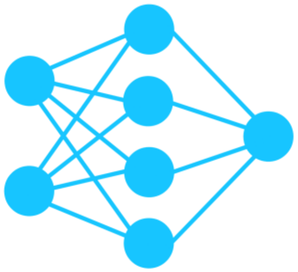
Reconnaissance d'images par réseaux de neurones



Mouad En-nasiry

Plan

- Motivation
- Réseau de neurones
- Optimisation des paramètres du réseau par entraînement
  - descente de gradient
  - Rétropropagation du gradient
  - Processus de rétropropagation du gradient
- Analyse de l'influence des paramètres du réseau sur l'apprentissage et évaluation avec étude des limites du modèle

Motivation

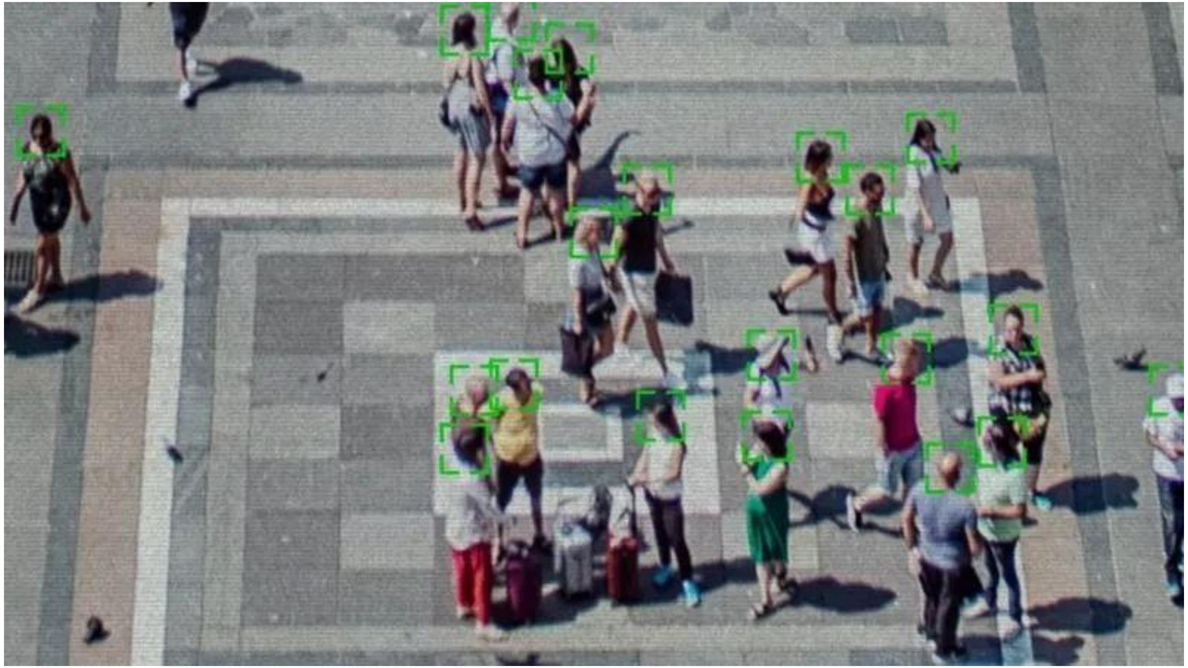


Figure 1 : Utilisation de la reconnaissance faciale en Australie comme outil de surveillance

1.

Réseau de neurones

Définition de notion, fonctionnement …

Réseau de neurones

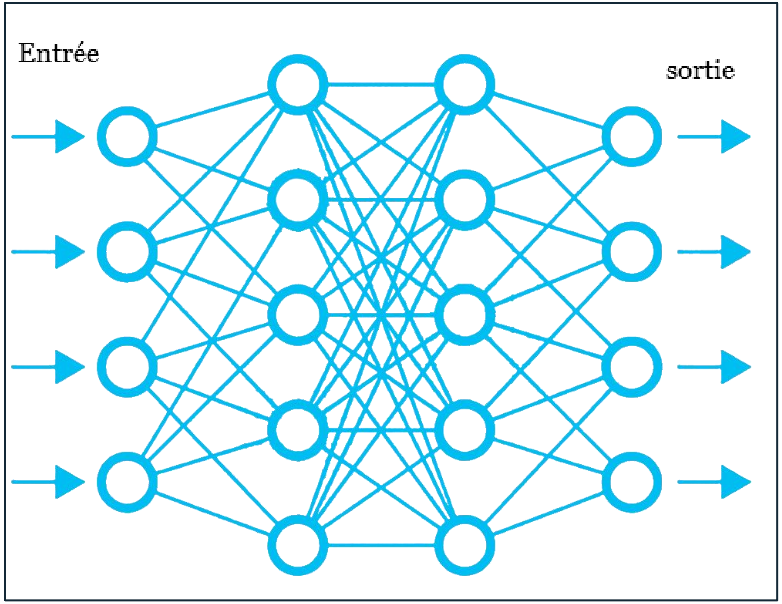


Figure 2 : Exemple de
Réseau de neurones

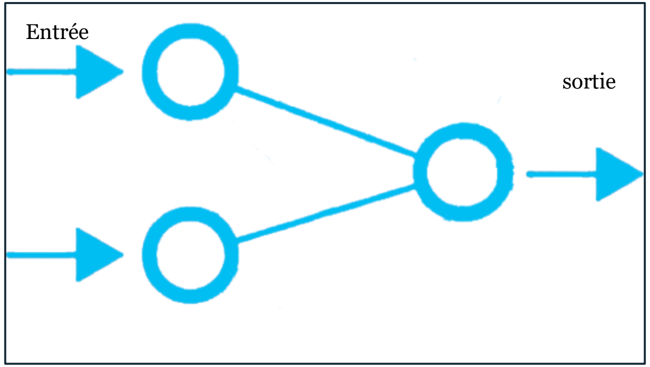
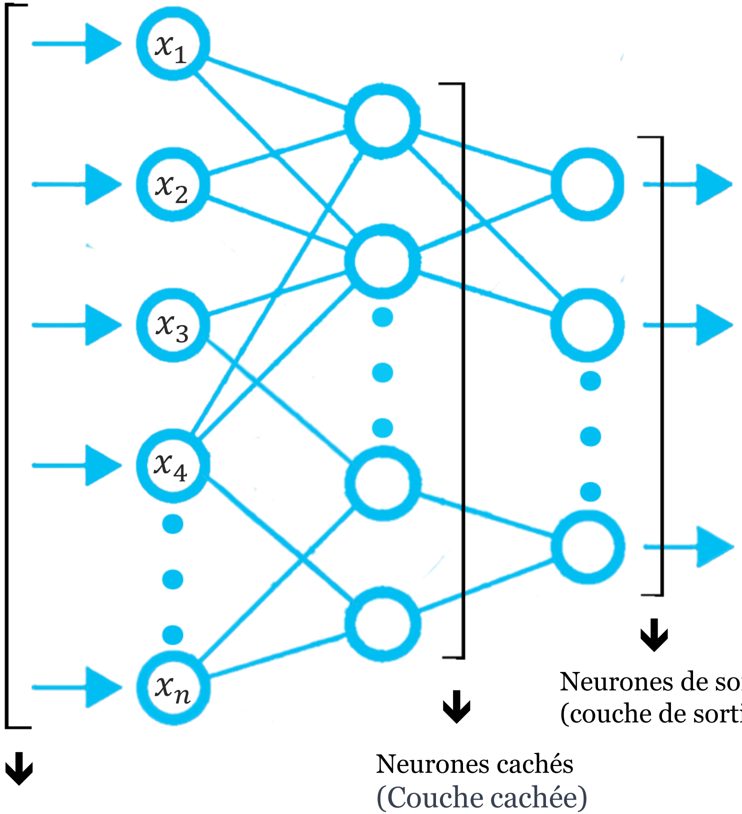


