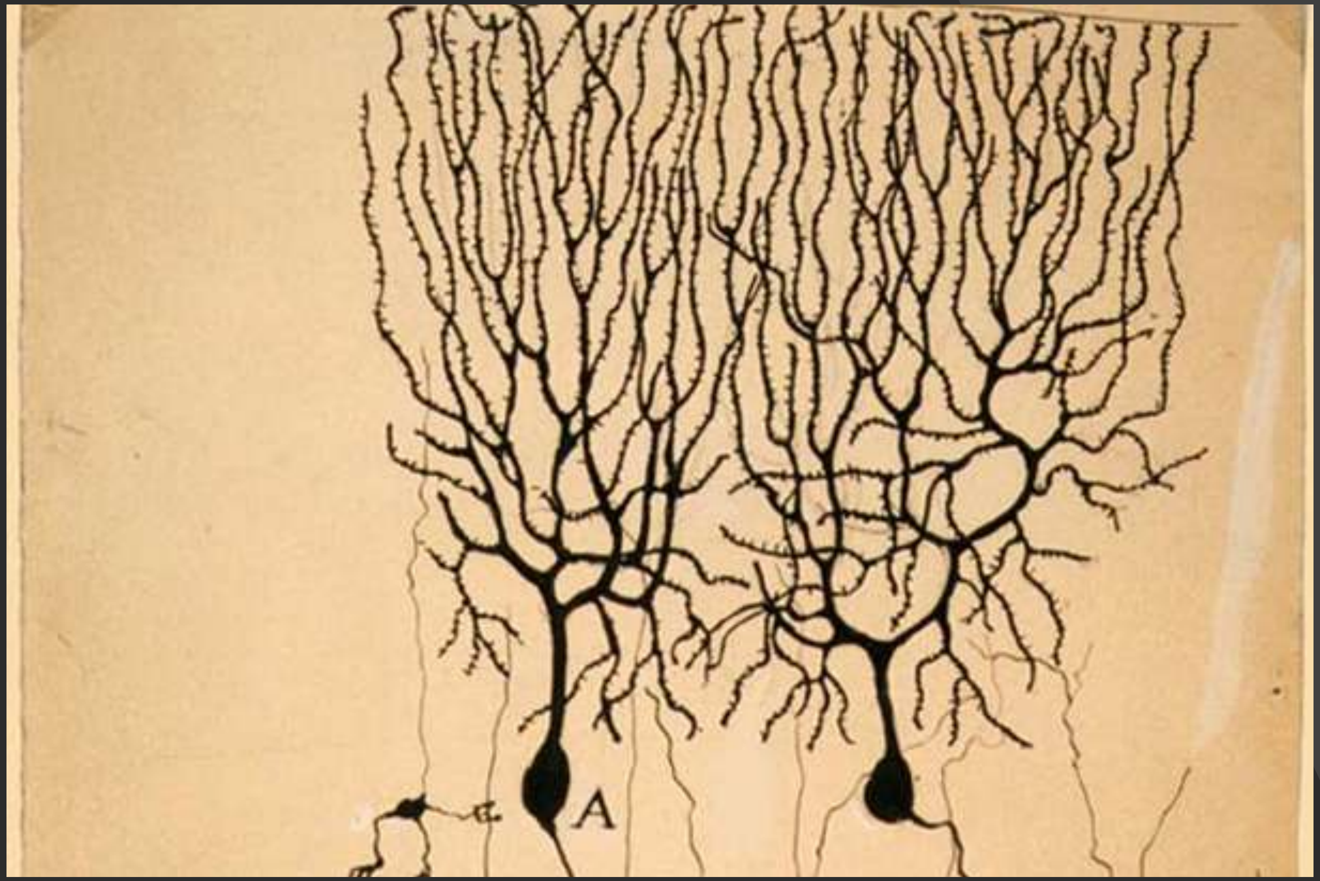
```python
def predict(X, w, b):
    X_t = np.transpose(X)
    y_pred_t = forward_pass(X_t, w, b)
    y_pred = np.transpose(y_pred_t)
    return y_pred
```

```python
# Challenge 3
def forward_pass(inp, w, b):

    return out
```

