Reconnaissance d'images

TIPE mathématiques appliquées et informatique pratique

No-one

08 juin 2021

Sommaire

Motivation

Figure – 1

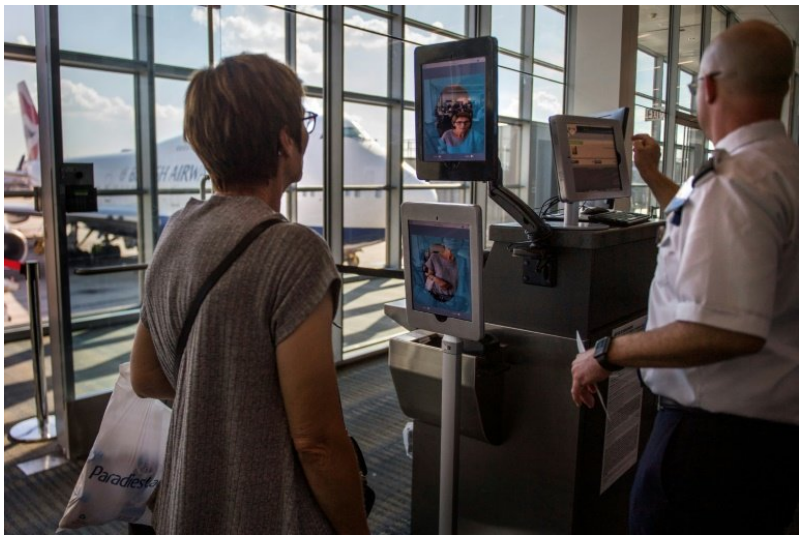


Figure – 2

- figure 1 : Système d'identification de visage des nouveaux smartphones
- figure 2 : Système de verification d'identité à l'aéroport international de Washington-Dulles
- Enjeux : Faciliter les tâches humaines ; plus de fluidité et rapidité.

Motivation

Figure – 3

Figure – 4

- Expérience de système reconnaissance faciale à Nice en fevrier 2019.
- Enjeux : détection de comportement suspect, mouvement de foule, personnes dangereuses, manifestations, enfants perdus. . .

Problématique

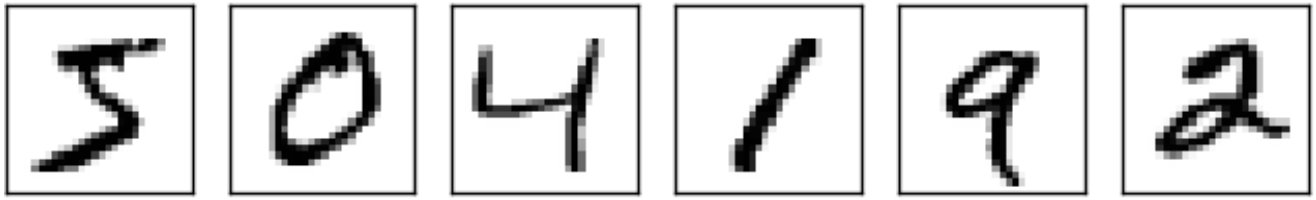


Figure – Images de chiffres écrits à la main

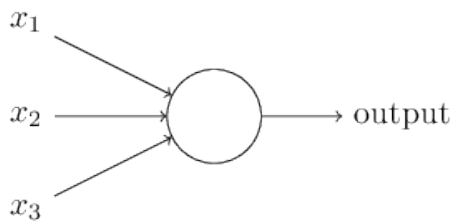
- L'oeil humain peut aisément reconnaître les images de chiffres, cette abélité à distinguer entre les images est exceptionnelle et unique.
- **Objectif :** Écrire un algorithme capable à reconnaître les chiffres manuscrits.

Réseau de neurones

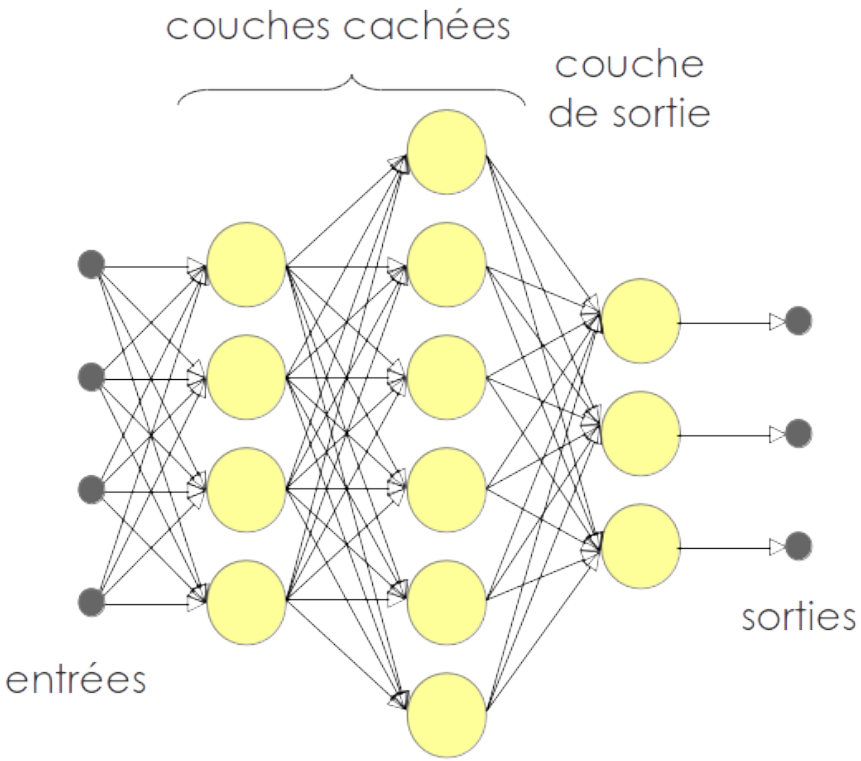
- **Exemple** : lors de la prise d'une décision (exemple achat d'un appartement) : Plusieurs paramètres (x_k= Entrée) influencent notre décision.
  - La localisation : Distance qui sépare l'appartement et le lieu de travail
  - moyens de transport (métro,gare...)
  - le prix ...
- Chaque paramètres a son importance (poids w_1 ,w_2...)
- On calcule la somme $\sum_j w_j x_j$

$$\text{Sortie} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{contrainte} \\ 1 & \text{if } \sum_j w_j x_j > \text{contrainte} \end{cases} \quad (1)$$

- On peut représenter le calcul fait au-dessus par un réseau de neurone basique (voir schéma)



Réseau de neurones

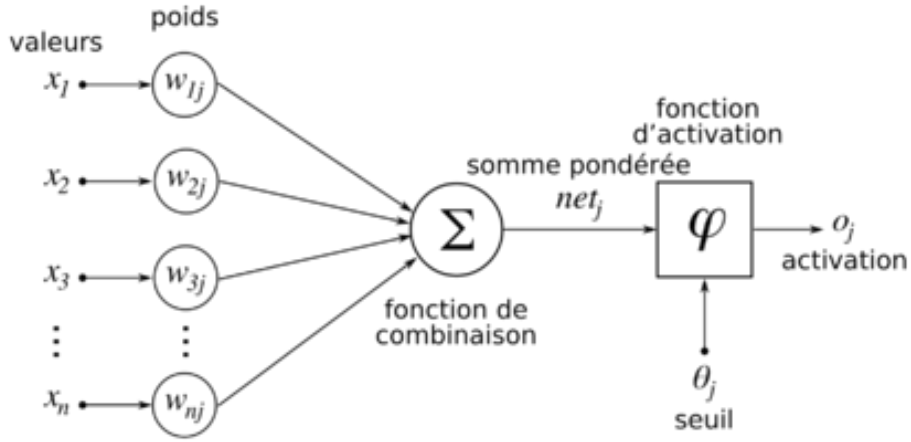


Un réseau de neurones (artificiels) est un modèle informatique dont la concéption est inspirée du fonctionnement des neurones biologiques.