Figure 1 : un neurone
(perceptron)

Réseau de neurones



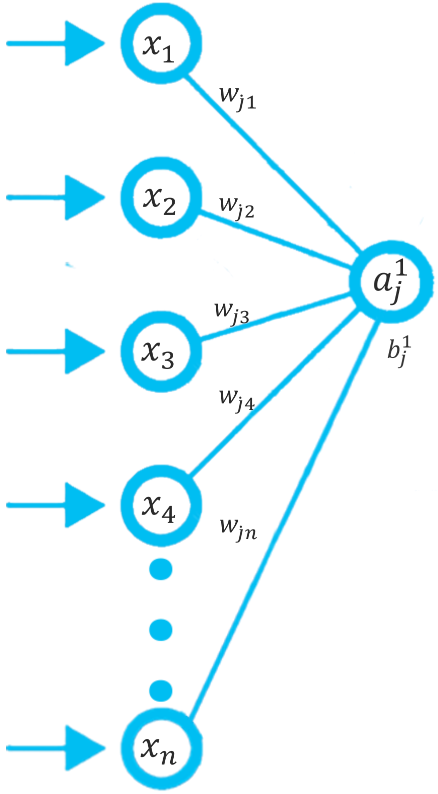
Neurones d'entrée
(couche d'entrée)

Neurones cachés
(Couche cachée)

Neurones de sortie
(couche de sortie)

x_i : l'entrée du i-ième neurone d'entrée.

Réseau de neurones



Exemple de la couche 1 avec a_j^1 (fonction d'activation le seuil)

$$a_j^1 = \begin{cases} 1 & si \sum_{k=1}^{n} w_{jk}.x_k \geq b_j^1 \\ 0 & si \sum_{k=1}^{n} w_{jk}.x_k \leq b_j^1 \end{cases}$$

$$\Rightarrow a_j^1 = \begin{cases} 1 & si\ w_j\ .x + b_j^1 \geq 0 \\ 0 & si\ w_j.x + b_j^1 \leq 0 \end{cases}$$

$w_j = (w_{j1},\ w_{j2},\ w_{j3} \ldots\ w_{jn})$, $x = (x_1,\ x_2,\ x_3 \ldots\ x_n)^t$

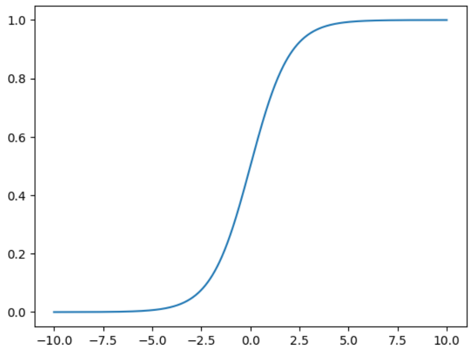
Réseau de neurones

Fonctions d'activation

- Sigmoïde

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

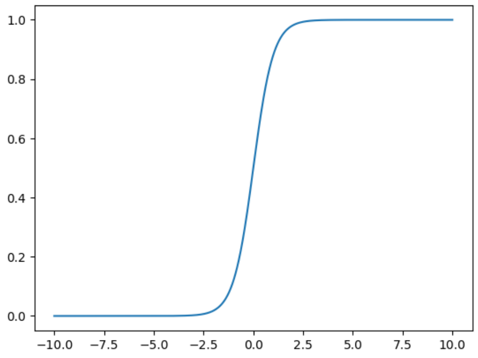
$$a_j^1 = \sigma(\, w_j^1 \cdot x + b_j^1)$$



- Tangente hyperbolique

$$\boxed{f(z) = \frac{\tanh(z)+1}{2}}$$

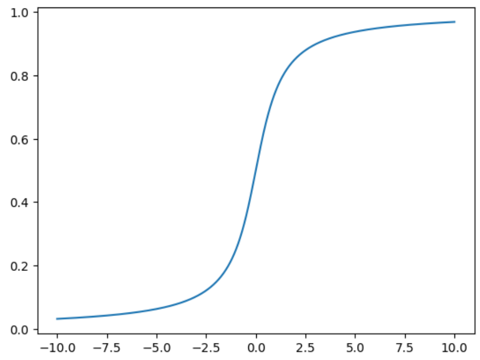
$$a_j^1 = f(\, w_j^1 \cdot x + b_j^1)$$



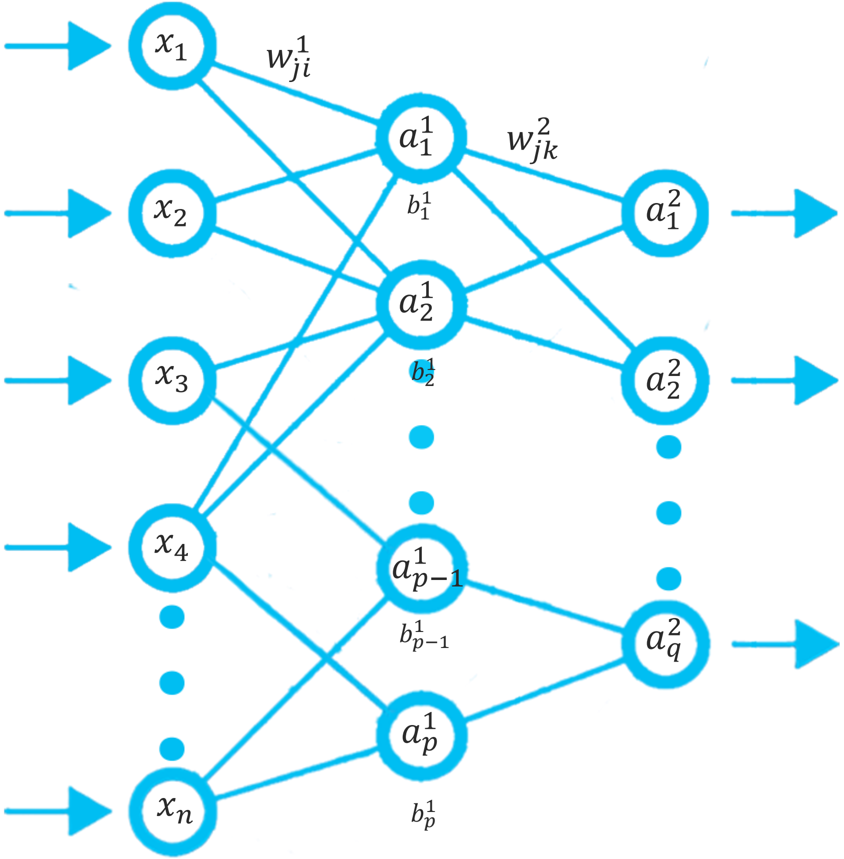
- arctangente

$$\boxed{f(z) = \frac{\arctan(z)+\pi/2}{\pi}}$$

$$a_j^1 = f(\, w_j^1 \cdot x + b_j^1)$$



Réseau de neurones



x_i : l'entrée du i-ième neurone d'entrée.

a_j^l : l'activation de la l-ième couche j-ième neurone.

b_i^l : le biais de la l-ieme couche j-ieme neurons

w_{jk}^l : le poids de la k-ième colonne reliant le neurone de la couche l-1 et le j-ième neurone de la l-ième couche.

