What is

AN ARTIFICIAL NEURAL NETWORK

SOURCE : NEURAL NETWOKS EXPLAINED FROM SCRATCH USING PYTHON / YOUTUBE : BOT ACADEMY

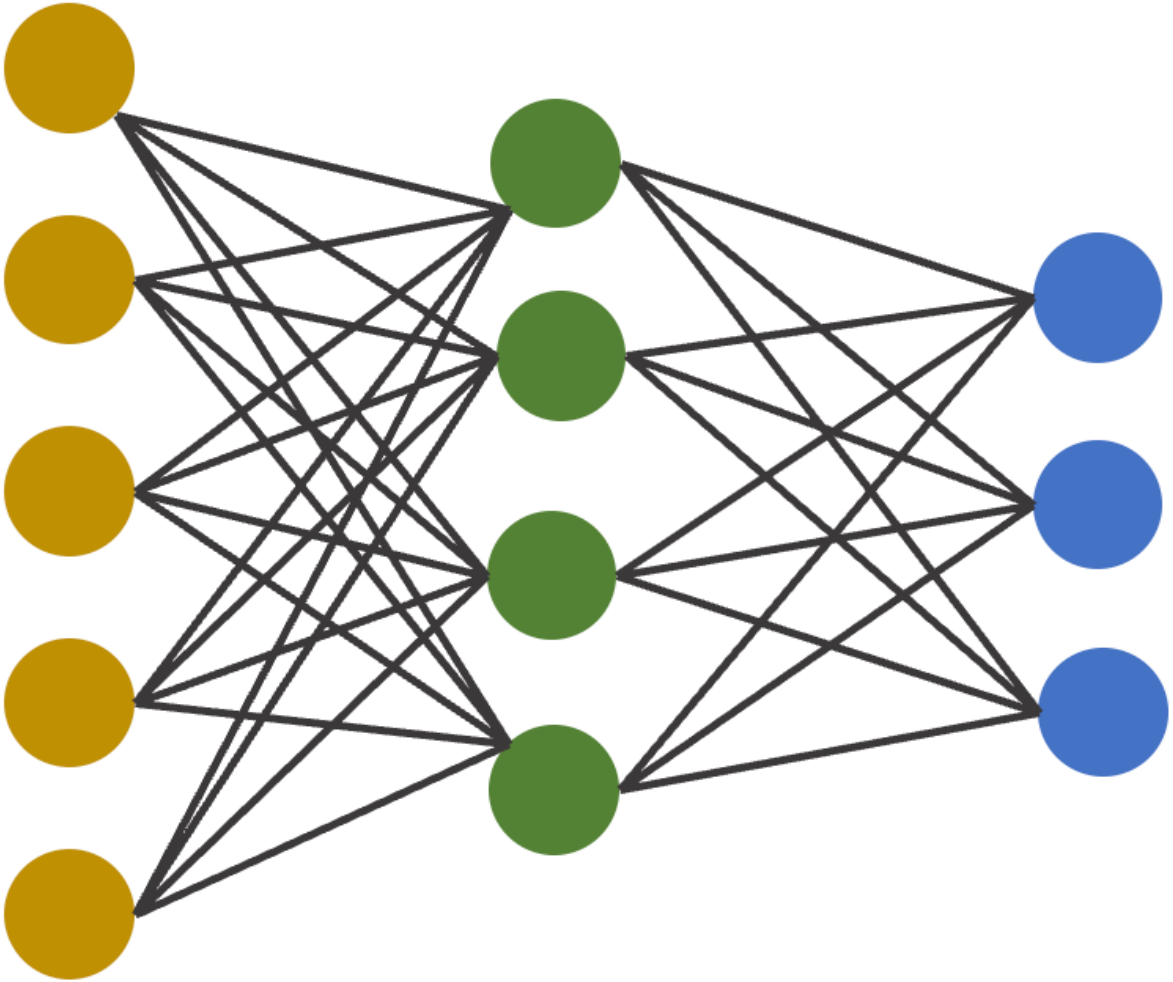
[01-2024]