Réseau de neurones

- Un réseau de neurones est constitué de :
  - **Couches :** La première couche est l'entrée et la dernière représente la sortie. (L : le nombre de couches ; $a^{(l)}$ la l'ième couche)
  - **Neurones :** Ils représentent les noeuds de chaque couches, ainsi le jème neurone de la l'ième couche est $a_j^{(l)}$
  - **Poids :** Ils permettent de passer d'une couche à la suivante ; On note $w^{(l)}$ la matrice des poids de passage de la l'ième couche à la suivante.
  - **Contraintes (seuils)** : $b^{(l)} = Vect(b_i^{(l)})$
  - **Fonction d'activation f :** C'est une fonction de $\mathbb{R}$ dans [0,1] . Exemple : la fonction Sigmoïde $\sigma(z) = \frac{1}{1+e^{-z}}$

Réseau de neurones



- On fournit au réseau une entrée sous forme d'un vecteur (dans notre exemple c'est un vecteur de dimension 784=28*28 qui représente une image 24*24 pixels en mode gris d'un chiffre)
- **Pour passer d'une couche à la suivante :**
  - $\forall l\ \forall i\quad a_i^{(l)} = f(\sum_j w_{ij}^{(l)} a_j^{(l-1)} + b_i^{(l)})$
  - il est équivalent à multiplier le vecteur par la matrice w :

$$\boxed{a^{(l)} = f(w^{(l)} \cdot a^{(l-1)} + b^{(l)})} \tag{2}$$

Remarque si $z = Vect(z_i)$ alors $f(z) = Vect(f(z_i))$

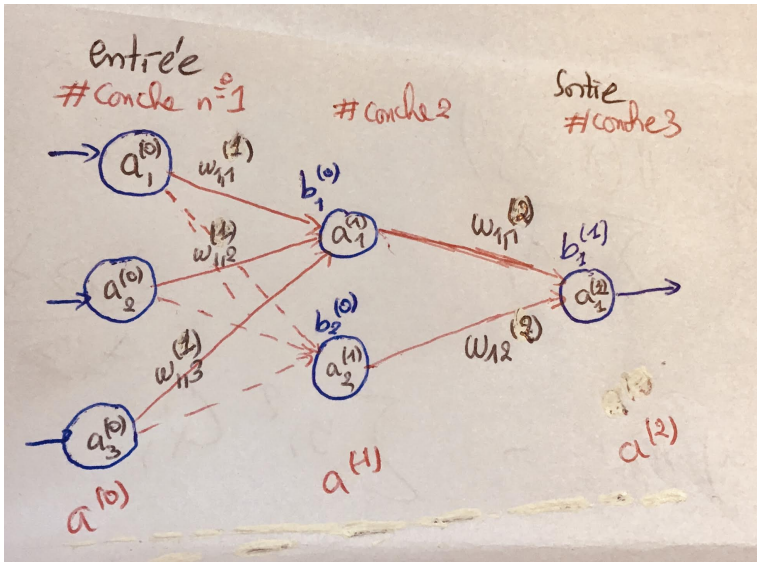
Réseau de neurones

Exemple : Réseau de neurones L=3 de poids et contraintes suivants :

$$w^{(1)} = \begin{pmatrix} 3.1 & 4 & 2 \\ 1.2 & 3 & -3 \end{pmatrix},$$

$$\mathrm{w}^{(2)} = \begin{pmatrix} -2.0 & 1.5 \end{pmatrix}$$

$$\mathrm{b}^{(1)} = \begin{pmatrix} -9 \\ -4.25 \end{pmatrix}, \; b^{(2)} = -2.9$$



- Pour une entrée $a^{(0)} = \begin{pmatrix} 0.5 \\ 0.7 \\ 0.2 \end{pmatrix}$ alors

$$a^{(1)} = \sigma(w^{(1)} \cdot a^{(0)} + b^{(1)}) = \begin{pmatrix} \sigma(-4.5) \\ \sigma(-2.5) \end{pmatrix} = \begin{pmatrix} 0.014 \\ 0.104 \end{pmatrix}$$

et en sortie $a^{(2)} = \sigma(w^{(2)} \cdot a^{(1)} + b^{(2)}) = (\sigma(-2.77)) = (0.058)$

Réseau neurones

```
1  class Network(object):
2      """tailles est une liste qui contient la taille de
       chaque couche. Le nombre de couches est len(tailles)"""
3      def __init__(self, tailles):
4          self.nb_tailles = len(tailles)
5          self.tailles = tailles
6          self.contraintes = [np.random.randn(x,1)
7                              for x in tailles[1:] ]
8          self.poids = [ np.random.randn(y,x)
9                      for x,y in zip(tailles[:-1],tailles[1:])
       ]
10
11     """Calcul prend en argument un vecteur de taille
       784=28*28 qui represente notre image de 28*28 pixels """
12
13     def calcul(self,a,f):
14         for b,w in zip(self.contraintes, self.poids):
15             a = f(np.dot(w,a)+b)
16         return a
```

Listing 1 – implémentation d'un réseau initialisé par des poids et contraintes choisies aléatoirement