Réseau de neurones

- Par récursivité on a $\forall l \in \{2 \dots, L\}$:

$$a^l = f(\cdot w^l a^{l-1} + b^l) \quad ; \qquad avec\ a^l \in M_{n,1}(\mathbb{R})$$

- la matrice des poids de la couche l :

$$w^l_{\cdot} = (w^l_{jk})_{jk}$$

- la matrice des biais de la couche l :

$$b^l_{\cdot} = (b^l_j)_j$$

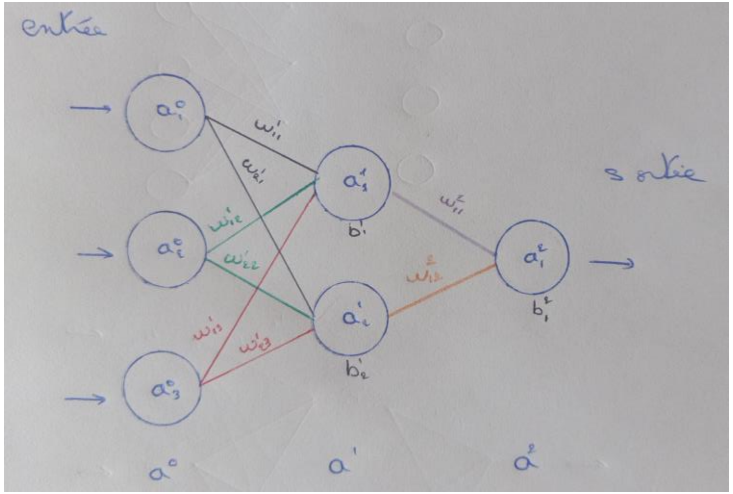
Réseau de neurones

```python
import numpy as np

class ReseauNeural(object):

    def __init__(self, tailles):
        self.num_layers = len(tailles)
        self.tailles = tailles
        self.biais = [np.random.randn(y, 1) for y in tailles[1:]]
        self.poids = [np.random.randn(y, x) for x, y in zip(tailles[:-1], tailles[1:])]

    def propagation_directe(self, a,f):
        for b, w in zip(self.biais, self.poids):
            a = f(np.dot(w, a)+b)
        return a
```

Implémentation sous python :réalisation d'un réseau neuronal avec des poids et des biais initiaux génères de manière aléatoire

Réseau de neurones

<u>Exemple d'un réseau de neurones de la forme [3,2,1]</u>



$\Rightarrow$

Implémentation sous python

```python
Reseau=ReseauNeural([3,2,1])
```

Avec les poids et les biais vont générer aléatoirement: (on prend $a^0 = (1,1,1)^t$)

$$\begin{cases} w^1 = \begin{pmatrix} 0.32 & -1.29 & 0.82 \\ -1.88 & 0.95 & 1.15 \end{pmatrix}, w^2 = (1.98\,, 2.58) \\ b^1 = \begin{pmatrix} -1.43 \\ -0.14 \end{pmatrix},\ b^2 = -0.44 \end{cases} \Rightarrow a^1 = \sigma\begin{pmatrix} -0.58 \\ -0.08 \end{pmatrix} = \begin{pmatrix} 0.35 \\ 0.48 \end{pmatrix} \Rightarrow a^2 = \sigma(1{,}49) = 0.81$$

2.

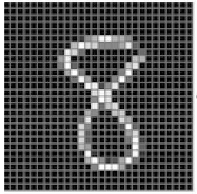
Optimisation des paramètres du réseau

la descente de gradient, descente stochastique de gradient , la rétropropagation du gradient …

2.1

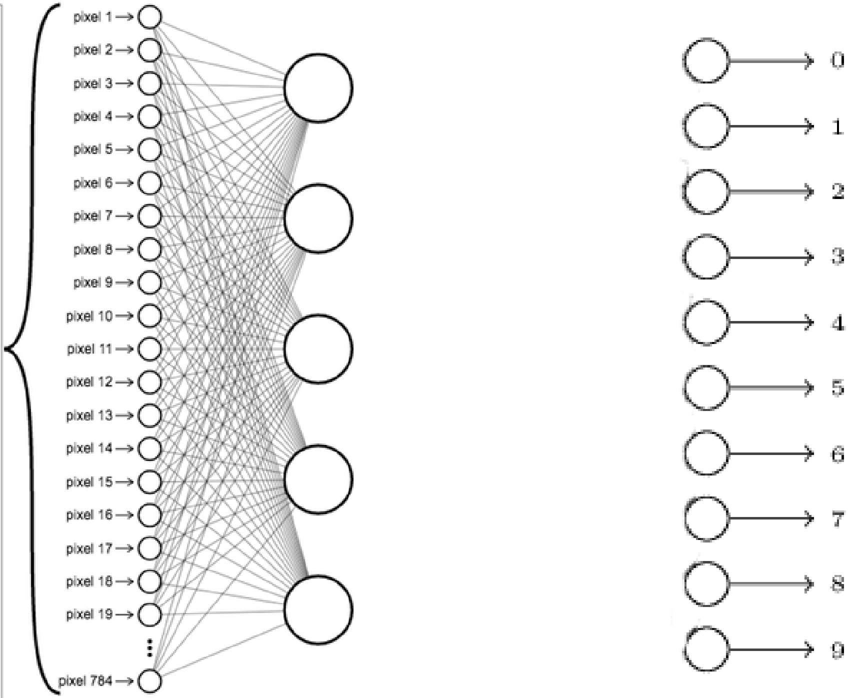
La descente de gradient

La descente de gradient



28 x 28
784 pixels

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	4	62	146	182	254	254	181	176	139	15	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	34	186	253	217	208	136	136	136	166	232	99	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	61	242	208	111	3	0	0	0	0	0	18	32	107	43	0	0	0	0	0	0
0	0	0	0	0	0	0	0	156	242	23	0	0	0	0	0	0	0	13	191	181	6	0	0	0	0	0	0
0	0	0	0	0	0	0	0	121	255	98	3	0	0	0	0	0	8	194	225	12	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	169	253	120	3	0	0	0	0	128	247	51	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	3	111	244	169	19	0	14	131	249	117	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	59	241	235	72	142	229	66	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	25	218	254	231	36	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	133	253	221	33	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	19	237	111	196	217	19	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	174	138	0	23	193	204	18	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	96	224	0	0	0	25	218	169	3	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	215	138	0	0	0	0	86	253	99	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	215	97	0	0	0	0	3	162	214	11	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	215	97	0	0	0	0	0	118	253	68	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	185	157	0	0	0	0	0	40	254	98	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	50	244	61	0	0	0	0	112	244	58	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	174	251	142	59	83	167	244	111	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	6	133	253	253	253	169	61	3	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



La descente de gradient

- Entraînement : L'objectif de l'entraînement consiste à optimiser les paramètres et les restrictions du réseau afin d'améliorer sa capacité à identifier plus efficacement les images.
- On définit le coût d'une image x par: $C_x(w,b) = \frac{\|y-a\|^2}{2} = \frac{1}{2}\sum_k (y_k - a_k)^2$
- On définit la fonction coût d'un ensemble d'entraînement de taille n par :

$$C(w,b) = \frac{1}{n}\sum_x C_x = \frac{1}{2n}\sum_x \|y-a\|^2$$

$y = (0,0,0,0,0,0,0,0,1,0)^t$ ($\Leftrightarrow 8$)

$a = \left(a_0^L, a_1^L, a_2^L, a_3^L, a_4^L, a_5^L, a_6^L, a_7^L, a_8^L, a_9^L\right)^t$

$x = (x_1, x_2, x_3, \dots, x_{784})^t$

$w = ((w^1), (w^2), \dots, (w^L))$

$b = ((b^1), (b^2), \dots, (b^L))$

$*$ *L soit le nombre des couches*

La descente de gradient

- Objectif : minimiser la fonction coût.

$$C(v+h) = C(v) + \langle \nabla C \mid h \rangle + o(\|h\|)$$

Alors $\Delta C \approx \langle \nabla C \mid \Delta v \rangle \qquad (E1)$

Tout D'abord on veut minimiser C

D'après l'inégalité de Cauchy-Schwarz : $|\Delta C| \approx |\langle \nabla C \mid \Delta v \rangle| \leq \|\nabla C\| \times \|\Delta v\|$