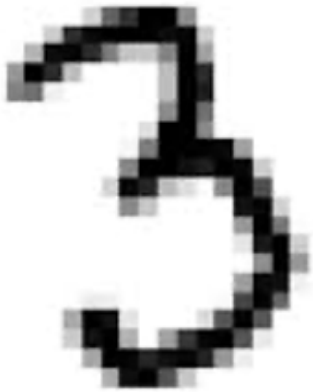
Jean-Michel Torres

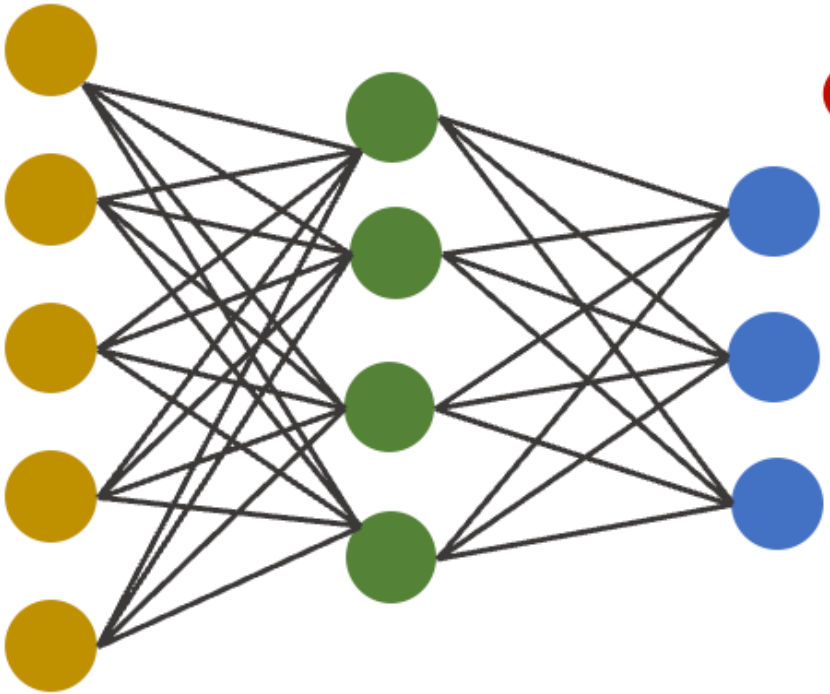
torresjm@fr.ibm.com

```python
from data import get_mnist
import numpy as np
import matplotlib.pyplot as plt

images, labels = get_mnist()
w_i_h = np.random.uniform(-0.5, 0.5, (20, 784))
w_h_o = np.random.uniform(-0.5, 0.5, (10, 20))
b_i_h = np.zeros((20, 1))
b_h_o = np.zeros((10, 1))

learn_rate = 0.01 ; nr_correct = 0 ; epochs = 3
for epoch in range(epochs):
    for img, l in zip(images, labels):
        img.shape += (1,)
        l.shape += (1,)
        # Forward propagation input -> hidden
        h_pre = b_i_h + w_i_h @ img
        h = 1 / (1 + np.exp(-h_pre))
        # Forward propagation hidden -> output
        o_pre = b_h_o + w_h_o @ h
        o = 1 / (1 + np.exp(-o_pre))
        # Cost / Error calculation
        e = 1 / len(o) * np.sum((o - l) ** 2, axis=0)
        nr_correct += int(np.argmax(o) == np.argmax(l))
        # Backpropagation output -> hidden (cost function derivative)
        delta_o = o - l
        w_h_o += -learn_rate * delta_o @ np.transpose(h)
        b_h_o += -learn_rate * delta_o
        # Backpropagation hidden -> input (activation function derivative)
        delta_h = np.transpose(w_h_o) @ delta_o * (h * (1 - h))
        w_i_h += -learn_rate * delta_h @ np.transpose(img)
        b_i_h += -learn_rate * delta_h
    # Show accuracy for this epoch
    print(f"Epoch : {epoch}, Précision : {round((nr_correct / images.shape[0]) * 100, 2)}%")
    nr_correct = 0
```

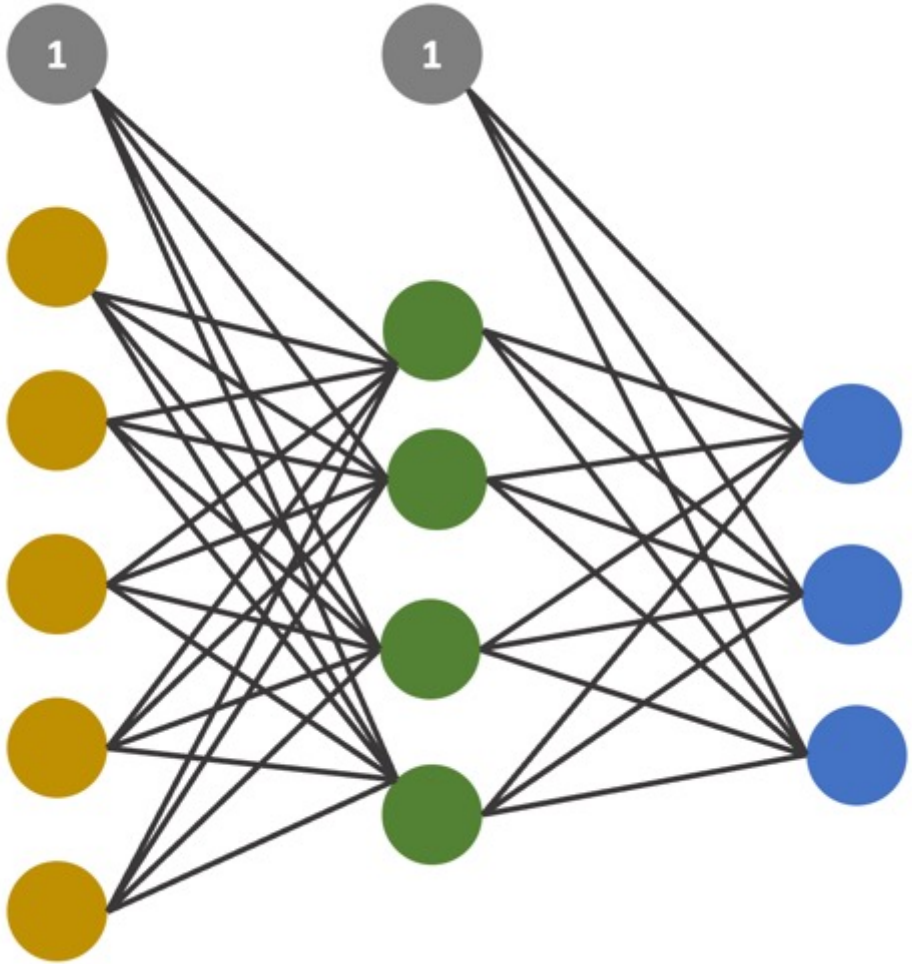
```
# Show results
while True:
    index = int(input("Entrer un nombre entre 0 et 59999 : "))
    img = images[index]
    plt.imshow(img.reshape(28, 28), cmap="Greys")

    img.shape += (1,)
    # Forward propagation input -> hidden
    h_pre = b_i_h + w_i_h @ img.reshape(784, 1)
    h = 1 / (1 + np.exp(-h_pre))
    # Forward propagation hidden -> output
    o_pre = b_h_o + w_h_o @ h
    o = 1 / (1 + np.exp(-o_pre))

    plt.title(f"le réseau de neurones a reconnu le chiffre {o.argmax()} ")
    plt.show()
```