Training neural networks





Gradient Descent

$$w_{jk}^{(i)} = w_{jk}^{(i)} - learning\_rate \frac{\partial cost\_function}{\partial w_{jk}^{(i)}}$$

$$b_{jk}^{(i)} = b_{jk}^{(i)} - learning\_rate \frac{\partial cost\_function}{\partial b_{jk}^{(i)}}$$

```python
def cost_function(X, y, w, b):
    '''
    Given
    - input
    - network parameters
    - labels

    calculates the mse
    '''

    y_pred = predict(X, w, b)
    m = y.shape[0]
    cost = np.sum((y_pred-y)**2)/m   # mse
    return cost
```

$$\frac{\partial cost\_function}{\partial w_{11}^{(0)}}$$

$$w = \begin{bmatrix} w_{00}^{(0)} & w_{01}^{(0)} \\ w_{10}^{(0)} & w_{11}^{(0)} \\ w_{20}^{(0)} & w_{21}^{(0)} \end{bmatrix}, \begin{bmatrix} w_{00}^{(1)} & w_{01}^{(1)} & w_{02}^{(1)} \end{bmatrix} \qquad w^{+} = \begin{bmatrix} w_{00}^{(0)} & w_{01}^{(0)} \\ w_{10}^{(0)} & w_{11}^{(0)} + eps \\ w_{20}^{(0)} & w_{21}^{(0)} \end{bmatrix}, \begin{bmatrix} w_{00}^{(1)} & w_{01}^{(1)} & w_{02}^{(1)} \end{bmatrix}$$

```python
# Implemented
def w_update(w, layer_id, i, j, new_param):↔
```

```python
# Implemented
def b_update(b, layer_id, i, j, new_param):↔
```

$$\frac{\partial cost\_function}{\partial w_{11}^{(0)}} \approx \frac{cost\_function(w^{+}, b) - cost\_function(w, b)}{epsilon}$$

9 weights, 4 biases, 13 parameters — 13 forward pass

Backpropagation — 1 forward 1 backward pass — 13 param or 1m param

Estimate gradient and do gradient descent

$$w = \begin{bmatrix} w_{00}^{(0)} & w_{01}^{(0)} \\ w_{10}^{(0)} & w_{11}^{(0)} \\ w_{20}^{(0)} & w_{21}^{(0)} \end{bmatrix}, \begin{bmatrix} w_{00}^{(1)} & w_{01}^{(1)} & w_{02}^{(1)} \end{bmatrix}$$

$$b = \begin{bmatrix} b_0^{(0)} \\ b_1^{(0)} \\ b_2^{(0)} \end{bmatrix}, \begin{bmatrix} b_0^{(1)} \end{bmatrix}$$

$$\frac{\partial cost\_function}{\partial w_{11}^{(0)}} \approx \frac{cost\_function(w^+, b) - cost\_function(w, b)}{epsilon}$$

$$w\_pds = \begin{bmatrix} \frac{\partial J}{\partial w_{00}^{(0)}} & \frac{\partial J}{\partial w_{01}^{(0)}} \\ \frac{\partial J}{\partial w_{10}^{(0)}} & \frac{\partial J}{\partial w_{11}^{(0)}} \\ \frac{\partial J}{\partial w_{20}^{(0)}} & \frac{\partial J}{\partial w_{21}^{(0)}} \end{bmatrix}, \begin{bmatrix} \frac{\partial J}{\partial w_{00}^{(1)}} & \frac{\partial J}{\partial w_{01}^{(1)}} & \frac{\partial J}{\partial w_{02}^{(1)}} \end{bmatrix}$$