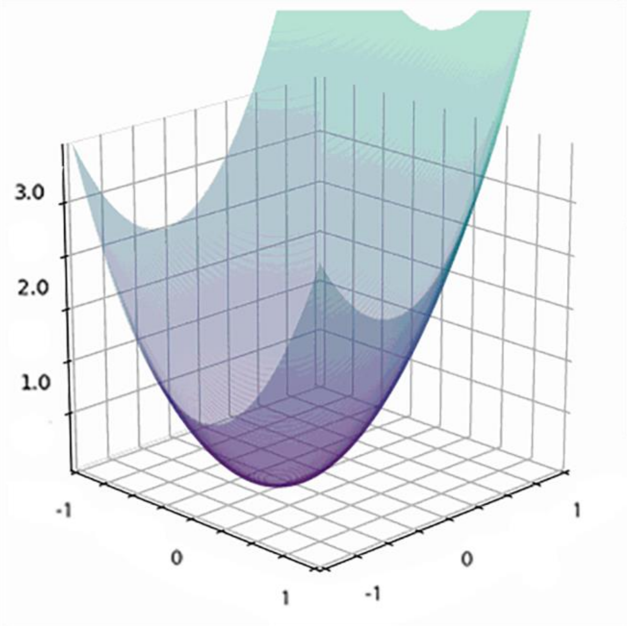
Donc $|\Delta C| \leq \|\nabla C\| \times \|\Delta v\|$

On prend $\Delta v = -\eta \nabla C$, avec $\eta > 0$

On aura $\Delta C \approx \langle \nabla C \mid \Delta v \rangle = \|\nabla C\| \, (-\eta \|\nabla C\|) = -\eta \|\nabla C\|^2$

$$\boxed{\Delta C \approx -\eta \|\nabla C\|^2 \leq 0}$$

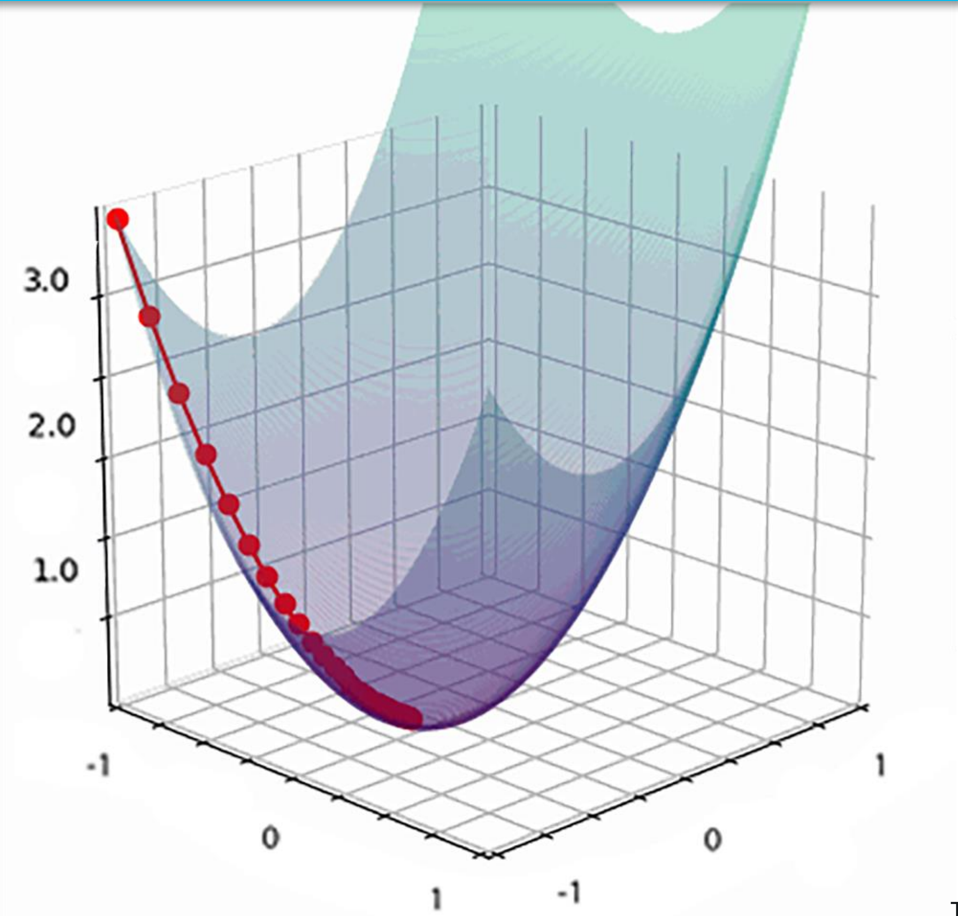
Ce qui assure le décroissement de la fonction coût



La descente de gradient

Donc : $\Delta v = -\eta \nabla C$ (E2)

$v \rightarrow v' = v - \eta \nabla C$ (E3)



La descente de gradient

- Reformulons la règle de mise à jour de la descente du gradient:

$$w_j \rightarrow w_j' = w_j - \eta \frac{\partial C}{\partial w_j} \qquad \text{(E4)}$$

$$b_j \rightarrow b_j' = b_j - \eta \frac{\partial C}{\partial b_j} \qquad \text{(E5)}$$

2.2

La rétropropagation du gradient

La rétropropagation du gradient

- On définit l'entrée pondérée par :

$$\boxed{z^l = w^l a^{l-1} + b^l}$$

- Vectorisation d'une fonction : $f(t_i) = f(t)_i$

$$\Rightarrow \quad a^l = f\,(z^l)$$

- Produit de Hadamard : $\mathrm{s} \odot \mathrm{t} = (s_i t_i)_\mathrm{i}$

- l'erreur δ_j^l par:

$$\boxed{\delta_j^l = \frac{\partial C}{\partial z_j^l}} \qquad (E6)$$

La rétropropagation du gradient

- Les quatre équations fondamentales de la rétropropagation: (Voir annexe pour la preuve)

$$\delta^L = f'(z^L) \odot \nabla_a C \qquad (RP1)$$

$$\delta^l = f'(z^l) \odot ((w^l)^t \delta^{l+1}) \qquad (RP2)$$

$$\delta_j^l = \frac{\partial C}{\partial b_j^l} \qquad (RP3)$$

$$a_k^{l-1} \delta_j^l = \frac{\partial C}{\partial w_{jk}^l} \qquad (RP4)$$

2.3

Processus de rétropropagation de gradient

Processus de rétropropagation du gradient

1. Entrer un ensemble d'exemples d'entrainement
2. Pour chaque exemple d'entraînement : Régler l'activation de l'entrée correspondante $a^{x,0}$, et effectuer les étapes suivantes:

   - Propagation directe : Pour tout $l = 1,2,3,\ldots,L$ calculer $z^{x,l} = w^{x,l} \cdot a^{x,l-1} + b^{l}$, $a^{x,l} = f\,(z^{x,l})$
   - Erreur de sortie $\delta^{x,L}$: Calculer le vecteur $\delta^{L} = f'(z^{L}) \odot \nabla_{\mathrm{a}} C$
   - Rétropropager l'erreur pour tout $l = L-1, L-2, \ldots, 2$ calculer

$$\delta^{l} = f'(z^{l}) \odot ((w^{l})^{t} \delta^{l+1})$$

3. Descente du gradient $l = L-1, L-2, \ldots, 2$ mettre à jour les poids selon la règle :

$$w^{l} \rightarrow w^{l} - \eta \delta^{x,l} \left(a^{x,l-1}\right)^{t}$$

Et les biais selon la règle :

$$b^{l} \rightarrow b^{l} - \eta \delta^{x,l}$$

Processus de rétropropagation du gradient

Pour l'entraînement, il est nécessaire de disposer des milliers nombre d'images représentant des chiffres.