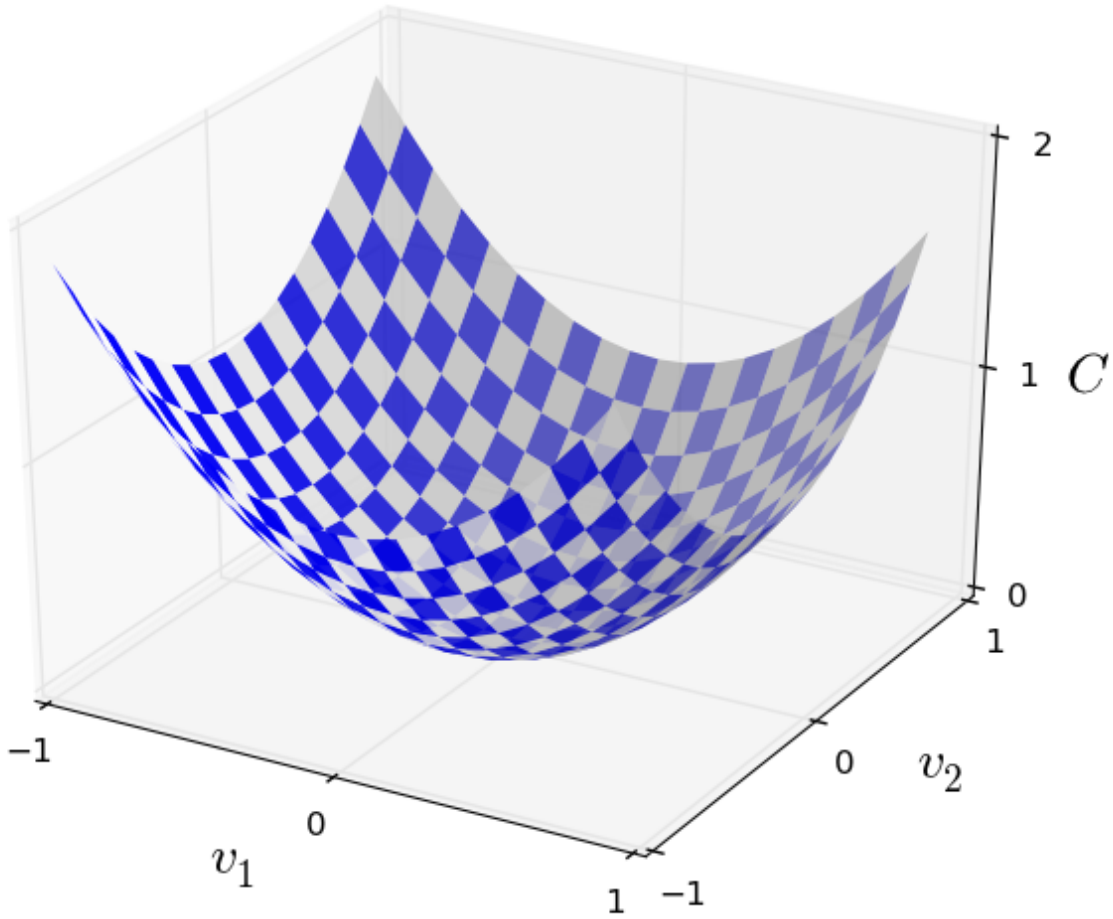
Entrainement et algorithme du gradient

- **Entrainement :** le but de l'entrainement est d'améliorer les poids et les contraintes du réseau àfin de mieux reconnaitre les images.
- Pour l'entrainement, on possède 50 000 couples (x,y) où x est une image du chiffre y .(Rq : si x représente le chiffre 0 alors y=$(1,0,0,0,0,0,0,0,0,0)^T$ et si il représente 9 alors y = $(0,0,0,0,0,0,0,0,0,1)^T$).
- On note a la réponse du réseau à l'entrée x. On definit le cout d'une image par $C_x = \frac{||a-y||^2}{2}$
- le cout d'un entrainement de taille n est donné par :

$$C(w,b) \equiv \frac{1}{n}\sum_x C_x \equiv \frac{1}{2n}\sum_x \|a(x)-y(x)\|^2. \tag{3}$$

- **Objectif :** Minimiser la fonction C (Qui ne dépend que des poids et des contraintes, à condition que n soit suffisament grand) en modifiant les poids et les contraintes.

Entrainement et algorithme du gradient



Entrainement et algorithme du gradient

$$C(v+h) = C(v) + \langle \nabla C(x), h \rangle + o(\|h\|)$$
$$\Delta C = \langle \nabla C(v), \Delta v \rangle + o(\|\Delta v\|)$$
$$\Delta C \simeq \langle \nabla C(v), \Delta v \rangle$$

- D'àprès l'inégalité de Cauchy-Schwartz

$$|\Delta C(v)| \simeq |\langle \nabla C(v), \Delta v \rangle| \leq \|\nabla C(v)\| \dot{\|} \Delta v\|$$

- Donc pour minimiser C le plus possible, il faut choisir $\Delta v = -\eta \nabla C(v)$ avec $\eta > 0$

$$\Delta C \simeq -\eta \dot{\|} \nabla C(v)\|^2 \leq 0 \quad (4)$$
$$\Delta v = -\eta \nabla C(v) \quad (5)$$

- la relation (5) va nous permettre de modifier les poids et les contraintes

Entrainement et algorithme du gradient

- On a directement de la relation (5)

$$w_{kj}^{(l)} \leftarrow w_{kj}^{(l)} - \eta \frac{\partial C}{\partial w_{kj}^{(l)}} \tag{6}$$

$$b_k^{(l)} \leftarrow b_k - \eta \frac{\partial C}{\partial b_k^{(l)}} \tag{7}$$

- Dans ce qui suit, on cherchera à calculer ces dérivées partielles. en commençant par le cas de l = L - 1

Entrainement et algorithme du gradient

- si a $= a^{(L-1)}$ est la derniere couche (la sortie) alors le cout d'un entrainement est

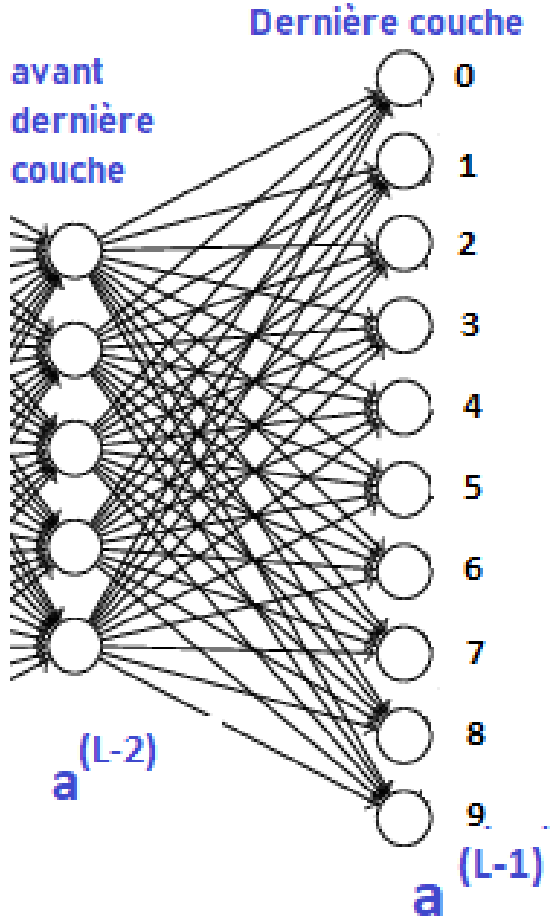
$$C_x = \frac{||a-y||^2}{2} = \frac{1}{2}\sum_k (a_k - y_k)^2$$

- Alors on a : $\frac{\partial C_x}{\partial a_k} = (a_k - y_k)$
- Sachant que $a_k = f(\sum_j w_{kj}^{(L-1)} a_j^{(L-2)} + b_k^{(L-1)})$

$$\frac{\partial C_x}{\partial b_k} = 1 \times f'(z_k) \times (a_k - y_k) \qquad (8)$$

$$\frac{\partial C_x}{\partial w_{kj}} = c_j \times f'(z_k) \times (a_k - y_k) \qquad (9)$$

Avec $z_k = \sum_j w_{kj}^{(L-1)} a_j^{(L-2)} + b_k^{(L-1)}$



Entrainement et algorithme du gradient

- Généralisation, on commence par introduire les notations suivantes
- je définis $(z^l)_{1\leq l\leq L-1}$ par $\boxed{z^{(l)} = w^{(l)} \cdot a^{(l-1)} + b^l}$
  (Ainsi on a $a^{(l)} = f(z^{(l)}) = f(w^{(l)} \cdot a^{(l-1)} + b^l)$)
- On considère $\delta^{(l)} = \nabla_{z^{(l)}} C$ identique à $\delta^{(l)} = Vect(\delta_j^{(l)})$ avec $\delta_j^{(l)} = \frac{\partial C}{\partial z_j^{(l)}}$
- **Produit d'Hadamard** si $u = Vect(u_j)$ et $v = Vect(v_j)$ deux vecteurs de même dimension, alors :

$$u \odot v = Vect(u_j v_j)$$

Entrainement et algorithme du gradient

- On trouve les relations suivantes :

$$\delta^{(l)} = f'(z^{(l)}) \odot ((w^{(l+1)})^T \cdot \delta^{(l+1)}) \tag{10}$$

$$\delta^{(L-1)} = f'(z^{(L-1)}) \odot (a - y) \tag{11}$$

$$\frac{\partial C}{\partial b_k^{(l)}} = \delta_j^{(l)} \tag{12}$$

$$\frac{\partial C}{\partial w_{kj}^{(l)}} = a_j^{(l-1)} \times \delta_k^{(l)} \tag{13}$$

- Voir annexe pour la preuve.

