Qu’est-ce qu’un

RÉSEAU DE NEURONES ARTIFICIELS ?

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

[01-2024]

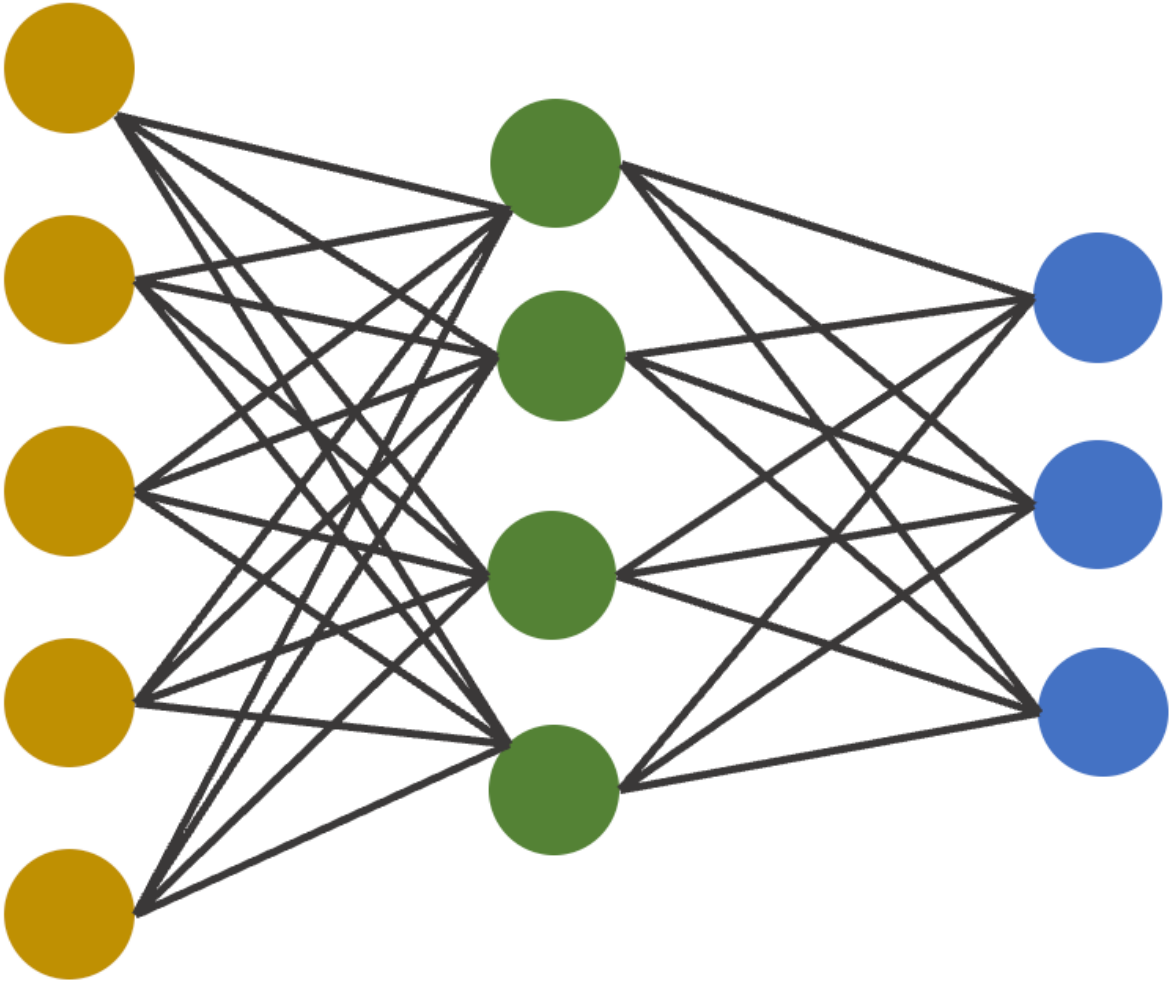
Jean-Michel Torres

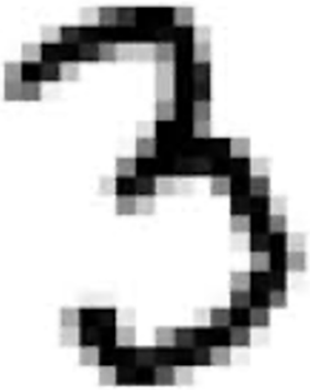
torresjm@fr.ibm.com

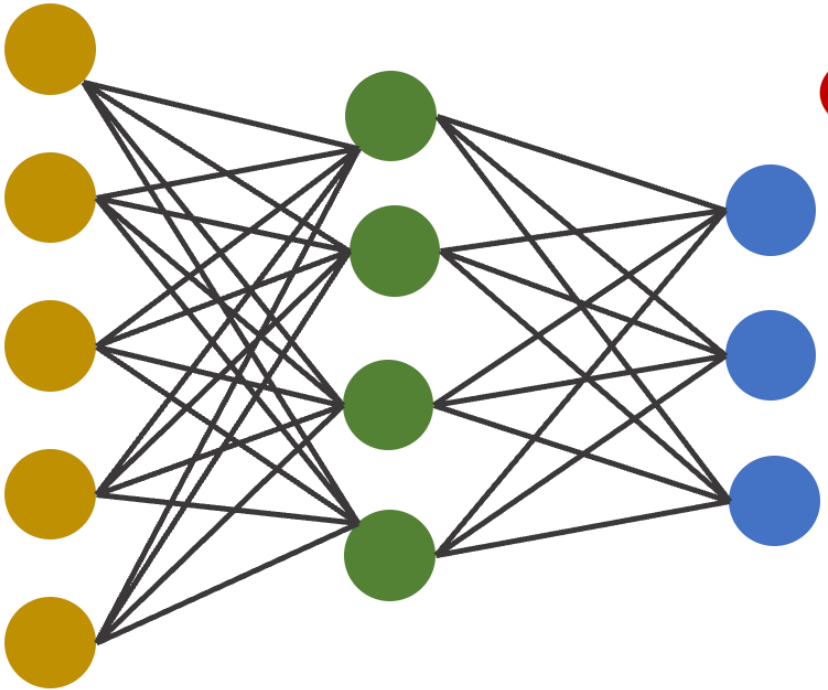
P-TECH édition