Solution : utilisation de la base de données MNIST

- Un ensemble d'images de chiffres manuscrits allant de 0 à 9.
- Il est constitué de 60 000 images d'apprentissage et de 10 000 images de test, toutes en niveaux de gris et de taille 28x28 pixels.
- Chaque image est étiquetée avec le chiffre correspondant,

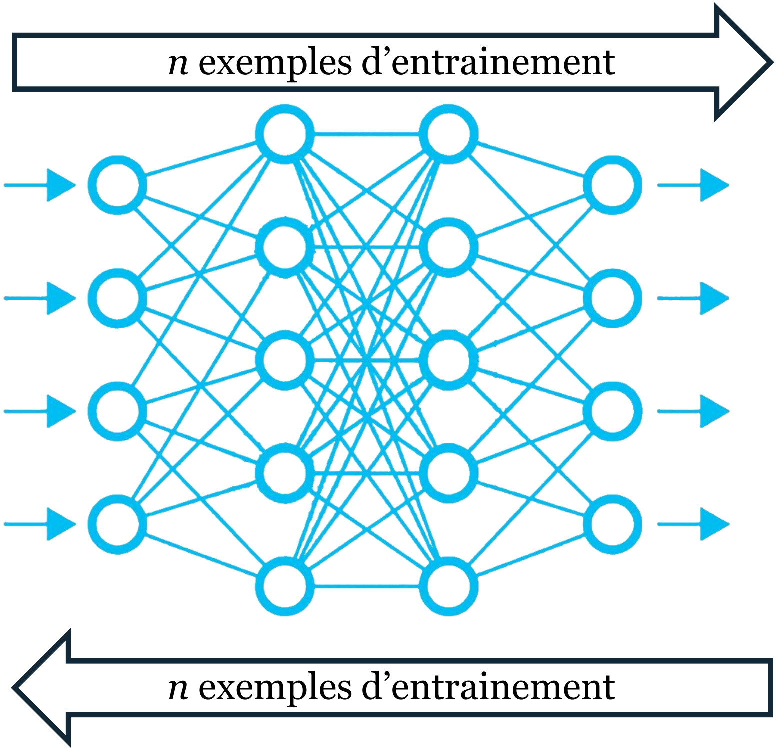
3.

Analyse de l'influence des paramètres du réseau sur l'apprentissage.

Analyse de l’influence des paramètres du réseau sur l’apprentissage

- Époques (epochs) : une passe avant et une passe arrière de tous les exemples d’entrainement.

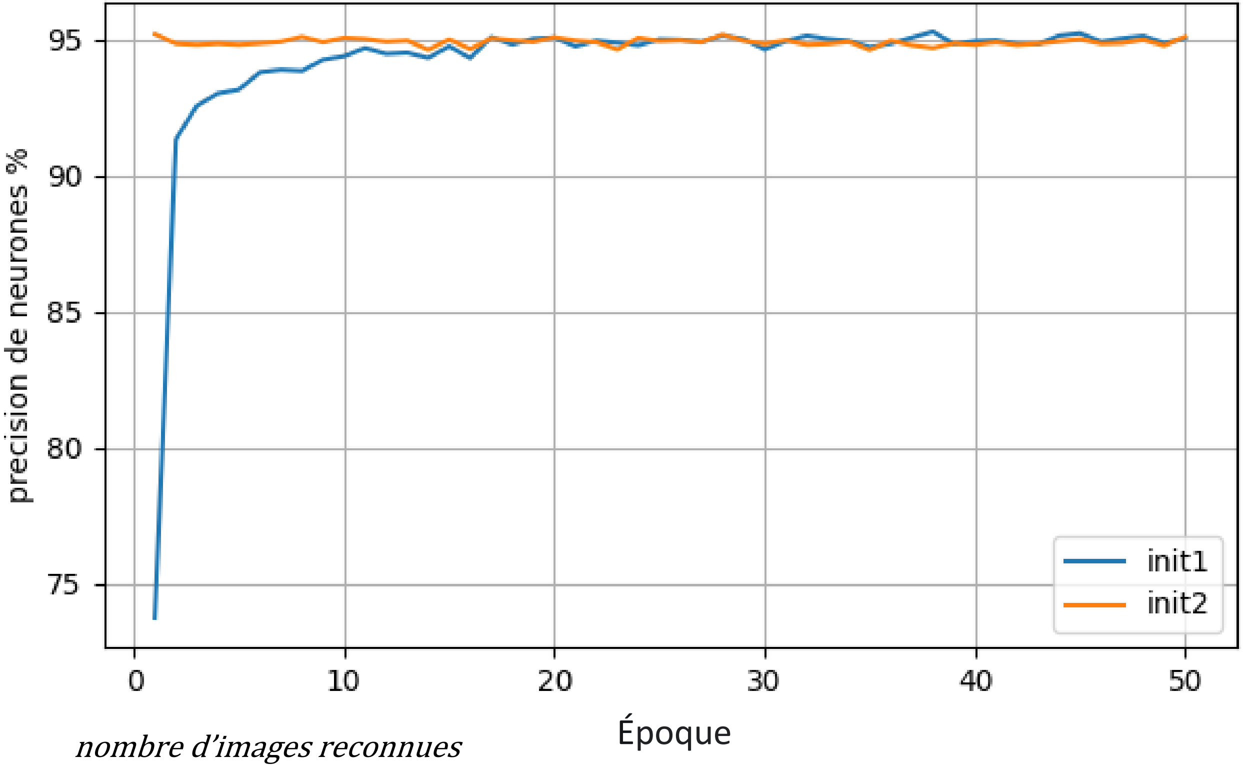
Exemples d’entrainement totale=n



- Après la première époque, les poids et les biais seront décents.
- En réintroduisant les données d'apprentissage dans le réseau de neurones, nous pouvons améliorer encore les poids et les biais