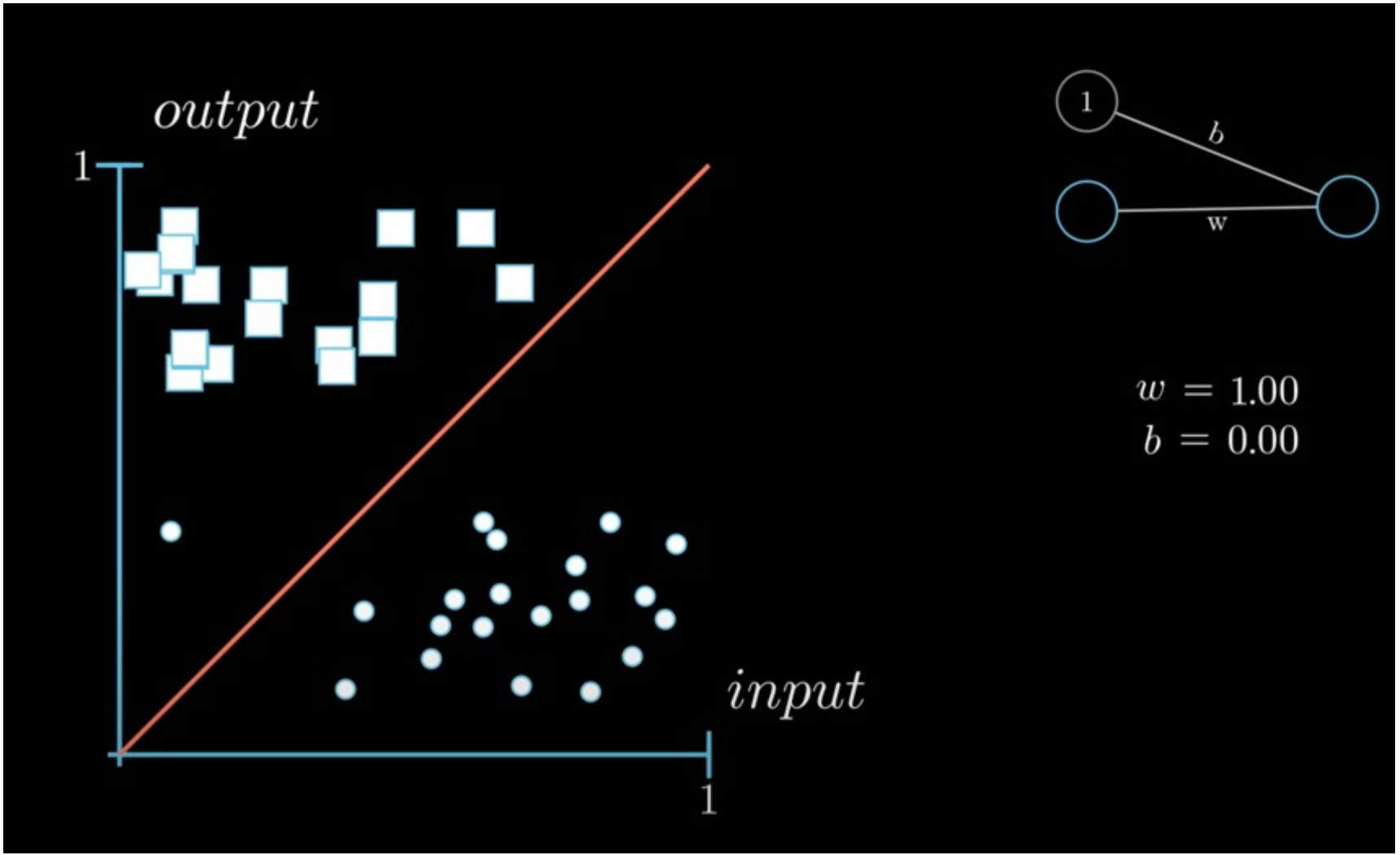


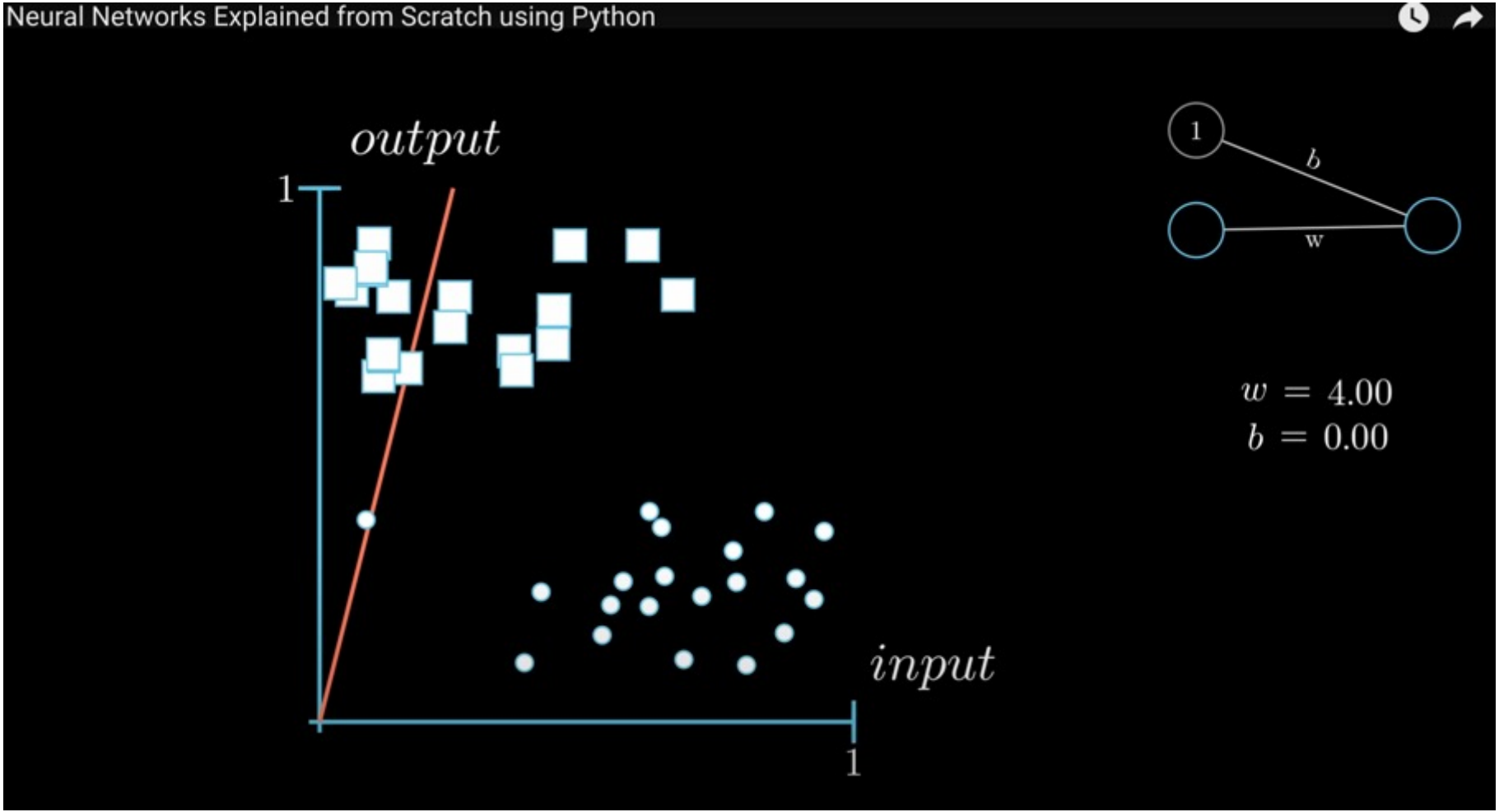


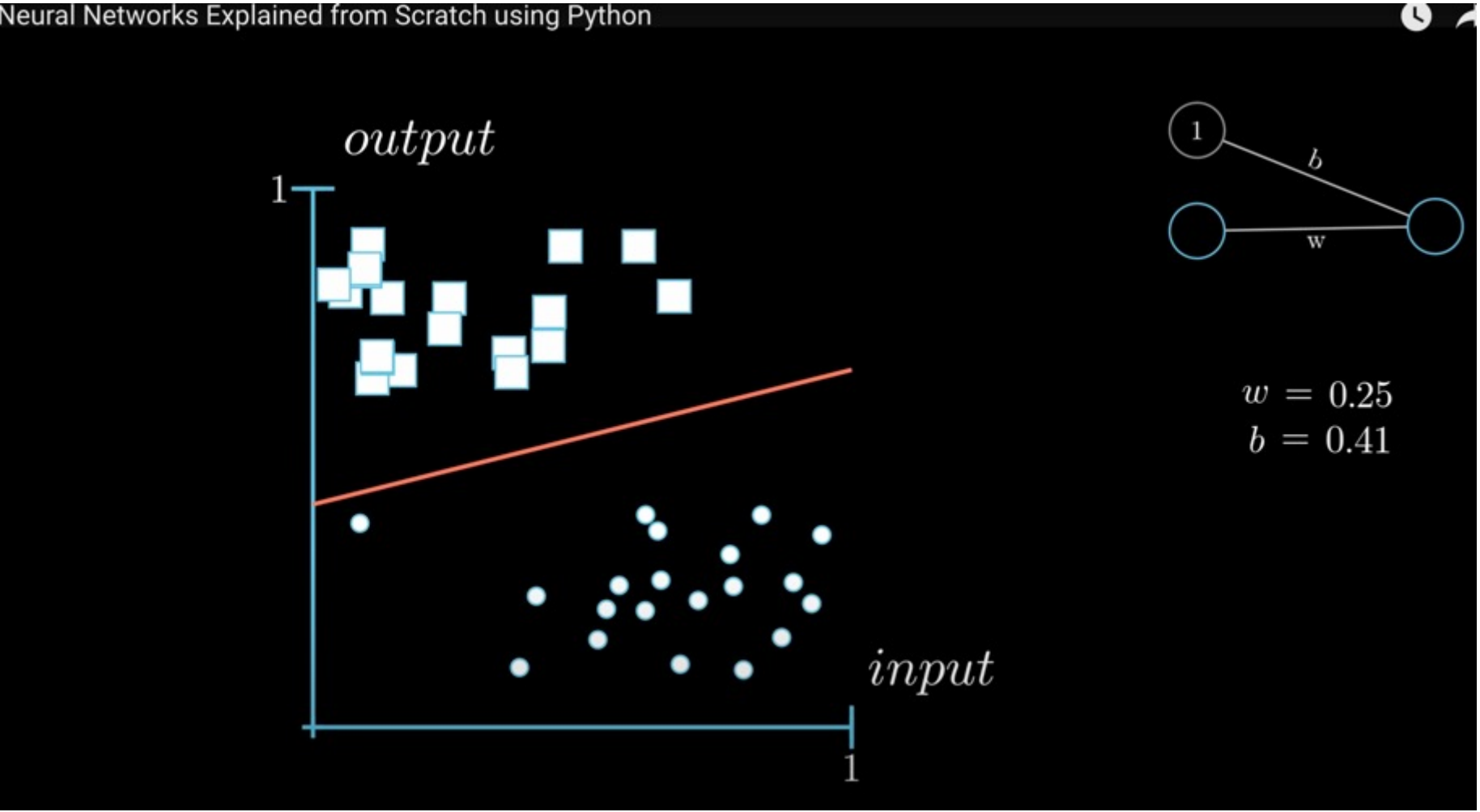