Entrainement et algorithme du gradient

- la mise à jour des poids et des contraintes (Relations (12)+(13)+(6)+(7))

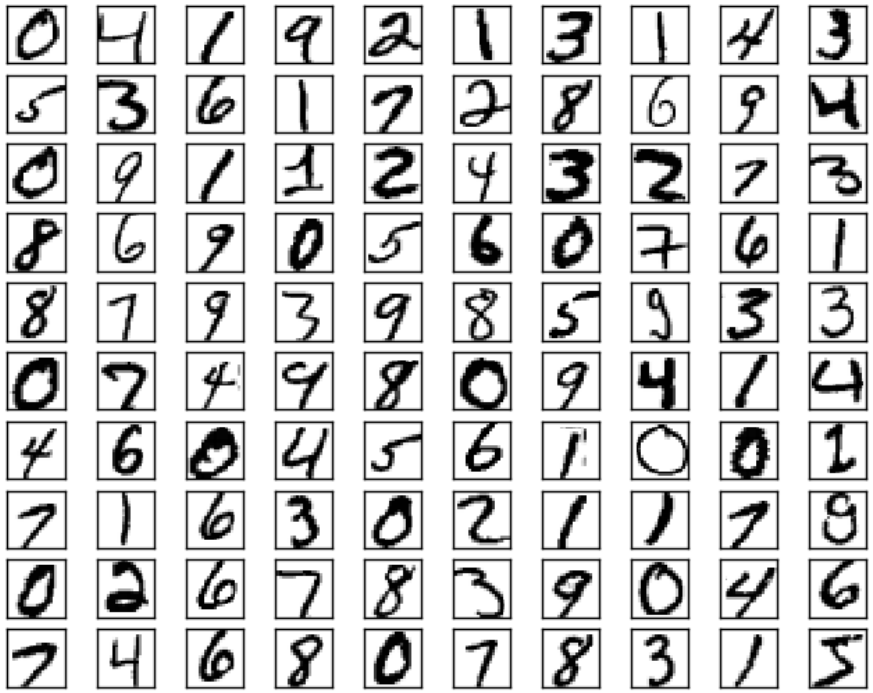
$$w_{kj}^{(l)} \leftarrow w_{kj}^{(l)} - \eta \times a_j^{(l-1)} \times \delta_k^{(l)} \quad (14)$$

$$b_k^{(l)} \leftarrow b_k^{(l)} - \eta \times \delta_k^{(l)} \quad (15)$$

- la mise à jour va nous permettre d'améliorer les paramétres du réseau (poids et contraintes), ainsi l'algorithme (le réseau) apprend à mieux reconnaitre les images des chiffres (apprentissage)

Entrainement et algorithme du gradient

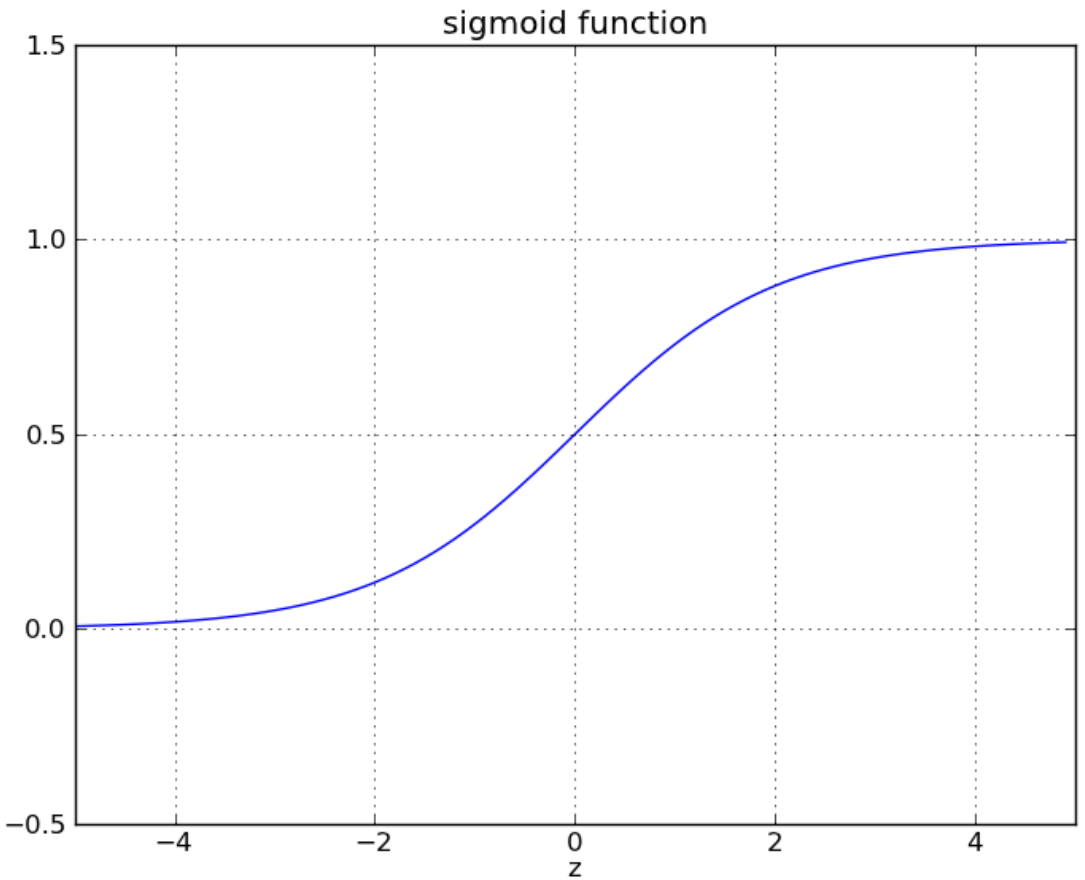
- Pour l'apprentissage, on a besoin de dizaines de milliers d'images de chiffres .
  - Heuresement, la base de données de MNIST regroupe 70 000 images de 28*28 pixels en mode gris, ainsi que leurs valeurs respectives. Ce sont des images de chiffres écrits à la main. On utilisera 50 000 images pour l'apprentissage et le reste pour les tests.