Analyse de l'influence des paramètres du réseau sur l'apprentissage

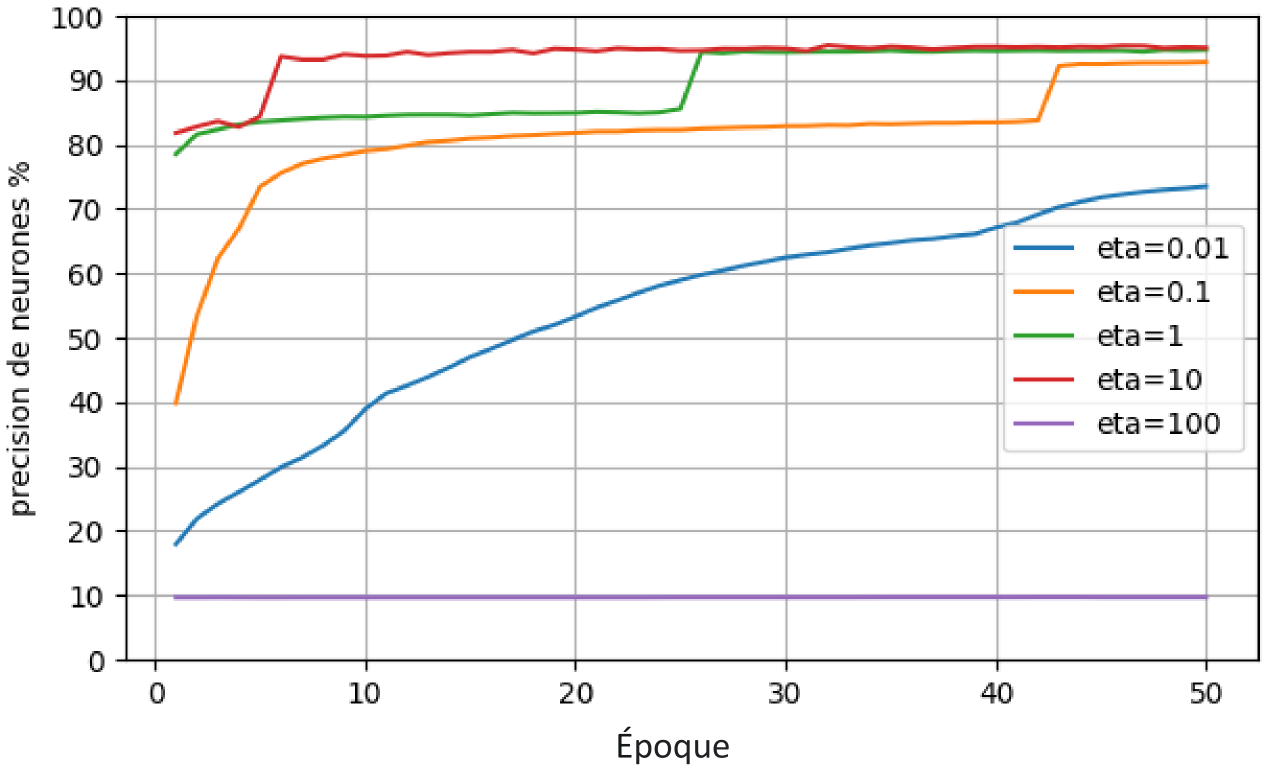
<u>La valeur initial:</u>



$$\text{Précision de neurones} = \frac{\text{nombre d'images reconnues}}{\text{nombre total d'images testées}}$$

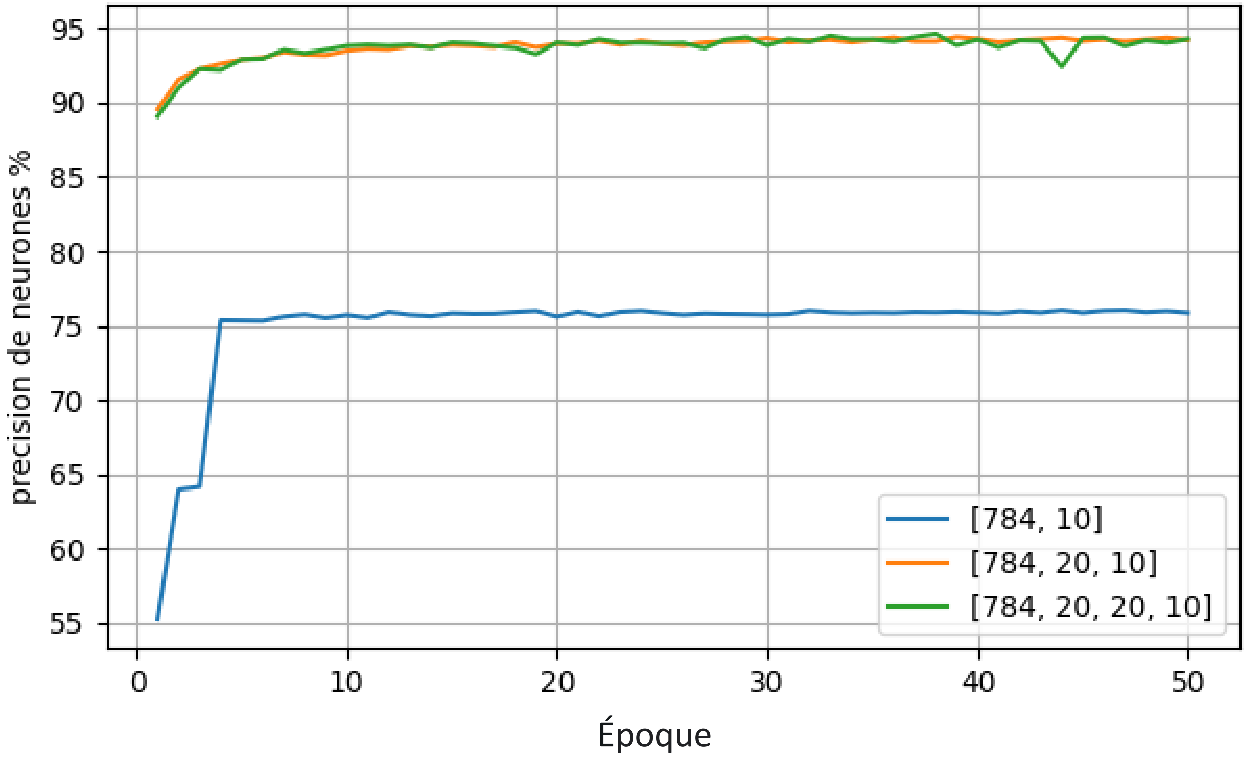
Analyse de l'influence des paramètres du réseau sur l'apprentissage

<u>Le taux d'apprentissage η:</u>



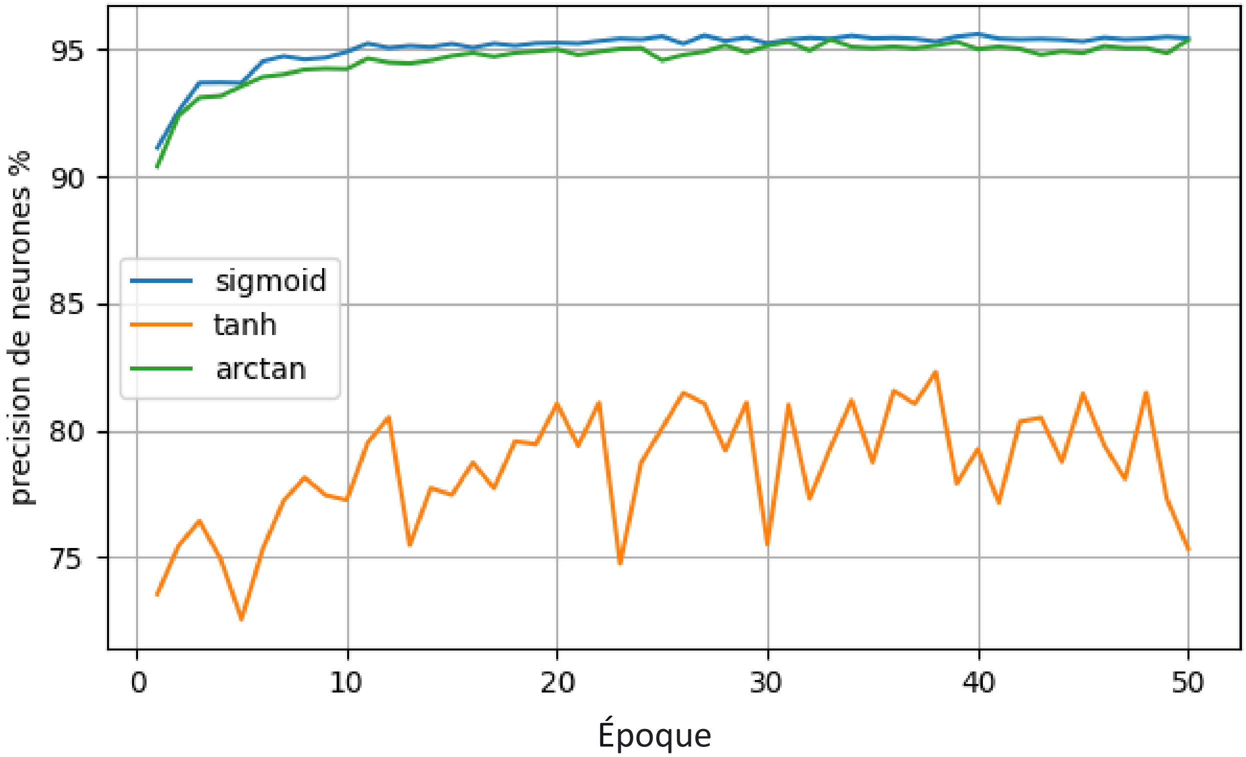
Analyse de l'influence des paramètres du réseau sur l'apprentissage

La structure du réseau:



Analyse de l’influence des paramètres du réseau sur l’apprentissage

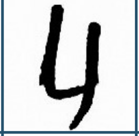
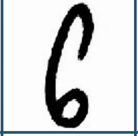
La fonction d’activation:



5.