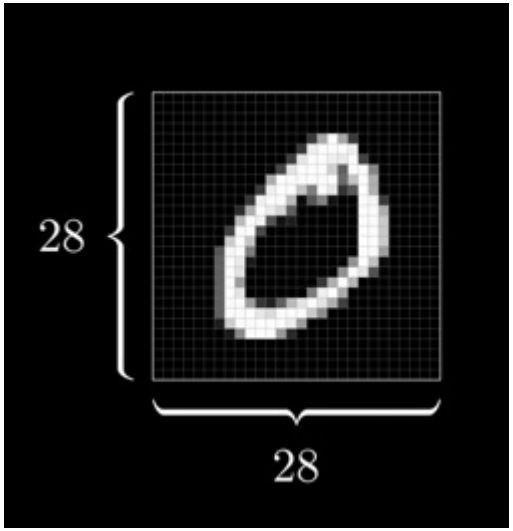

I think this
is a 3

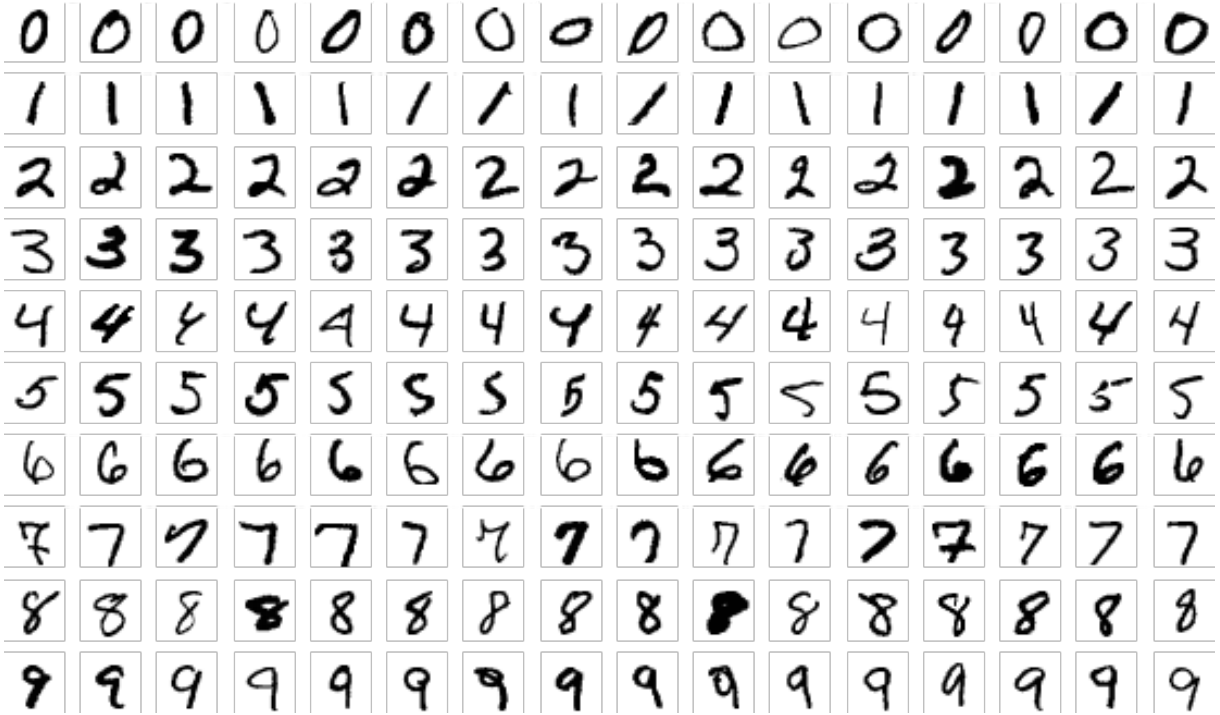

1
1

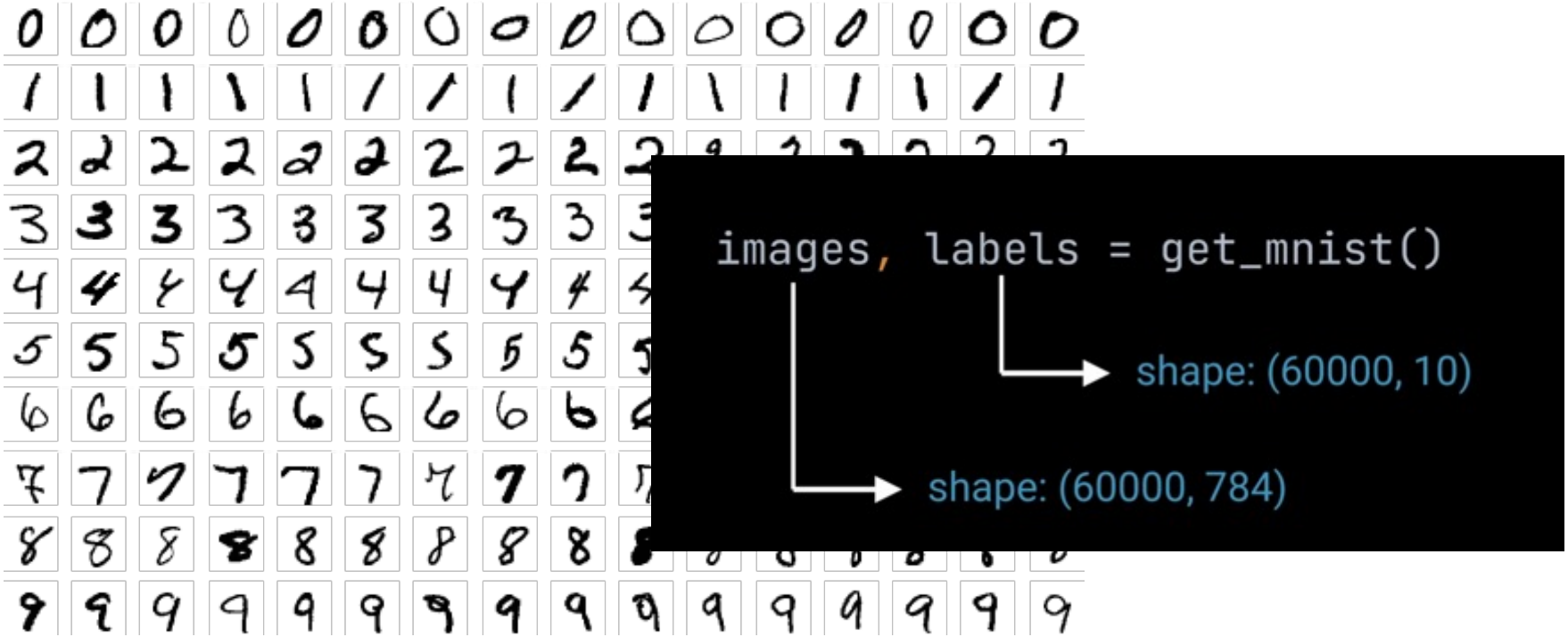

output
1
input
1
1
b
w
$w = 1.00$
$b = 0.00$


Neural Networks Explained from Scratch using Python
output
1
input
1
1
b
w
$w = 4.00$
$b = 0.00$