$$b\_pds = \begin{bmatrix} \frac{\partial J}{\partial b_0^{(0)}} \\ \frac{\partial J}{\partial b_1^{(0)}} \\ \frac{\partial J}{\partial b_2^{(0)}} \end{bmatrix}, \begin{bmatrix} \frac{\partial J}{\partial b_0^{(1)}} \end{bmatrix}$$

```python
# Challenge 4
def gradient_estimator(X, y, w, b):
    eps = 1e-4

    return w_pds, b_pds
```

$$w_{jk}^{(i)} = w_{jk}^{(i)} - learning\_rate \frac{\partial cost\_function}{\partial w_{jk}^{(i)}}$$

$$b_{jk}^{(i)} = b_{jk}^{(i)} - learning\_rate \frac{\partial cost\_function}{\partial b_{jk}^{(i)}}$$

```python
# Challenge 5
def one_step_gd(X, y, w, b, lr):
    w_pds, b_pds = gradient_estimator(X, y, w, b)

    return new_w, new_b
```

Expected behavior from gradient descent

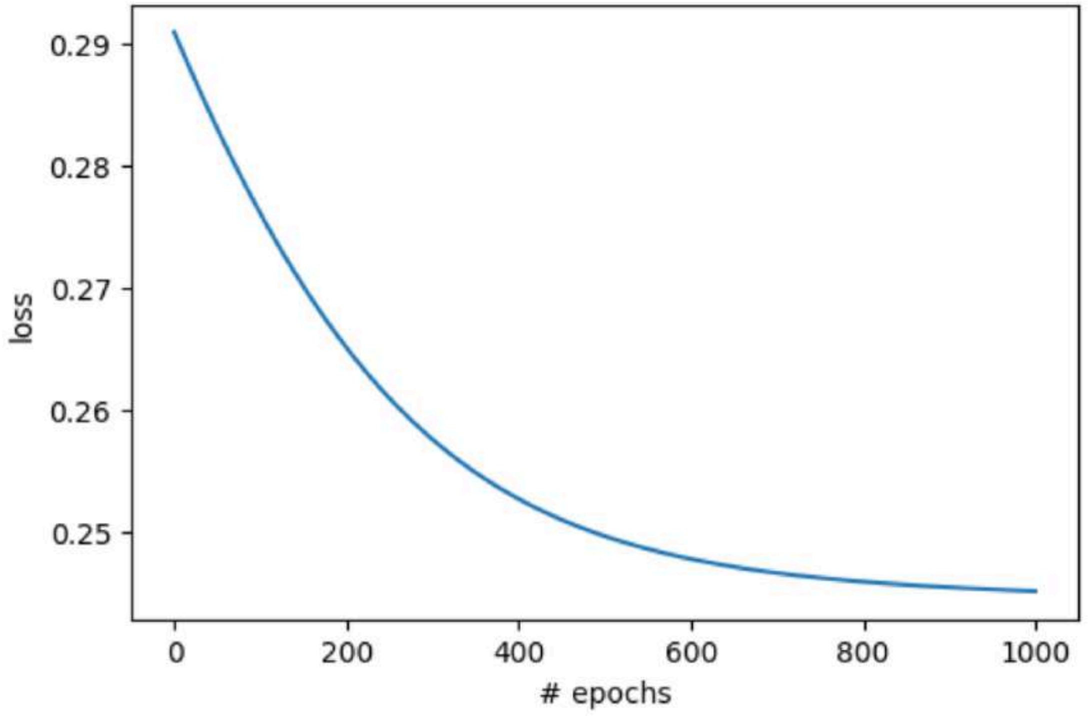
```python
# Implemented
def GD(X, y, w, b, epoch, lr):
    errors = []

    for i in range(epoch):
        w, b = one_step_gd(X, y, w, b, lr)
        error = cost_function(X, y, w, b)
        errors.append(error)

    plt.plot(errors)

    return w, b
```

```python
w, b = GD(X, y, w, b, epoch=1000, lr=0.01)
```