Evaluation et limites du model.

Evaluation et limites du model

personne	Ecritures manuscrits									
A										
	0	4	8	3	4	5	6	7	8	9
I										
	9	1	1	3	4	8	6	7	9	3
M										
	0	1	8	3	4	5	4	7	1	7

Annexe

- *Annexe 1: implémentation d’un réseau de neurones (‘ network.py’).*
- *Annexe 2: entrainement du réseau et analyse des paramètres (‘execution.ipynb’).*
- *‘Annexe 3: reconnaissance et évaluation des chiffres manuscrits (‘execution.ipynb’).*
- *Preuve des équations (RP1), (RP2), (RP3), (RP4) (page 16).*

MERCI POUR VOTRE ATTENTION.

Annexe 1

```python
import numpy as np

class ReseauNeural(object):

    def __init__(self, tailles):
        self.num_layers = len(tailles)
        self.tailles = tailles
        self.biais = [np.random.randn(y, 1) for y in tailles[1:]]
        self.poids = [np.random.randn(y, x) for x, y in zip(tailles[:-1], tailles[1:])]

    def propagation_directe(self, a,f):
        for b, w in zip(self.biais, self.poids):
            a = f(np.dot(w, a)+b)
        return a

    def Descente_gradient(self, donnees_entrainement, epochs, taille_mini_lot, eta,f,f_prime,
            donnees_test=None):
        L=[]
        n_test = len(donnees_test)
        n = len(donnees_entrainement)
        for j in range(1,epochs+1):
            np.random.shuffle(donnees_entrainement)
            mini_lots = [ donnees_entrainement[k:k+taille_mini_lot] for k in
                          range(0, n, taille_mini_lot)]
            for mini_lot in mini_lots:
                self.mettre_a_jour(mini_lot, eta,f,f_prime)
```

```python
            if donnees_test:
                print(f"epoque {j}: {self.evaluation(donnees_test,f)} / {n_test}")
                L.append([j,self.evaluation(donnees_test,f)/ n_test*100])
            else:
                print(f"epoque {j} est completé")

        return L


    def mettre_a_jour(self, mini_lot, eta,f,f_prime):
        nabla_b = [np.zeros(b.shape) for b in self.biais]
        nabla_w = [np.zeros(w.shape) for w in self.poids]
        for x, y in mini_lot:
            delta_nabla_b, delta_nabla_w = self.retroprop(x, y,f,f_prime)
            nabla_b = [nb+dnb for nb, dnb in zip(nabla_b, delta_nabla_b)]
            nabla_w = [nw+dnw for nw, dnw in zip(nabla_w, delta_nabla_w)]
        self.poids = [w-(eta/len(mini_lot))*nw for w, nw in zip(self.poids, nabla_w)]
        self.biais = [b-(eta/len(mini_lot))*nb for b, nb in zip(self.biais, nabla_b)]

    def retroprop(self, x, y,f,f_prime):
        nabla_b = [np.zeros(b.shape) for b in self.biais]
        nabla_w = [np.zeros(w.shape) for w in self.poids]
        activation = x
        activations = [x] # liste pour stocker toutes les activations, couche par couche
        zs = [] # liste pour stocker tous les vecteurs z, couche par couche
        #passage en avant (calculer les activations correspondant à x)
```

```python
        for b, w in zip(self.biais, self.poids):
            z = np.dot(w, activation)+b
            zs.append(z)
            activation = f(z)
            activations.append(activation)
        # passage en arrière
        delta = self.derivee_cout(activations[-1], y) * f_prime(zs[-1])
        nabla_b[-1] = delta
        nabla_w[-1] = np.dot(delta, activations[-2].transpose())
        for l in range(2, self.num_layers):
            z = zs[-l]
            sp = f_prime(z)
            delta = np.dot(self.poids[-l+1].transpose(), delta) * sp
            nabla_b[-l] = delta
            nabla_w[-l] = np.dot(delta, activations[-l-1].transpose())
        return (nabla_b, nabla_w)

    def evaluation(self, donnees_test,f):
        resultat_test = [(np.argmax(self.propagation_directe(x,f)), y)
                        for (x, y) in donnees_test]
        return sum(int(x == y) for (x, y) in resultat_test)

    def derivee_cout(self, activation_sortie, y):
        return (activation_sortie-y)

```

Annexe 1

```python
def sigmoid(z):
    return 1.0/(1.0+np.exp(-z))
def sigmoid_prime(z):
    return sigmoid(z)*(1-sigmoid(z))
def tanh(z):
    return (np.tanh(z)+1)/2
def tanh_prime(z):
    return (1-tanh(z)**2)/2
def arctan(z):
    return np.arctan(z)/np.pi+0.5
def arctan_prime(z):
    return (1/(1+z**2))/(np.pi)
```

Annexe 2

```python
import mnist_loader #pour la resolution du mnist data set
import network as n
donnees_entrainement, validation_data, donnees_test = mnist_loader.load_data_wrapper()

net = n.ReseauNeural([784,30,10])
def entrainement(data,net,epochs,taille_mini_lots,eta,f,f_prime,test=None):
    prog=net.Descente_gradient(data, epochs, taille_mini_lots, eta,f,f_prime, donnees_test=test)
    return prog
```

```python
import matplotlib.pyplot as plt
import numpy as np
from copy import copy #comparer les performances en conservant les valeurs initiales d'origine.

def graph(data_sets):#fonction pour le tracage des courbes
    for data_set, label in data_sets:
        x=[data[0] for data in data_set]
        y=[data[1] for data in data_set]
        plt.plot(x, y, label=label)
```

```python
#courbe pour differente val d'eta
eta001=entrainement(donnees_entrainement,copy(net),50,10,0.01,n.sigmoid,n.sigmoid_prime
                    ,test=donnees_test)
eta01=entrainement(donnees_entrainement,copy(net),50,10,0.1,n.sigmoid,n.sigmoid_prime
                   ,test=donnees_test)
eta1=entrainement(donnees_entrainement,copy(net),50,10,1,n.sigmoid,n.sigmoid_prime
                  ,test=donnees_test)
```

Annexe 2

```
eta10=entrainement(donnees_entrainement,copy(net),50,10,10,n.sigmoid,n.sigmoid_prime
                   ,test=donnees_test)
eta100=entrainement(donnees_entrainement,copy(net),50,10,100,n.sigmoid,n.sigmoid_prime
                   ,test=donnees_test)

matrices = [[eta001,'eta=0.01'], [eta01,'eta=0.1'], [eta1,'eta=1'], [eta10,'eta=10']
            , [eta100,'eta=100']]
graph(matrices)

plt.xlabel('Époque')
plt.ylabel('precision de neurones %')
plt.legend()
plt.grid()
plt.yticks(np.arange(0, 101, 10))
plt.gcf().set_size_inches(7,4)
plt.savefig("eta.png")
plt.show()
```

```
#courbes de differente structure du reseau

M=entrainement(donnees_entrainement,n.ReseauNeural([784,10]),50,10,3,n.sigmoid
                        ,n.sigmoid_prime,test=donnees_test)
M20=entrainement(donnees_entrainement,n.ReseauNeural([784,20 ,10]),50,10,3,n.sigmoid
                          ,n.sigmoid_prime,test=donnees_test)
M2020=entrainement(donnees_entrainement,n.ReseauNeural([784,20,20,10]),50,10,3,n.sigmoid
                            ,n.sigmoid_prime,test=donnees_test)
```