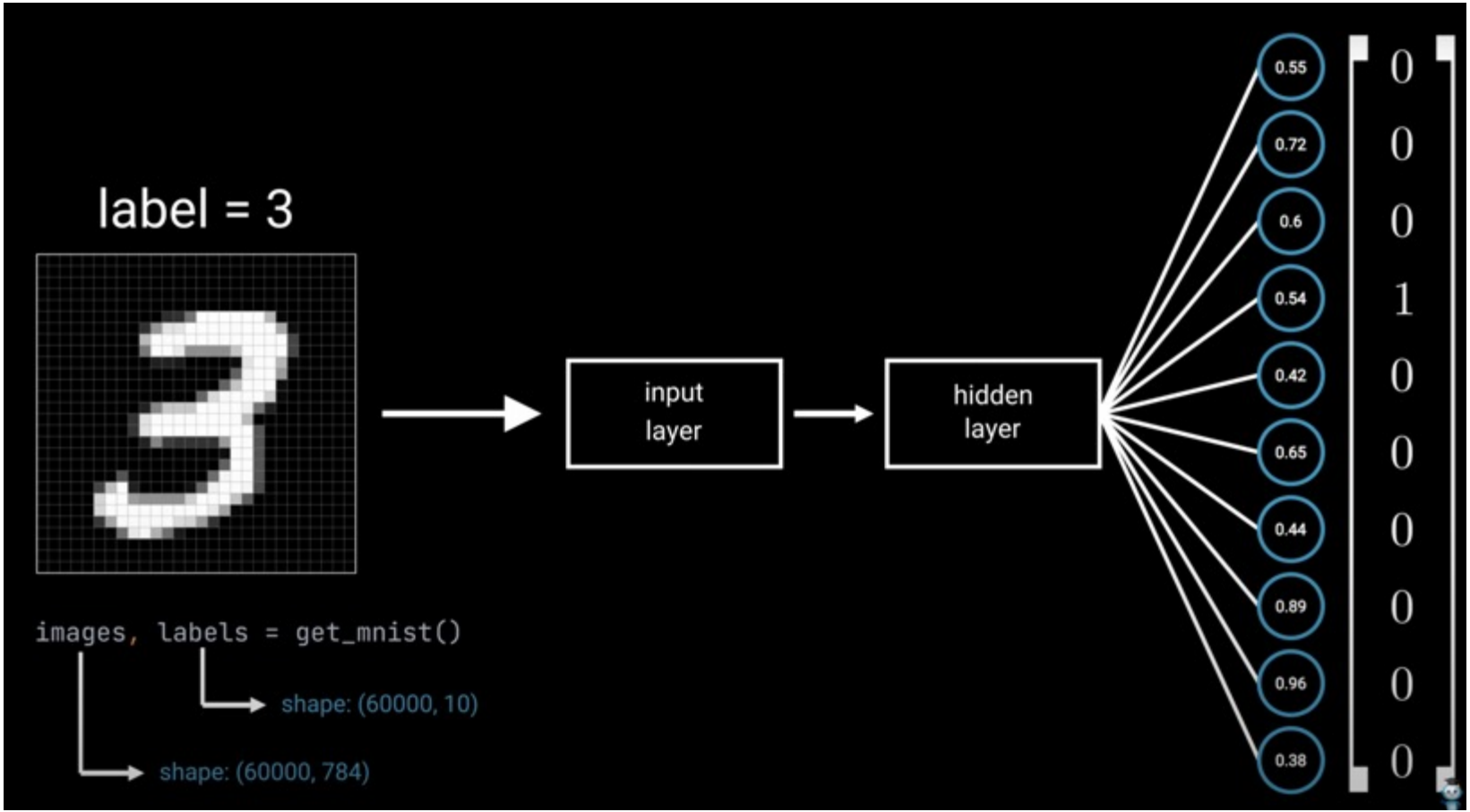

Neural Networks Explained from Scratch using Python
output
1
input
1
1
b
w
$w = 0.25$
$b = 0.41$


28
28

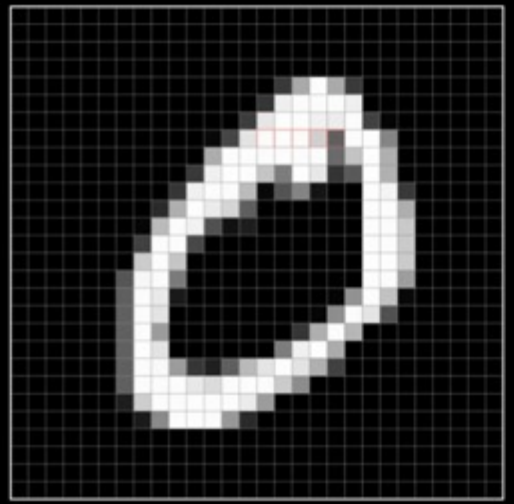


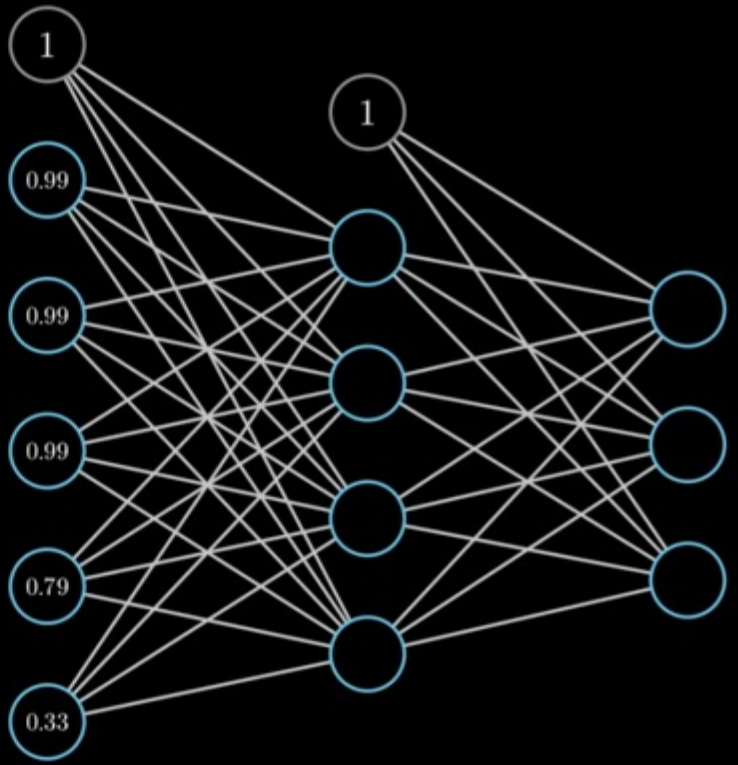


This loads in memory 60000 images (of 784 pixels, each pixel is a byte coding 256 levels of grey)
And we load lthe corresponding labels (for each image one of the 10 possible numbers (0,1, ..., 9).