fonction d'activation

- **Sigmoid**

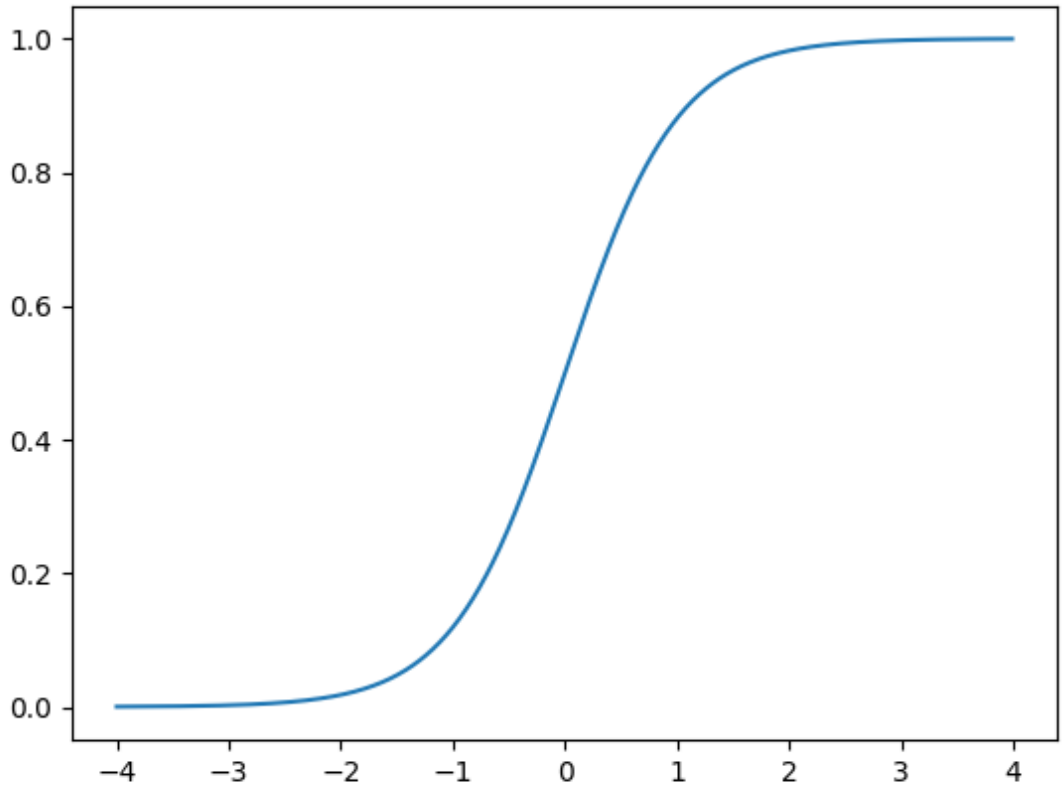
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