Annexe 2

```

data_sets = [[M, '[784, 10]'], [M20, '[784, 20, 10]'], [M2020, '[784, 20, 20, 10]']]
graph(data_sets)

plt.xlabel('Époque')
plt.ylabel('precision de neurones %')
plt.legend()
plt.grid()
plt.gcf().set_size_inches(7,4)
plt.savefig("couches.png")
plt.show()
```

```
#courbe pour differente fonction d'activation
net=n.ReseauNeural([784,30,10])
sigmoid=entrainement(donnees_entrainement,copy(net),50,10,3,n.sigmoid,n.sigmoid_prime
                              ,test=donnees_test)
tanh=entrainement(donnees_entrainement,copy(net),50,10,3,n.tanh,n.tanh_prime
                           ,test=donnees_test)
arctan=entrainement(donnees_entrainement,copy(net),50,10,3,n.arctan,n.arctan_prime
                             ,test=donnees_test)

data_sets = [[sigmoid, 'sigmoid'], [tanh, 'tanh'], [arctan, 'arctan']]
graph(data_sets)

plt.xlabel('Époque')
plt.ylabel('precision de neurones %')
```

Annexe 2

```
plt.legend()
plt.grid()
plt.gcf().set_size_inches(7,4)
plt.savefig("fonction.png")
plt.show()
```

```
#courbes de deux differentes valeurs initials
net1=n.ReseauNeural([784,30,10])
net2=n.ReseauNeural([784,30,10])
init1=entrainement(donnees_entrainement,net1,50,10,3,n.sigmoid,n.sigmoid_prime
                            ,test=donnees_test)
init2=entrainement(donnees_entrainement,net1,50,10,3,n.sigmoid,n.sigmoid_prime
                            ,test=donnees_test)

data_sets = [[init1, 'init1'], [init2, 'init2']]
graph(data_sets)

plt.xlabel('Époque')
plt.ylabel('precision de neurones %')
plt.legend()
plt.grid()
plt.gcf().set_size_inches(7,4)
plt.savefig("init.png")
plt.show()
```

Annexe 2

```python
#preparer une image pour l'evaluation
from PIL import Image, ImageFilter
import numpy as np
import matplotlib.pyplot as plt
def imageprepare(argv): #l'image est suppose carree de taille plus de 28*28 pixels
    im = Image.open(argv).convert('L')
    width = float(im.size[0])
    height = float(im.size[1])
    newImage = Image.new('L', (28, 28), (255))  # crée une toile blanche de 28x28 pixels

    nheight = int(round((20.0 / width * height), 0))  # resize height according to ratio width
    # redimensionner et affiner
    img = im.resize((20, nheight), Image.LANCZOS).filter(ImageFilter.SHARPEN)
    wtop = int(round(((28 - nheight) / 2), 0))  # calculer la position horisontale
    newImage.paste(img, (4, wtop))  # coller l'image redimensionee dans la toile blanche

    newImage.save("image.png")

    tv = list(newImage.getdata())  # obtenir les valeurs des pixels

    # normaliser les pixels de 0 à 1. 0 est un blanc pur, 1 est un noir pur.
    tva = [[(255 - x) * 1.0 / 255.0] for x in tv]
    return [[(255 - x) * 1.0 / 255.0] for x in tv]
```

Annexe 3

```python
#entrainer pour evaluer
net = n.ReseauNeural([784,30,10])
net.Descente_gradient(donnees_entrainement, 50, 10, 3,n.sigmoid,n.sigmoid_prime,
                      donnees_test=donnees_test)
```

```python
#evaluer une image
M0=imageprepare("C:\\Users\\mouad\\Desktop\\chiffres\\4M.jpg")
print(np.argmax(net.propagation_directe(M0,n.sigmoid)))
```

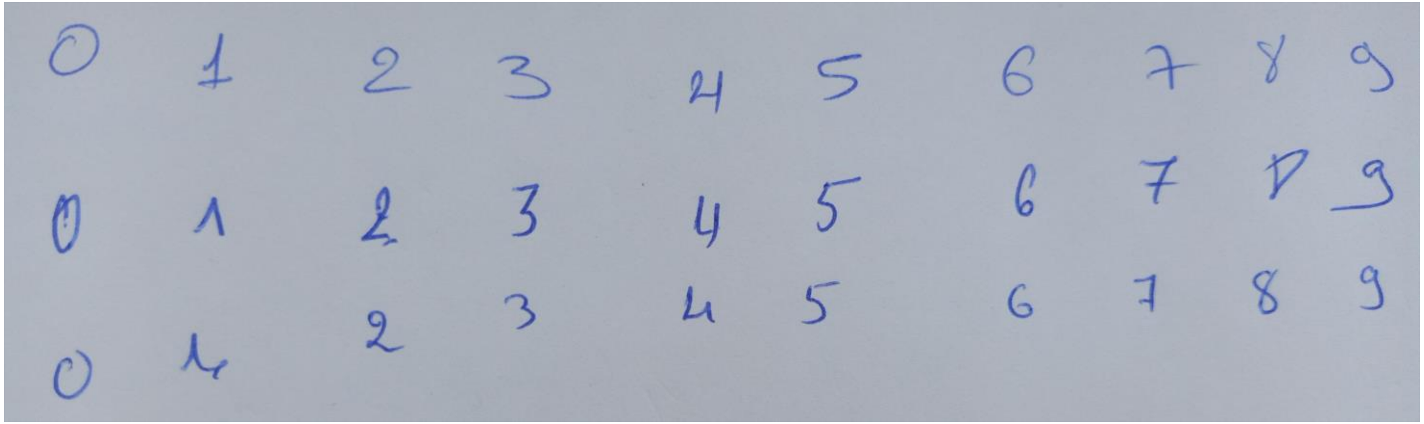
```python
#evaluation des ecritures manuscrites de 3 personnes
I,A,M=[],[],[]
for i in range(10):
    MA=imageprepare(f"C:\\Users\\mouad\\Desktop\\chiffres\\{str(i)}A.jpg")
    A+=[ np.argmax(net.propagation_directe(MA,n.sigmoid)) ]
    MI=imageprepare(f"C:\\Users\\mouad\\Desktop\\chiffres\\{str(i)}I.jpg")
    I+=[ np.argmax(net.propagation_directe(MI,n.sigmoid)) ]
    MM=imageprepare(f"C:\\Users\\mouad\\Desktop\\chiffres\\{str(i)}M.jpg")
    M+=[ np.argmax(net.propagation_directe(MM,n.sigmoid)) ]
print('A='+str(A)+'\n'+'I='+str(I)+'\n'+'M='+str(M))
```

Annexe 3

M

I

A



(RP1) :

D'après **la règle de la chaîne** : $$\delta_j^l = \frac{\partial C}{\partial z_j^l} = \sum_i \frac{\partial a_i^l}{\partial z_j^l}\frac{\partial C}{\partial a_i^l} = \frac{\partial a_j^l}{\partial z_j^l}\frac{\partial C}{\partial a_j^l} = f'\left(z_j^l\right)\frac{\partial C}{\partial a_j^l}$$

Donc (RP1) $$\delta^L = f'(z^L)\odot\nabla_a C$$

(RP2) :

D'après **la règle de la chaîne** :

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} = \sum_i \frac{\partial z_i^{l+1}}{\partial z_j^l}\frac{\partial C}{\partial z_i^{l+1}} = \sum_i \frac{\partial z_i^{l+1}}{\partial z_j^l}\delta_i^{l+1} = \sum_i f'\left(z_j^l\right)w_{ij}^{l+1}\delta_i^{l+1}$$

Car $$\frac{\partial z_i^{l+1}}{\partial z_j^l} = w_{ij}^{l+1}\, f'\left(z_j^l\right)$$

Donc $$\delta^l = f'(z^L)\odot((w^l)^t\delta^{l+1})$$

(RP3) :

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} = \frac{\partial b_j^l}{\partial z_j^l}\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial b_j^l}$$

(RP4):

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial z_j^l}{\partial w_{jk}^l}\frac{\partial C}{\partial z_j^l} = a_k^{l-1}\delta_j^l$$