```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
```

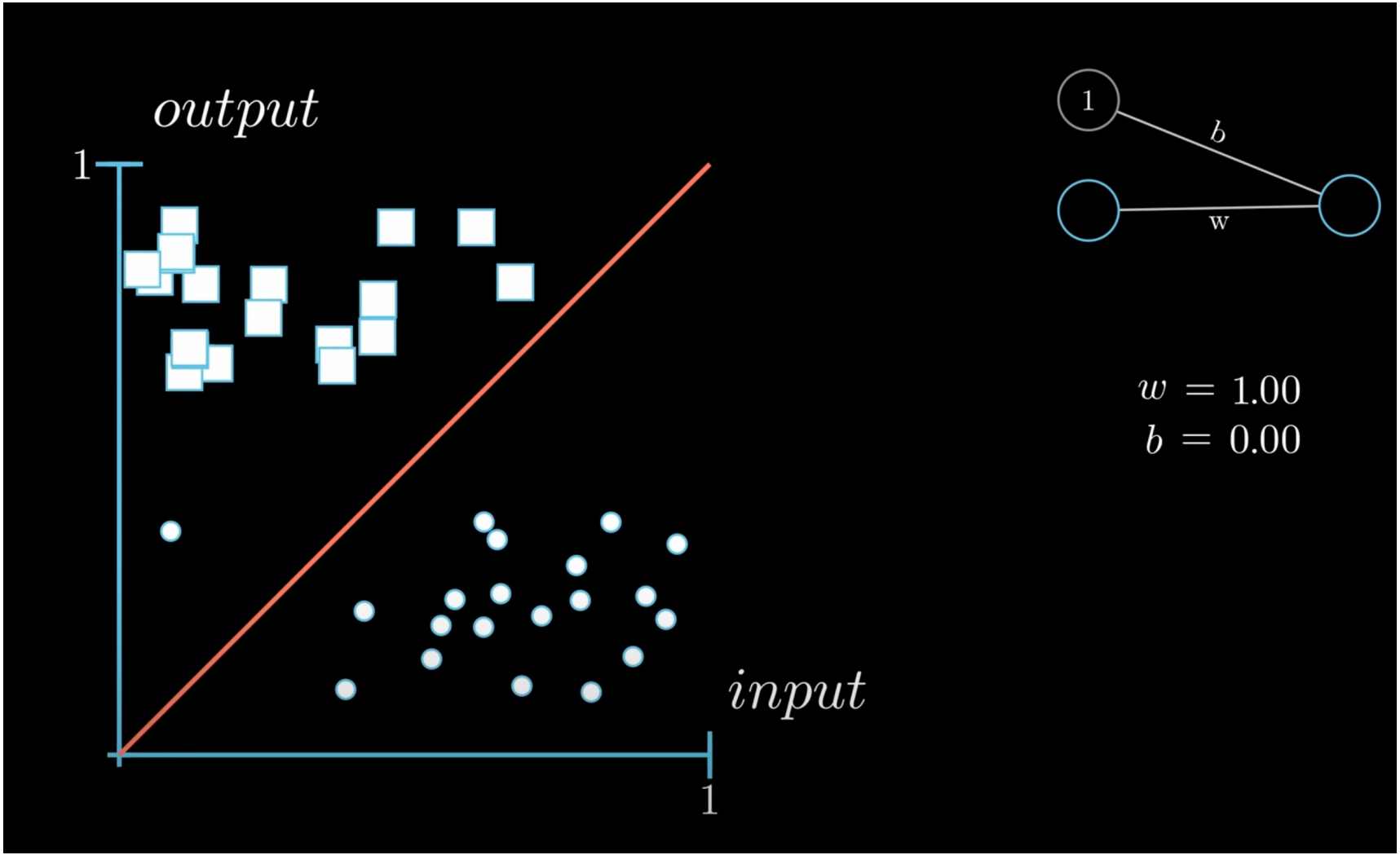
```
# Show results
while True:
    index = int(input("Entrer un nombre entre 0 et 59999 : "))
    img = images[index]
    plt.imshow(img.reshape(28, 28), cmap="Greys")

    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()
```