label = 3
input layer
hidden layer
0.55
0.72
0.6
0.54
0.42
0.65
0.44
0.89
0.96
0.38
0
0
0
1
0
0
0
0
0
images, labels = get_mnist()
shape: (60000, 10)
shape: (60000, 784)

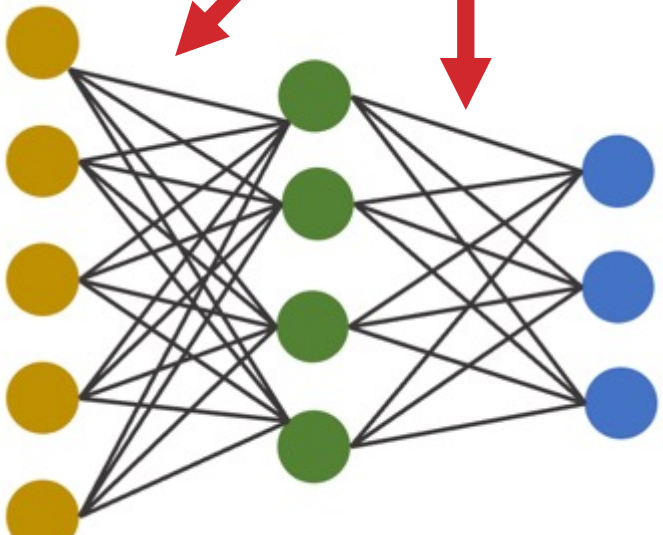
label = 0





We start with filling RANDOM values in the Neural Network weights

```
w = weights, b = bias, i = input, h = hidden, o = output, l = label
e.g. w_i_h = weights from input layer to hidden layer
"""

w_i_h = np.random.uniform(-0.5, 0.5, (20, 784))
w_h_o = np.random.uniform(-0.5, 0.5, (10, 20))
```



(this specifies the size of the hidden layer)

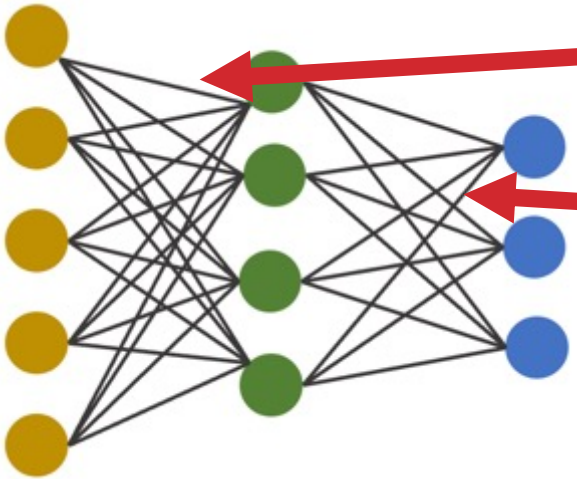
`learn_rate` : represent how much we change the weight between neurons each time we modify this.

```
learn_rate = 0.01
nr_correct = 0
epochs = 3
```

`epochs` : number of « passes » (for each pass we use all the data : images and labels)

`(nr_correct` : a counter to record how many times the network guessed correctly)

Now for each epoch, and for each image : we present the image in the NN entry and we compute the value of the next layer using the current weights.

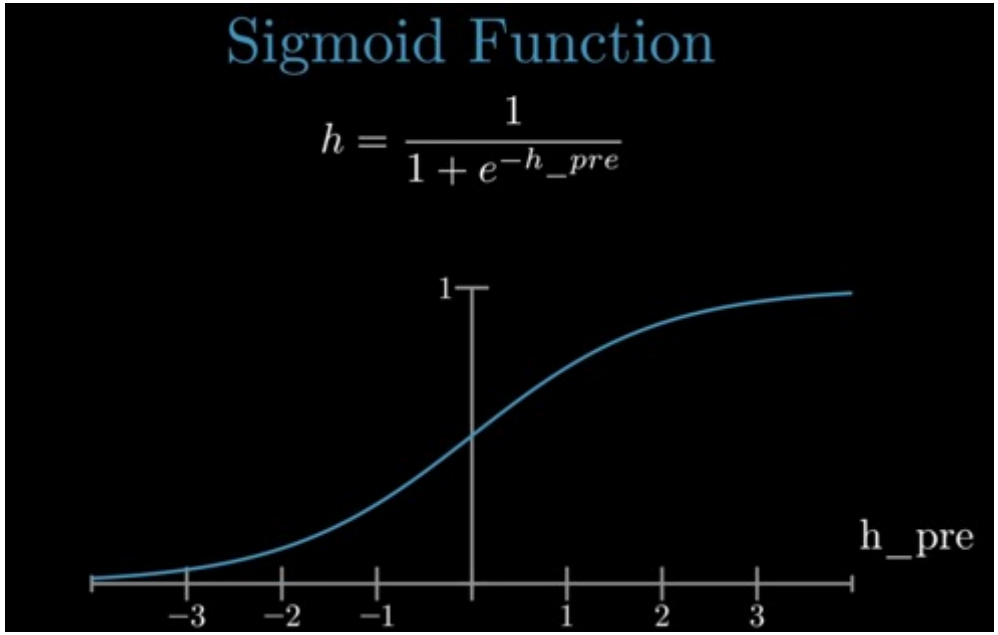


```
for epoch in range(epochs):
    for img, l in zip(images, labels):
        img.shape += (1,)
        l.shape += (1,)
        # Forward propagation input -> hidden
        h_pre = b_i_h + w_i_h @ img
        h = 1 / (1 + np.exp(-h_pre))
        # Forward propagation hidden -> output
        o_pre = b_h_o + w_h_o @ h
        o = 1 / (1 + np.exp(-o_pre))
```

One more detail of the computation :

```
o = 1 / (1 + np.exp(-o_pre))
```

```
h = 1 / (1 + np.exp(-h_pre))
```