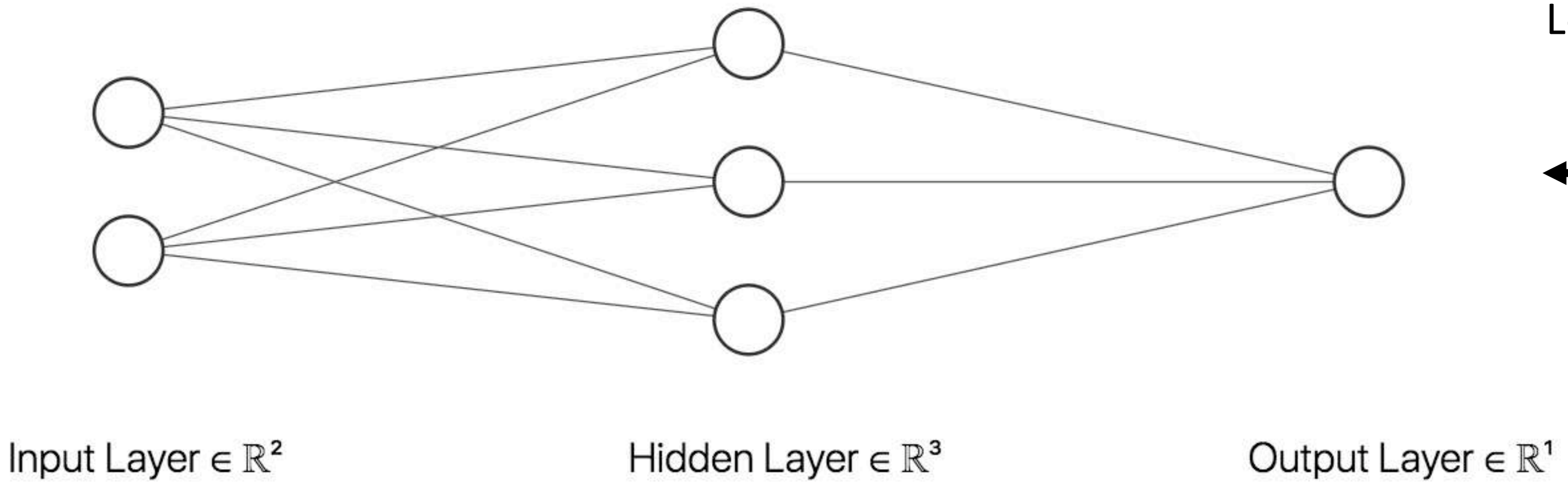


Overview

Overview

Random initialization

Weights, biases, activation function



Loss function

Labels

Optimizer

Learning rate

Further questions to expand your knowledge

How to do multilabel/multiclass classification?

Different loss/activation functions?

Advantages and disadvantages of adding more layers?

Are there better initialization techniques?

Gradient descent variants?

Regularization?

How to do regression?

Different architectures for certain problems? Why CNN RNN?

Vanishing/Exploding gradients?

How did people achieve developing 100-1000 layers networks?

Backpropagation?

Data augmentation?

An informed look into first example

```python
# Build the architecture
model = Sequential()
model.add(Dense(30, input_dim=784))
model.add(Activation('relu'))
model.add(Dense(30))
model.add(Activation('relu'))
model.add(Dense(10))
model.add(Activation('softmax'))
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 30)                23550
_________________________________________________________________
activation_1 (Activation)    (None, 30)                0
_________________________________________________________________
dense_2 (Dense)              (None, 30)                930
_________________________________________________________________
activation_2 (Activation)    (None, 30)                0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                310
_________________________________________________________________
activation_3 (Activation)    (None, 10)                0
=================================================================
Total params: 24,790
Trainable params: 24,790
Non-trainable params: 0
_________________________________________________________________
```

```python
# Learning algorithm, Loss
from keras.optimizers import SGD
model.compile(optimizer=SGD(lr=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
```

```python
# Train and test
H = model.fit(x_train, y_train, batch_size=32, epochs=5, validation_data=(x_test, y_test))
```