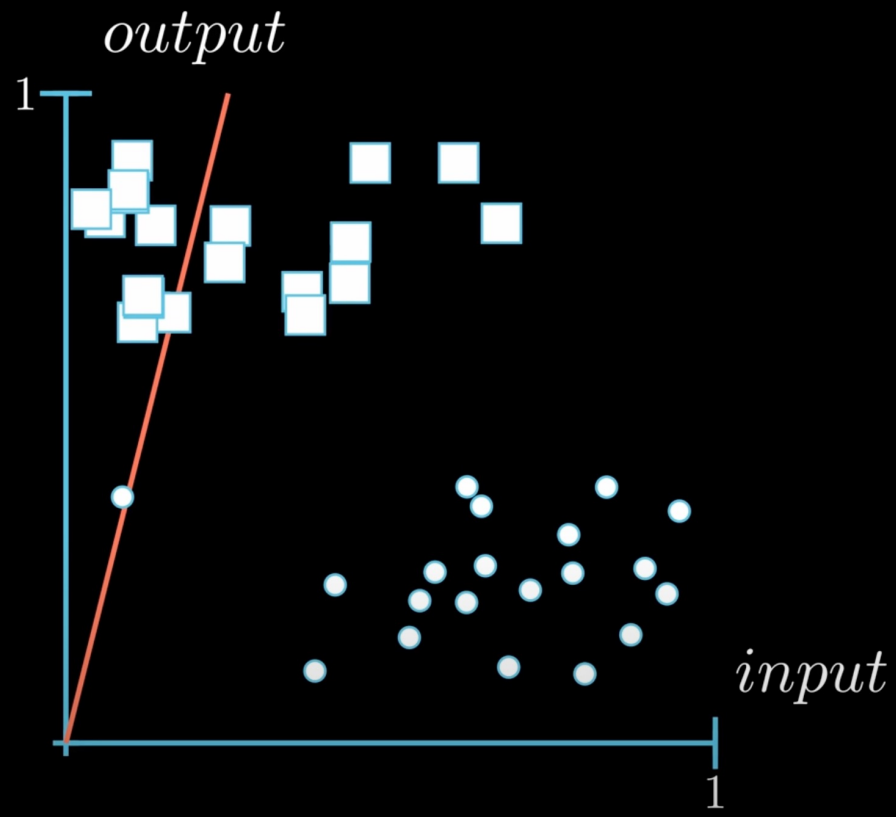


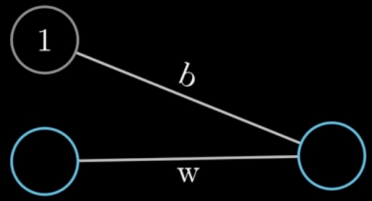



On dirait un
3


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

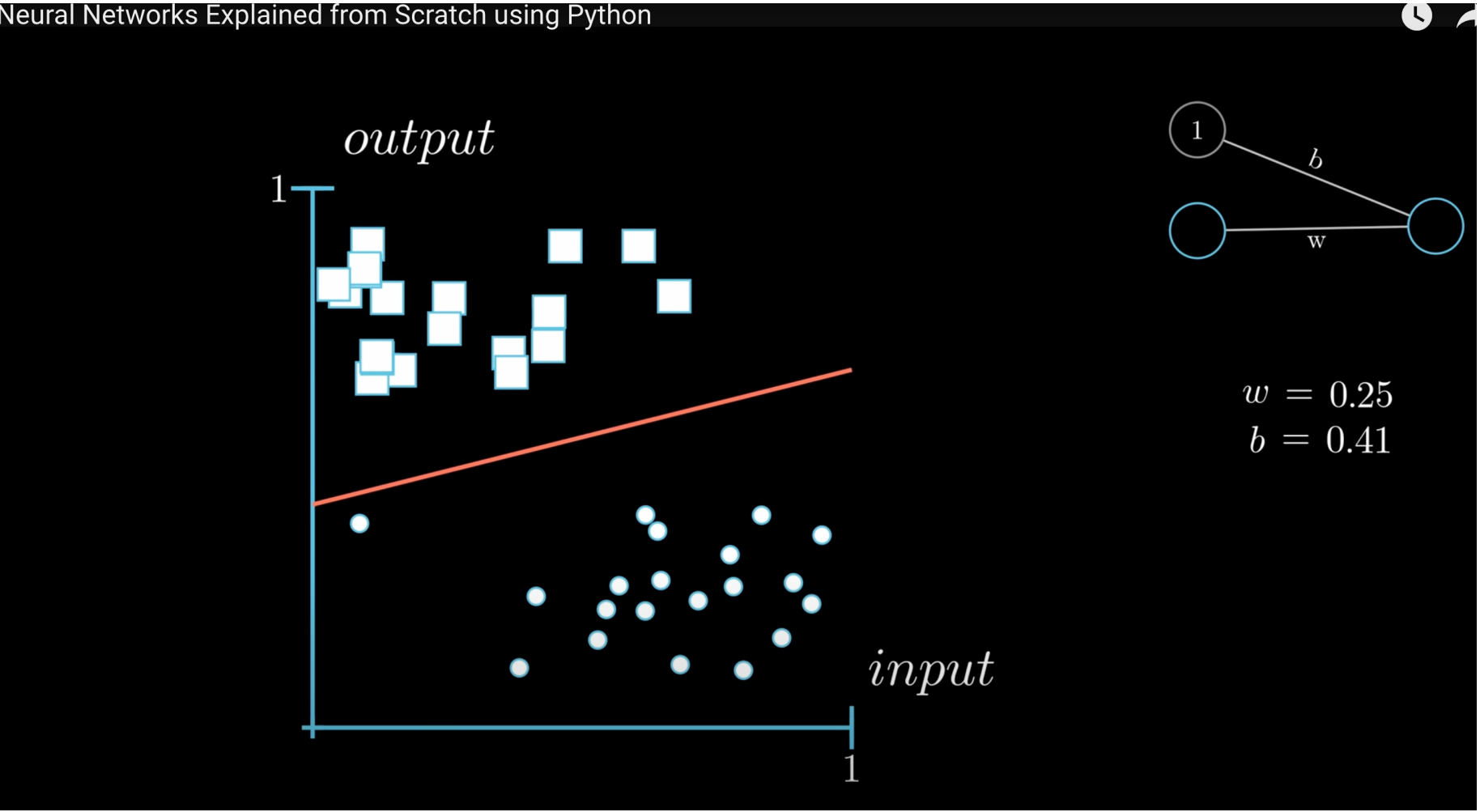
Neural Networks Explained from Scratch using Python