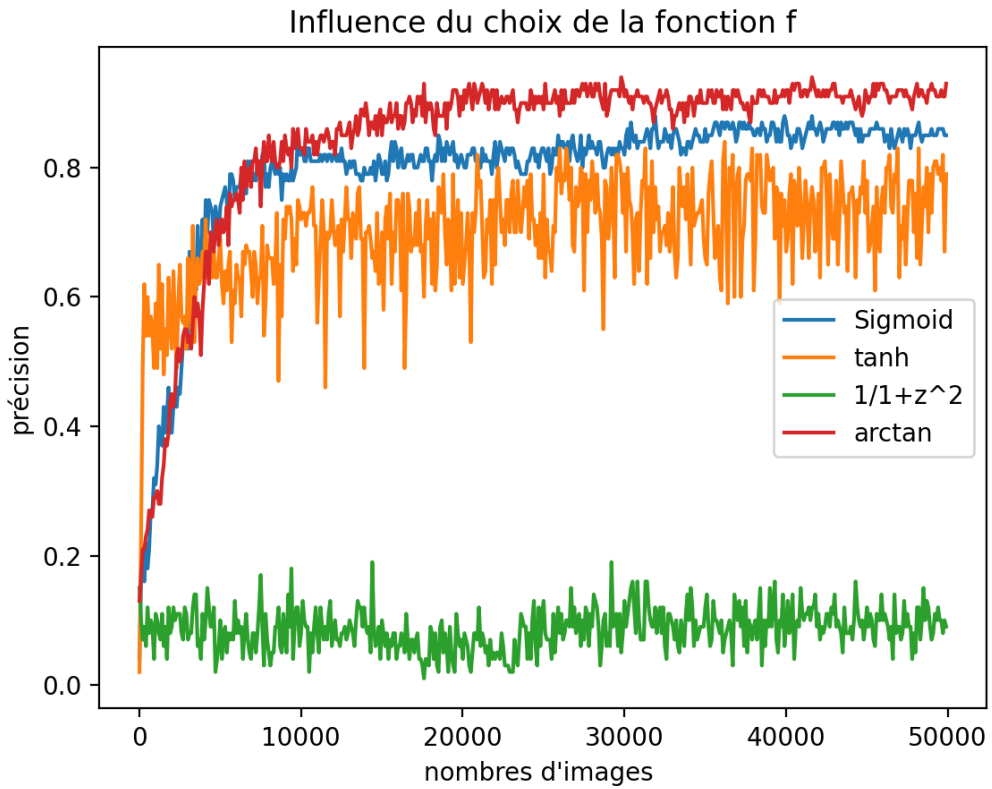
fonction d'activation

- **Tangente hyperbolique**

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



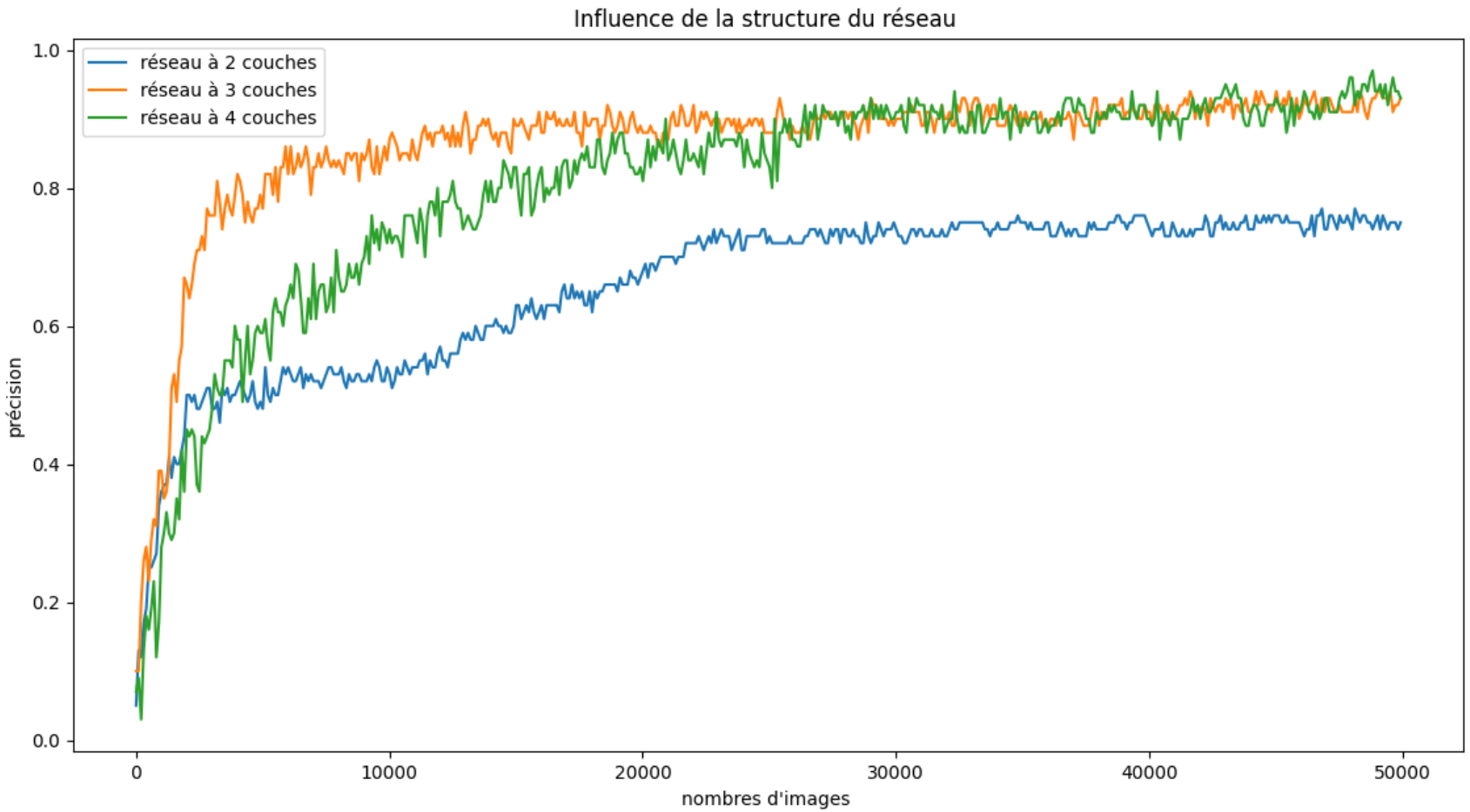
Influence du choix de la fonction f



- La precision $= \dfrac{\text{nombre d'images reconnues}}{\text{nombre total d'images testées}}$
- la courbe montre l'évolution de la précision de l'algorithme tout au long de l'entrainement (pour différentes fonctions d'activations)
- À la fin de l'entrainement (après 50.000 images), on obtient une précision de 94 pourcent !

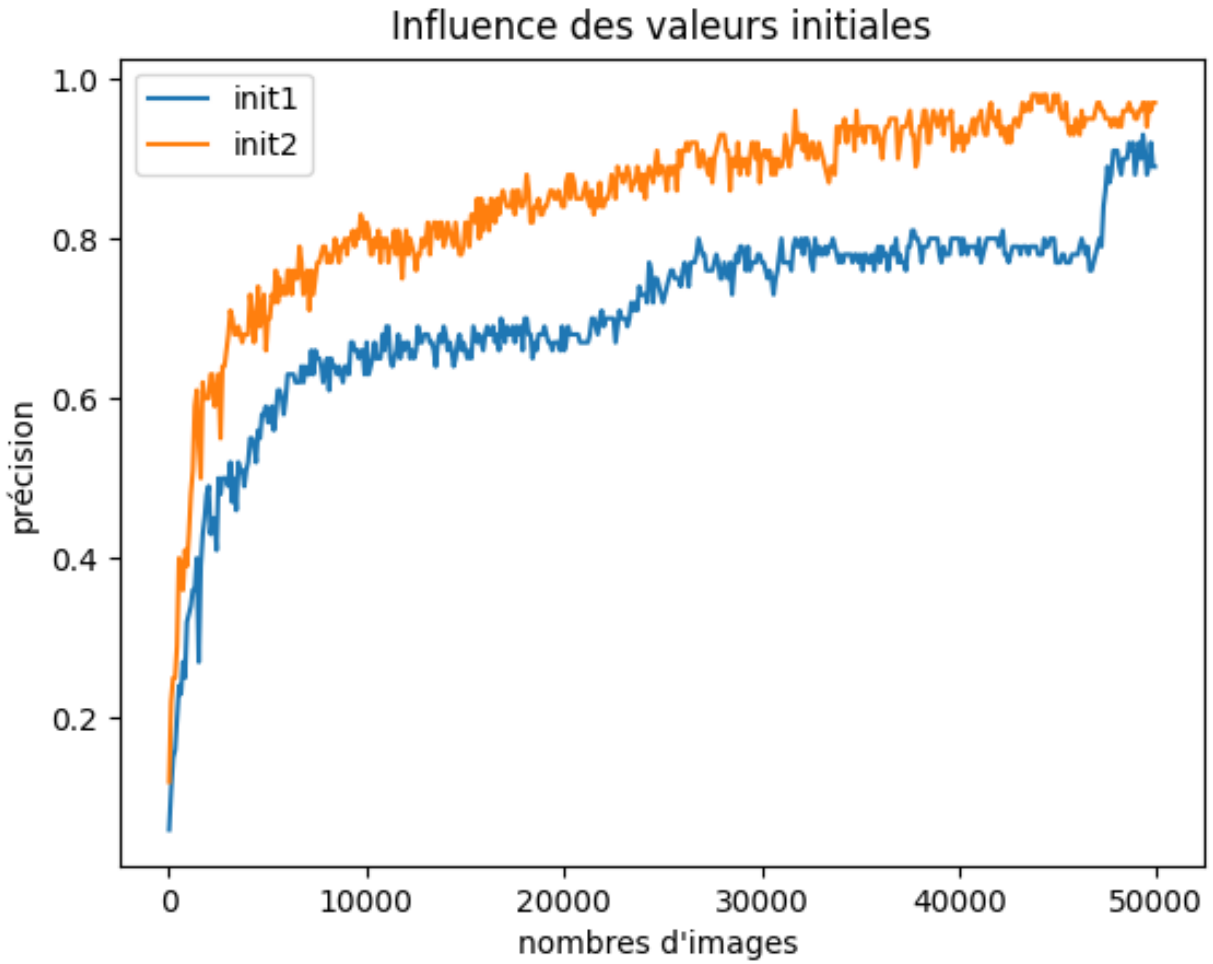
Influence de nombres de couches

```
structure1 = N.Network([784,10])
structure2 = N.Network([784,16,10])
structure3 = N.Network([784,16,16,10])
```