It is a technical detail : a renormalization to avoid values in the neurons to be in a large range. The sigmoid function keeps all values beteen 0 an 1.

```
for epoch in range(epochs):
    for img, l in zip(images, labels):
        img.shape += (1,)
        l.shape += (1,)
        # Forward propagation input -> hidden
        h_pre = b_i_h + w_i_h @ img
        h = 1 / (1 + np.exp(-h_pre))
        # Forward propagation hidden -> output
        o_pre = b_h_o + w_h_o @ h
        o = 1 / (1 + np.exp(-o_pre))
```

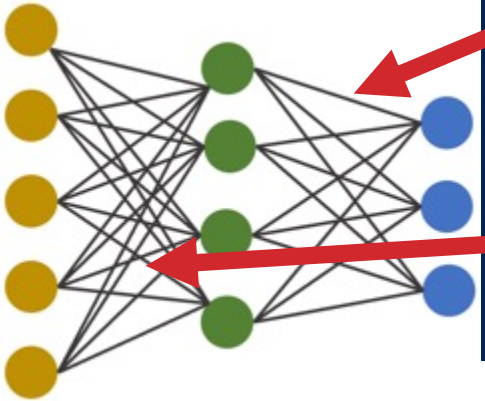
```
# Cost / Error calculation
e = 1 / len(o) * np.sum((o - l) ** 2, axis=0)
nr_correct += int(np.argmax(o) == np.argmax(l))
```

When the output values « **o** », have been computed, it is possible to evaluate the difference between the label value (**l**) and the output : **delta_o = o - l**

And this value (plus the `learning_rate` is the used to update the weights « `w_h_o` ».

Just next, we also update the weights between hidden and inputs (« `w_i_h` »)

And that's it, we do the same with next image, and when all images have been used, we do the same for next epoch.



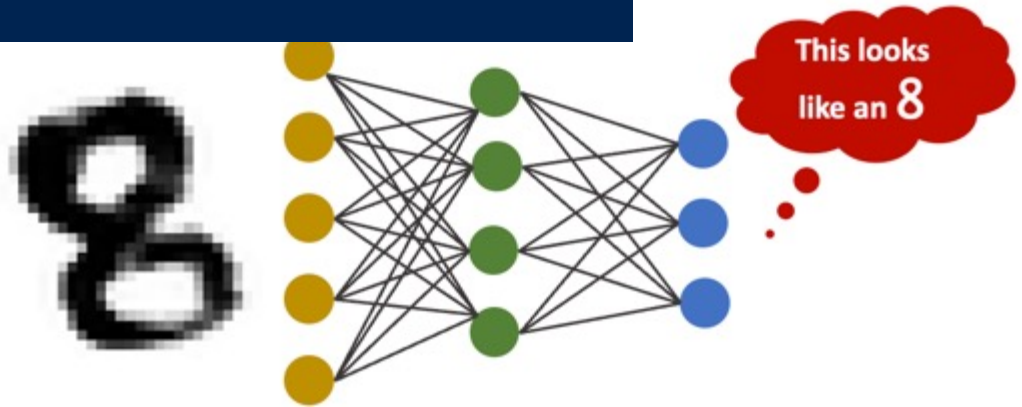
```
# Backpropagation output -> hidden (cost function derivative)
delta_o = o - l
w_h_o += -learn_rate * delta_o @ np.transpose(h)
b_h_o += -learn_rate * delta_o
# Backpropagation hidden -> input (activation function derivative)
delta_h = np.transpose(w_h_o) @ delta_o * (h * (1 - h))
w_i_h += -learn_rate * delta_h @ np.transpose(img)
b_i_h += -learn_rate * delta_h
```

At each epoch end we display the success rate:

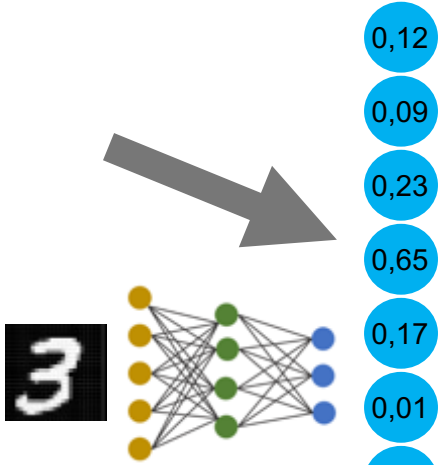
```
    # Show accuracy for this epoch
    print(f"Epoch : {epoch}, Précision : {round((nr_correct / images.shape[0]) * 100, 2)}%")
    nr_correct = 0
```

```python
# Show results
while True:
    index = int(input("Entrer un nombre entre 0 et 59999 : "))
    img = images[index]
    plt.imshow(img.reshape(28, 28), cmap="Greys")

    img.shape += (1,)
    # Forward propagation input -> hidden
    h_pre = b_i_h + w_i_h @ img.reshape(784, 1)
    h = 1 / (1 + np.exp(-h_pre))
    # Forward propagation hidden -> output
    o_pre = b_h_o + w_h_o @ h
    o = 1 / (1 + np.exp(-o_pre))

    plt.title(f"le réseau de neurones a reconnu le chiffre {o.argmax()} ")
    plt.show()
```

We the epochs are finished, training is done. We can use the network for predicting (thisis the inference phase)
The computation is the same as in the beginning (forward propagation)



- The user is prompted to choose an image (by it's number : 0 to 59999)
- The program displays the image chosen, and computes the output value
- Finally, we display the neuron number with the max value



0,12
0,09
0,23
0,65
0,17
0,01
0,31
0,05
0,42
0,27

```
# Show results
while True:
    index = int(input("Entrer un nombre entre 0 et 59999 : "))
    img = images[index]
    plt.imshow(img.reshape(28, 28), cmap="Greys")

    img.shape += (1,)
    # Forward propagation input -> hidden
    h_pre = b_i_h + w_i_h @ img.reshape(784, 1)
    h = 1 / (1 + np.exp(-h_pre))
    # Forward propagation hidden -> output
    o_pre = b_h_o + w_h_o @ h
    o = 1 / (1 + np.exp(-o_pre))

    plt.title(f"le réseau de neurones a reconnu le chiffre {o.argmax()} ")
    plt.show()
```

Try it :

You can change :

- Number of epochs,
- Hidden layer size,
- Learning rate.

- Conclusions ?