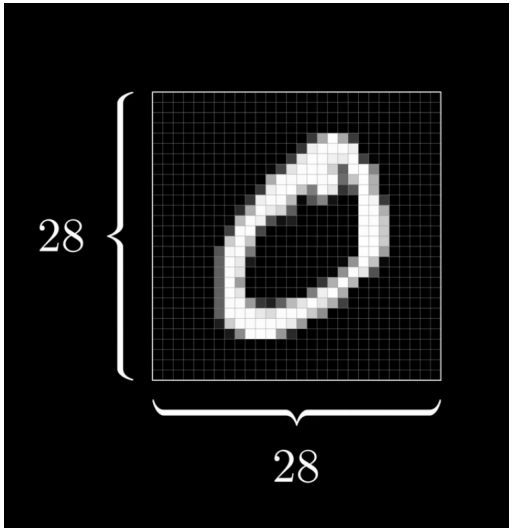


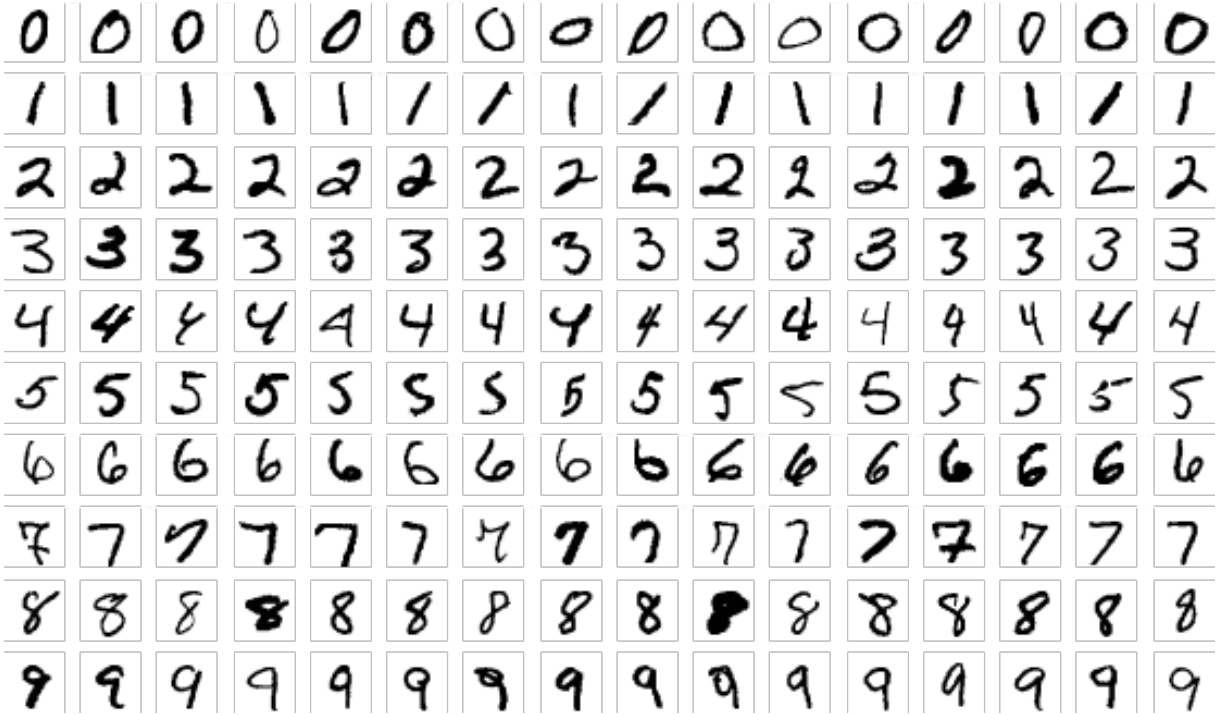


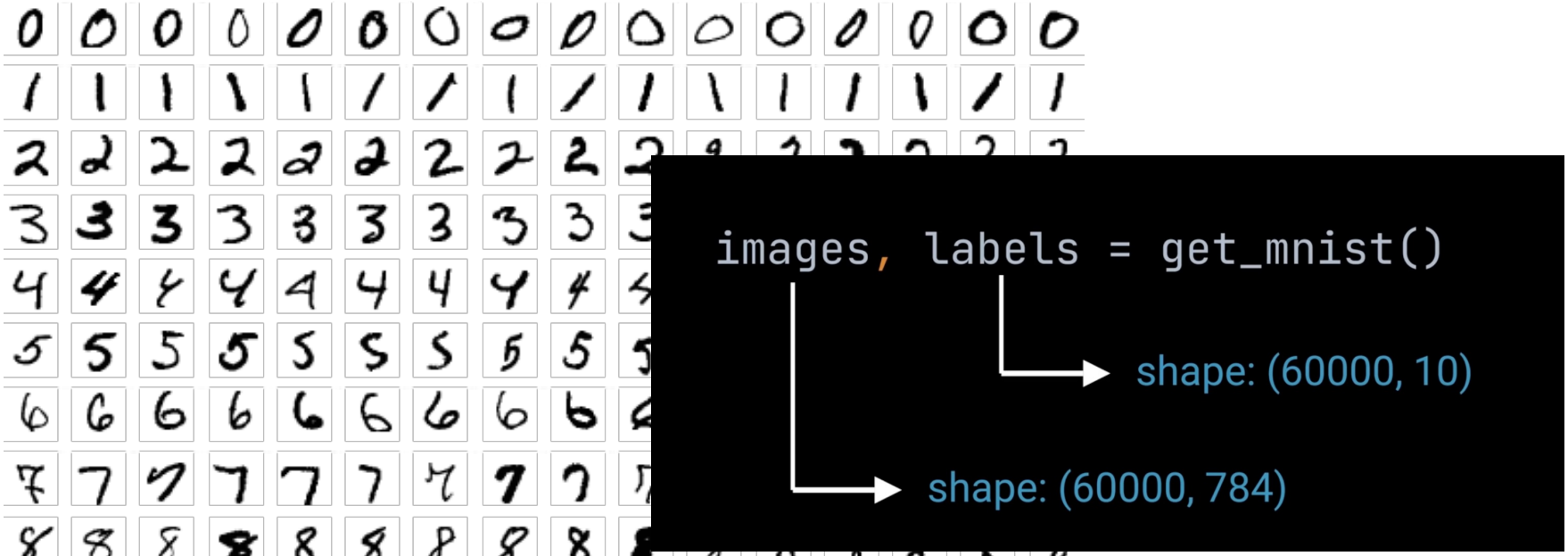
$w = 4.00$

$b = 0.00$

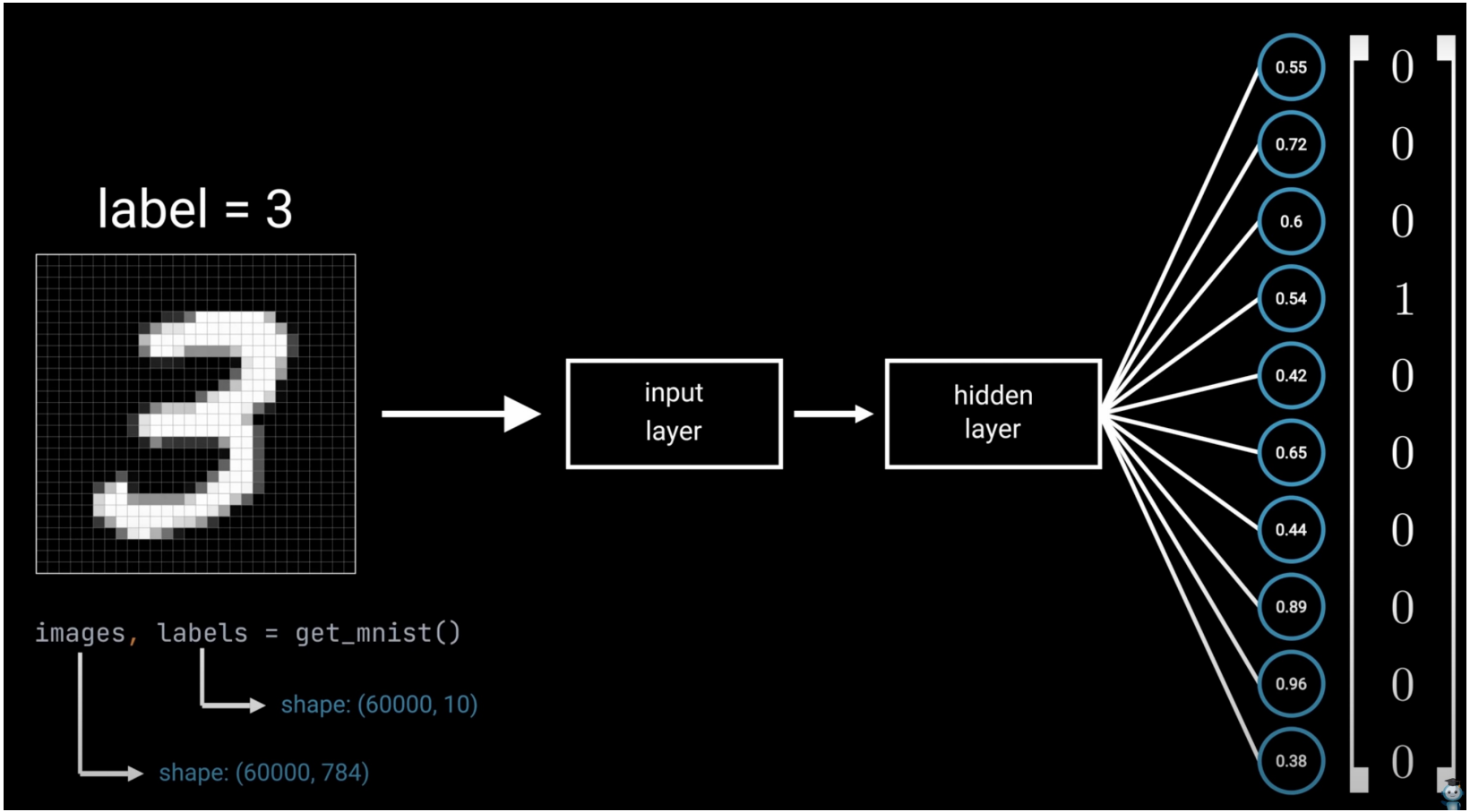

output
1