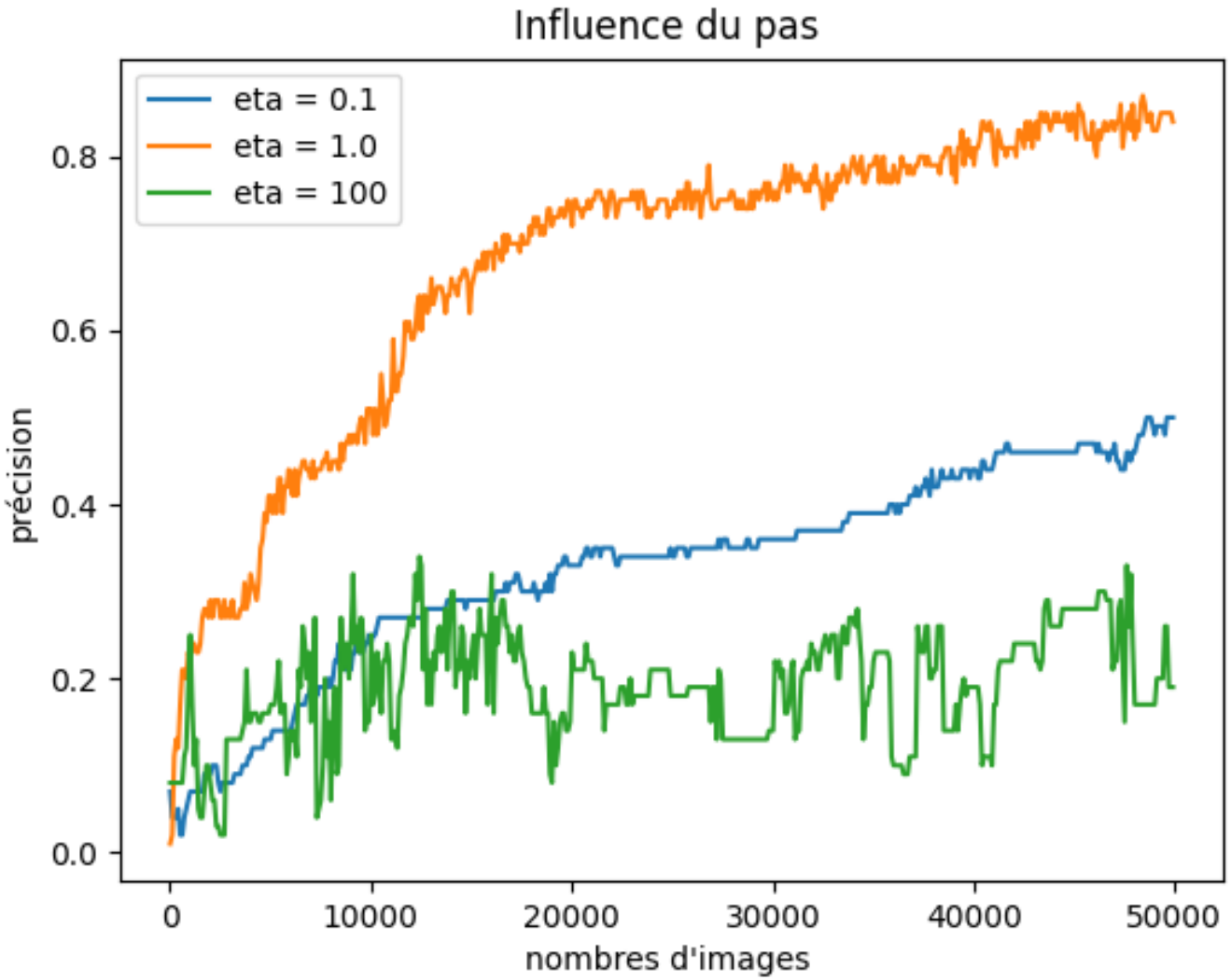


Influence de l'initialisation

```
reseau1 = N.Network([784,30,10])
reseau2 = N.Network([784,30,10])
```



Influence du pas (eta)



Limites du modèle

- les images des chiffres doivent être centrées.
- l'orientation du chiffre .

Solutions envisageables :

Modifier les images d'apprentissage en ajoutant les mêmes à orientation près.

Annexe

1. Code 1 : définir un réseau de neurones et algorithme de gradient.
2. Code 2 : Entrainement et influence des paramétres.
3. Code 3 : Algorithme reconnaissance d'image d'un chiffre
4. Démonstartion des relations 10 à 13 (page 17)

MERCI POUR VOTRE ATTENTION !

Code 1

```
"""CODE 1 : Reseau de neurones + Algorithme du gradient"""
class Network(object):

    def __init__(self, tailles): #tailles est une liste qui contient la dimension de chaque couche
        self.nb_tailles = len(tailles) #c'est le nombre de couche
        self.tailles = tailles
        self.contraintes = [np.random.randn(x,1) for x in tailles[1:] ]
        self.poids = [ np.random.randn(y,x)
                       for x,y in zip(tailles[:-1],tailles[1:])  ]
        #contraintes et poids sont initialement choisis al atoirement
    def calcul(self,a,f):
        for b,w in zip(self.contraintes, self.poids):
            a = f(np.dot(w,a)+b)
        return a
```

```
1
2     def variation(self,x,y,eta,f,df) :
3         delta_b = [np.zeros(y) for y in self.tailles[1:]]#va
      contenir la variation des contraintes
4         delta_w = [np.zeros(y) for y in self.tailles[1:]]#
    pareil pour les poids
5         a = x
6         zl=[] #liste qui contient les z_l
7         activations=[x] #la liste qi contient les a_l (les
    couches)
8         #on commence par remplir la liste z et activation
9         for b,w in zip(self.contraintes,self.poids):
10            z= np.dot(w,a)+b
11            zl.append(z)
12            a = f(z)
13            activations.append(a)
14        delta = df(zl[-1]) * (activations[-1]-y)
15        delta_b[-1] = (-eta)*delta
16        delta_w[-1]= np.dot(delta, activations[-2].transpose
    ()) *(-eta)
17        #delta reprensente initialement delta_{L-1}
18        for l in range(2,self.nb_tailles):
19            delta = delta = df(zl[-l]) * np.dot(self.poids[-
```

```
l+1].transpose(),delta)
20          delta_b[-l] = (-eta)*delta
21          delta_w[-l]= np.dot(delta, activations[-l-1].
transpose()) *(-eta)
22      return (delta_b,delta_w)
23
24  # mise    jour : change la valeur des poids et des
contraintes
25  #pour un minimiser la fonction C (le cout)
26  def mise_a_jour (self,mini,eta,f,df):
27      n =len(mini)
28      for x,y in mini :
29          delta_b,delta_w= self.variation(x,y,eta,f,df)
30          self.contraintes=[b+db/n for b,db in zip(self.
contraintes,delta_b)]
31          self.poids=[w+dw/n for w,dw in zip(self.poids,
delta_w)]
32
33  #Evaluer est une fonction qui prend en argument une
liste contenant les images
34  #   tester, et retourne le nombre de reponse correcte
35
36  def evaluer(self,test_data,f):
```

```
37          resultats = [(np.argmax(self.calcul(x,f)),y) for (x,
    y) in test_data]
38          #np.argmax renvoie l'indice de la case ayant la plus
     grande valeur
39          #x est l'image    tester.
40          return sum (int(y==z) for (z,y) in resultats) / len(
    test_data)
41
42      def entrainements(self,training_data,mini_taille ,eta,
    nb_entrainements,f,df):
43          n=len(training_data)
44          for j in range(nb_entrainements):
45              random.shuffle(training_data)
46              for k in range(0,n,mini_taille):
47                  mini = training_data[k:k+mini_taille]
48                  self.mise_a_jour(mini,eta,f,df)
49
50      def evolution_entrainement(self,training_data,
    mini_taille,eta,test,f,df):
51          y = []
52          x = []
53          n=len(training_data)
54          random.shuffle(training_data)
```

```
55          for k in range(0,n,mini_taille):
56              mini = training_data[k:k+mini_taille]
57              self.mise_a_jour(mini,eta,f,df)
58              x.append(k+mini_taille)
59              y.append(self.evaluer(test,f))
60          return (x,y)
```