input
1
1
b
w
$w = 0.25$
$b = 0.41$


28
28

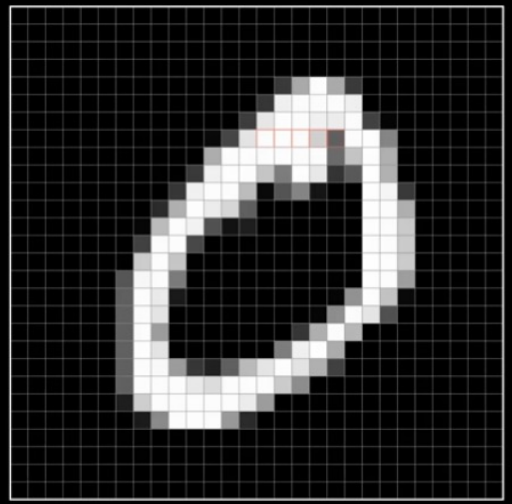


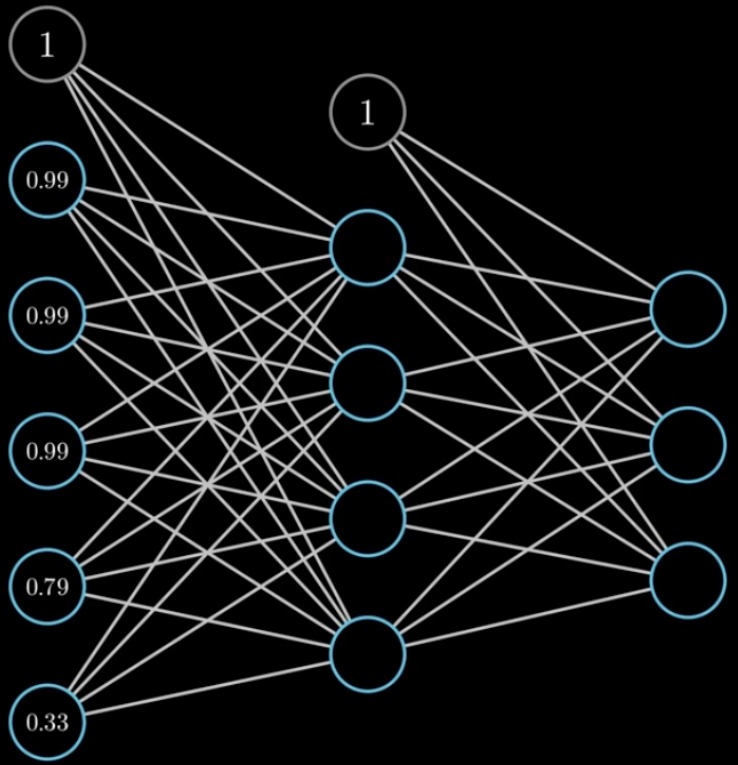


On charge dans la mémoire 60000 images de 784 pixels (donc un tableau de 60000 par 784 contenant un nombre entre 0 et 255 représentant le niveau de gris pour chaque pixel.
Et on charge les étiquettes correspondantes qui désignent pour chaque image un des 10 chiffres possibles.