Code 2

```
""" CODE 2 : Evolution de l'entrainement et influence des
    parametres"""

import networktipe as N
import data1 as D
import matplotlib.pyplot as plt
import numpy as np
training_data , test_data = D.load()
test = test_data[0:100]
#influence du choix de la fonction f
""" sigmoid et sa d riv e"""
def sigmoid(z):
    return 1.0/(1.0+np.exp(-z))

def ds(z):
    return sigmoid(z)*(1-sigmoid(z))
```

```
""" tangente hyperbolique """
def tanh(z):
    return (np.tanh(z)+1.0)/2

def dtanh(z) :
    return 0.5 * (1-tanh(z)**2)

""" 3 me  fonction"""
def f3(z):
    return 1.0/(1.0+z*z)
def df3(z):
    return 2*z/(1.0+z*z)**2

""" 4eme fonction"""
def f4(z):
    return np.arctan(z)/np.pi + 0.5

def df4(z) :
    y = 1 / (1.0 + z * z)
    return y/np.pi

##INFLUENCE DU CHOIX DE LA FONCTION
net1= N.Network([784,30,10])
```

```
net2= N.Network([784,30,10])
net3= N.Network([784,30,10])
net4= N.Network([784,30,10])
x,y1= net1.evolution_entrainement(training_data,10,3.0,test,
    sigmoid,ds)
_,y2= net2.evolution_entrainement(training_data,10, 3.0 ,
    test , tanh , dtanh)
_,y3= net3.evolution_entrainement(training_data,10,3.0,test,
    f3,df3)
_,y4= net4.evolution_entrainement(training_data,10,3.0,test,
    f4,df4)

plt.plot(x[0:5000:10], y1[0:5000:10], label='Sigmoid')  #
    Plot some data on the (implicit) axes.
plt.plot(x[0:5000:10], y2[0:5000:10], label='tanh')  # etc.
plt.plot(x[0:5000:10], y3[0:5000:10], label='1/1+z^2')
plt.plot(x[0:5000:10], y4[0:5000:10], label='arctan')
plt.xlabel("nombres d'images ")
plt.ylabel('pr cision')
plt.title("Influence du choix de la fonction f")
plt.legend()
plt.show()
```

```
42 ##influence des valeurs initiales
43
44 reseau1 = N.Network([784,30,10])
45 reseau2 = N.Network([784,30,10])
46
47 a,b = reseau1.evolution_entrainement(training_data,10,3.0,
       test,sigmoid,ds)
48 _,c = reseau2.evolution_entrainement(training_data,10,3.0,
       test,sigmoid,ds)
49
50 plt.plot(a[0:5000:10], b[0:5000:10], label='init1')
51 plt.plot(a[0:5000:10], c[0:5000:10], label='init2')
52 plt.xlabel("nombres d'images ")
53 plt.ylabel('pr cision')
54 plt.title("Influence des valeurs initiales")
55 plt.legend()
56 plt.show()
57
58 #influence du pas
59 pas1 = N.Network([784,30,10])
60 pas2 = N.Network([784,30,10])
61 pas3 = N.Network([784,30,10])
62
```

```
63 x1,z1 = pas1.evolution_entrainement(training_data,10,0.1,
       test,sigmoid,ds)
64 _,z2  = pas2.evolution_entrainement(training_data,10,1.0,
       test,sigmoid,ds)
65 _,z3  = pas3.evolution_entrainement(training_data,10,100,
       test,sigmoid,ds)
66
67 plt.plot(x1[0:5000:10], z1[0:5000:10], label='eta = 0.1')
68 plt.plot(x1[0:5000:10], z2[0:5000:10], label='eta = 1.0')
69 plt.plot(x1[0:5000:10], z3[0:5000:10], label='eta = 100')
70 plt.xlabel("nombres d'images ")
71 plt.ylabel('pr cision')
72 plt.title("Influence du pas ")
73 plt.legend()
74 plt.show()
75
76 #influence de la structure du r seau
77 structure1 = N.Network([784,10])
78 structure2 = N.Network([784,16,10])
79 structure3 = N.Network([784,16,16,10])
80
81 a1,b1 = structure1.evolution_entrainement(training_data
       ,10,3.0,test,sigmoid,ds)
```

```
_,b2  = structure2.evolution_entrainement(training_data
    ,10,3.0,test,sigmoid,ds)
_,b3  = structure3.evolution_entrainement(training_data
    ,10,3.0,test,sigmoid,ds)

plt.plot(a1[0:5000:10], b1[0:5000:10], label='r seau 2
    couches')
plt.plot(a1[0:5000:10], b2[0:5000:10], label='r seau 3
    couches')
plt.plot(a1[0:5000:10], b3[0:5000:10], label='r seau 4
    couches')
plt.xlabel("nombres d'images ")
plt.ylabel('pr cision')
plt.title("Influence de la structure du r seau ")
plt.legend()
plt.show()
```

Code 3

```
1  import networktipe as N
2  import cv2
3  import numpy as np
4
5
6  net = N.un_reseau()
7
8  #convertir en une image de taille 28*28 pixels
9  def to_28_28(image):
10     new= cv2.resize(image,(28,28))
11     return new
12
13 #convertir une image de taille 28*28 en un vecteur de
       dimension 784 = 28*28
14
15 def to_vector (image) :
16     x=cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
17     return np.reshape(1.0-x/255, (784, 1))
```

```
1
2
3  def reconnaitre(source) :
4      image = cv2.imread(source)
5      im = to_28_28(image)
6      x = to_vector(im)
7      return np.argmax(net.calcul(x, N.sigmoid))
8
9
10 ##images tester
11 billgates= "c:/Users/Lenovo/Desktop/pillowtest/Bill.jpg"
12 chiffre1_1 = "c:/Users/Lenovo/Desktop/TIPE/chiffre1(1).png"
13 chiffre1_2 = "c:/Users/Lenovo/Desktop/TIPE/chiffre1(2).png"
14 chiffre6  = "c:/Users/Lenovo/Desktop/TIPE/chiffre6.png"
15 chiffre5  = "c:/Users/Lenovo/Desktop/TIPE/chiffre5.png"
```

Démonstration

- Partant des notation introduites dans la page 17, on obtient facilement la relation 12 et 13 :

$$\boxed{\frac{\partial C}{\partial w_{kj}^{(l)}} = \frac{\partial z_k^{(l)}}{\partial w_{kj}^{(l)}} \cdot \frac{\partial C}{\partial z_k^{(l)}} = a_j^{(l-1)} \times \delta_k^{(l)}}$$

$$\boxed{\frac{\partial C}{\partial b_k^{(l)}} = \delta_k^{(l)}}$$

Démonstration

- On cherche à montrer la relation de recurrence verifier par $\delta^{(l)}$ (relation 10)
- $\delta_k^{(l)} = \frac{\partial C}{\partial z_k^{(l)}} = \frac{\partial a_k^{(l)}}{\partial z_k^{(l)}} \times \frac{\partial C}{\partial a_k^{(l)}}$

$$\delta_k^{(l)} = f'(z_k^{(l)}) \times \sum_j \frac{\partial z_j^{(l+1)}}{\partial a_k^{(l)}} \frac{\partial C}{\partial z_j^{(l+1)}}$$

$$\delta_k^{(l)} = f'(z_k^{(l)}) \times \sum_j w_{jk}^{(l+1)} \times \delta_k^{(l+1)}$$

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

- pour $l = L - 1$ (voir page 16)

$$\boxed{\delta^{(L-1)} = f'(z^{(L)}) \odot (a - y)}$$