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)

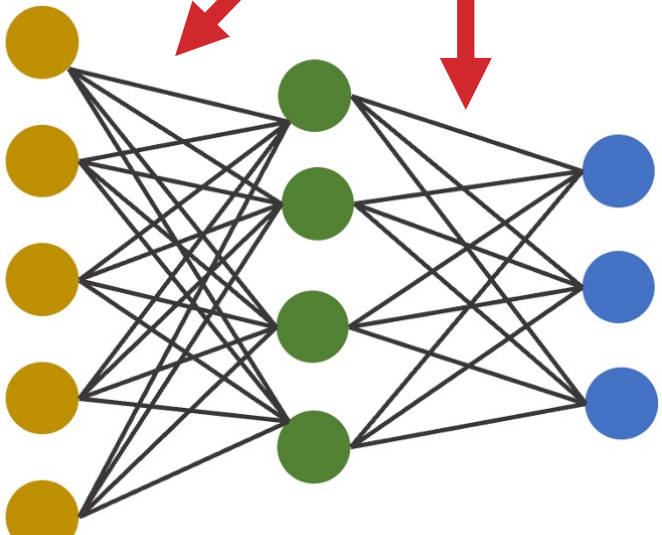
label = 0





On commence par mettre AU HASARD les valeurs des poids dans le réseau.

```
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))
```



(on a choisi une couche cachée contenant 20 neurones)

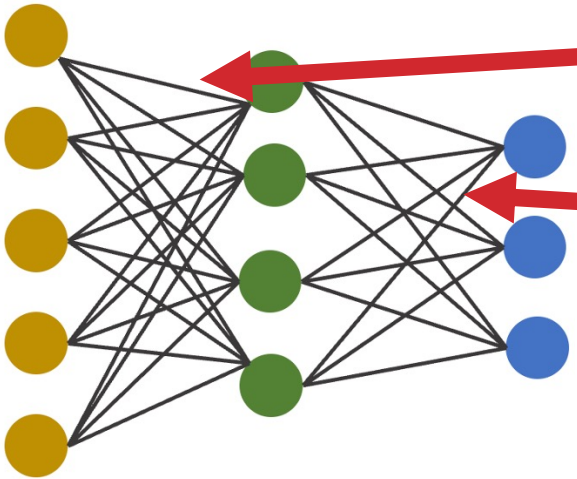
`learn_rate` : taux d'apprentissage, c'est la « vitesse » à laquelle on va changer la « force de liaison » entre les neurones (les valeurs des poids entre les neurones)

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

`epochs` : nombre de fois où on va utiliser tout le paquet d'image pour faire l'entrainement.

`(nr_correct` : juste un compteur pour savoir combien de fois le réseau a correctement classé une image).

Maintenant, pour chaque epoch, pour chaque image : on met l'image (les valeurs des pixels de l'image) en entrée du réseau et on calcule les neurones de la couche suivante avec les valeurs des poids qui les relient.

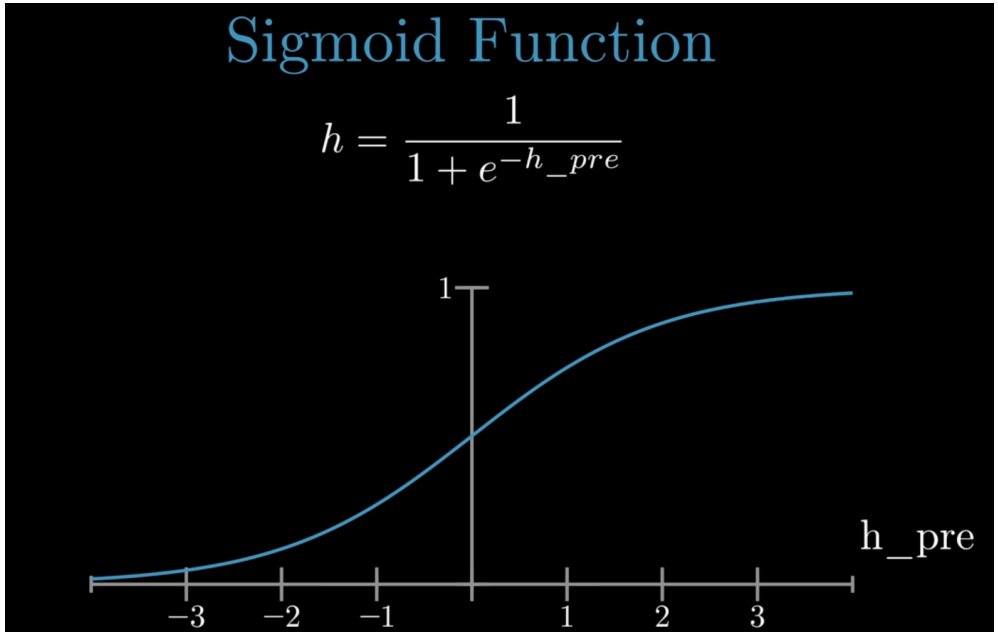


```
for epoch in range(epochs):
    for img, l in zip(images, labels):
        img.shape += (1,)
        l.shape += (1,)
        # Forward propagation input -> hidden
        h_pre = b_i_h + w_i_h @ img
        h = 1 / (1 + np.exp(-h_pre))
        # Forward propagation hidden -> output
        o_pre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

Un détail du calcul :

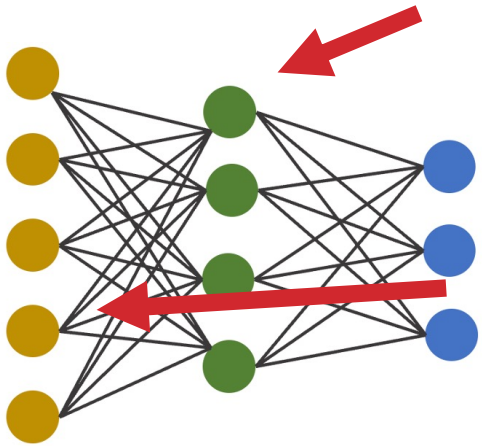
```
o = 1 / (1 + np.exp(-o_pre))
```

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



C’est une « renormalisation », pour éviter que les valeurs des neurones soient trop dispersées), on les garde entre 0 et 1

Une fois qu'on a calculé les valeurs de sortie « o », on peut calculer la différence avec la valeur attendue « l » : **`delta_o = o - l`**
Et on utilise cette valeur pour mettre à jour les poids « `w_h_o` » (ainsi que le `learning_rate`)
Et puis on met aussi juste après les poids entre la couche d'entrée et la couche cachée (« `w_i_h` »)
Et c'est fini, on passe à l'image suivante et quand on a fini, on passe à l'epoch suivante.

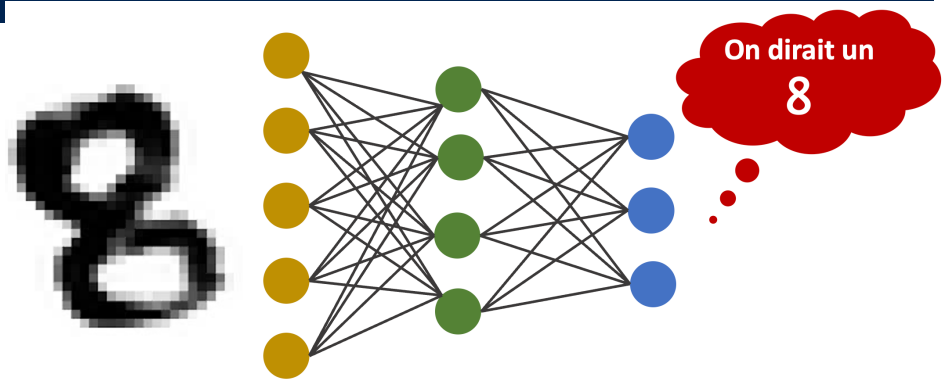


```
# Backpropagation output -> hidden (cost function derivative)
delta_o = o - l
w_h_o += -learn_rate * delta_o @ np.transpose(h)
b_h_o += -learn_rate * delta_o
# Backpropagation hidden -> input (activation function derivative)
delta_h = np.transpose(w_h_o) @ delta_o * (h * (1 - h))
w_i_h += -learn_rate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

A la fin de chaque epoch on affiche le taux de réussite :

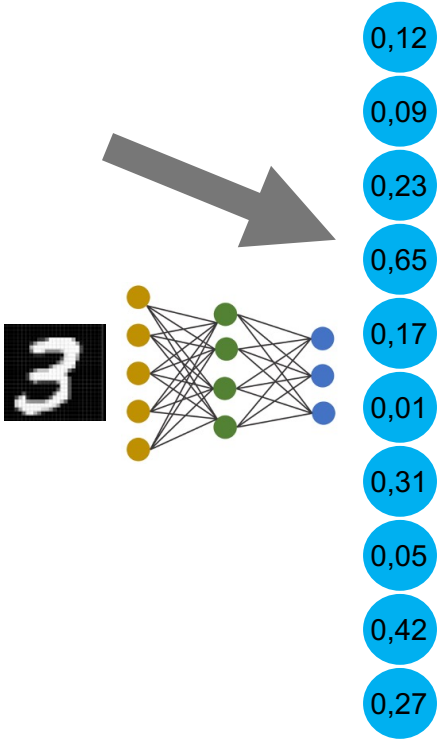
```
    # Show accuracy for this epoch
    print(f"Epoch : {epoch}, Précision : {round((nr_correct / images.shape[0]) * 100, 2)}%")
    nr_correct = 0
```

```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()
```



Phase appelée « inférence » (utilisation).
C'est le même calcul qu'au début du programme, sauf que maintenant on a fini l'entrainement et on lui demande de reconnaître l'image et on pense qu'il va réussir

- On choisit une image par son numéro (entre 0 et 59999),
- Le programme affiche l'image puis utilise les poids des 2 couches pour calculer la valeur de sortie
- Enfin il affiche le numéro du neurone parmi ceux de la couche de sortie qui contient la plus grande valeur.




```
# 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()
```

Programme :

- NN.py : il y a une première phase d'entrainement, le programme affiche la performance du réseau atteint après chaque epoch

- Faire varier le nombre « d'epochs »,
- Faire varier le nombre de neurones de la couche cachée,
- Faire varier le « taux d'apprentissage ».

- Que peut-on en déduire ?