PROJET PYTHON

RAGAVAN Ranushan MÉHEUST William

Fonctionnement d'un réseau neuronne

Avant d'attaquer le sujet, il serait intéressent de chercher à comprendre le fonctionnement global d'un réseau neuronne afin d'avoir une meilleur compréhension de l'algorithme Rn.

Un réseau neuronne (deep learning) est une structure informatique essentiellement mathématiques dont la mission est de prendre des décisions. Sa conception s'inspire fortement des reseaux neuronne qu'on retrouve en biologie. En effet, les réseaux neuronnes sont composés de multitudes de neuronnes liés les un aux autres et qui communiquent par stimulies. La conception de réseaux neuronnes artificelles suivent le même principe, cet à dire une multitude de neuronnes liés entre eux.

En deep learning, un neuronne est appelé un perceptrons.

1) Perceptron

Le perceptron est un modèle de classification binaire capable de séparer linéairement 2 classes de points.

Le modèle linéaire est de la forme $f(x_1,...,x_n)=a_1x_1+...+a_nx_n+b$ avec b le biais et les a_i des coeficients à déterminer. La classe de l'entré sera donnée par le signe de la fonction f.

Le modèle linéaire va créer une 'frontière' et plus un point sera éloigné de sa frontière et plus sa probabilité d'étre dans la classe 1 est élevé. Ainsi une couche de neuronne est enfaite un systeme linéaire et le resultat de cette couche est l'input de la prochaine couche.

2) Fonction d'activation

Cependant la valeur obtenue par le modèle linéaire n'est pas une probabilité. Afin de convertir cette sortie en probabilité, on passe la fonction dans une fonction dite d'activation qui tranforme les résultat des couches de neuronne en probabilité.

A titre d'exemple on peut citer la fonction logistique qui transforme une sortie z en probabilité à l'aide de la fonction logistique $a(z)=1/(1+e^{-z})$

On peut également cité softmax qui sera utilisé plus tard. Après avoir convertie les données en probabilités, le réseau neuronne va chercher à évaluer son erreur.

3) Loss function

Après avoir convertie les données en probabilités, le réseau de neuronnes va chercher à évaluer son erreur. Elle va donc utiliser des normes afin de calculer la distance entre l'étiquette prédite et la vraie étiquette. On peut citer le MSE (Mean squared error) qui correspond à la norme 2 ou même le MAE (Mean absolute error) qui correspond à la norme 1.

4) Forward propagation

Après avoir évaluer l'erreur, on minimise cette erreur à l'aide d'algorithme de descente de gradient comme la descente de gradient stochastique.

Compréhension du script Rn

- **shape**: est une liste contenant le nombre de perceptrons de chaques couchess de neuronnes et len(shape)= le nombre de couches.
- **--init--**:
  - Génére une liste self.b d'array à valeur nulle, ayant le même nombre d'éléments que la liste shape. De plus chaques array possèdent un nombre d'éléments égale à une valeur de liste shape. Cette liste est le biais des perceptrons des couches.
  - Génére une liste self.a contenant des array de dimension (shape[i+1],shape[i]) qui generent cette fois-ci les coeficients des perceptrons.
- **--str--**: Génére deux listes contenant des tuples avec les moyennes et les écarts types de self.a et self.b.
- **copy**: fait des copies de a et b
- **calculesortie**: $X[-1]$ contient l'output de la dernière couche de neuronnes, $X[0]$ contient les systeme $Ax+b$ des différentes couches.
- **Retro**: réalise une descente de gradient sur a et b dans le but de les optimiser.

Modèle 1

Le but de ce modèle est de prédire X un echantillon gaussien bruité (par du bruit gaussien) en disposant des valeurs initiales $theta$ qui suivent aussi une loi gaussienne.

-Dans un premier temps pour généré theta on utilise rng.normal et pour l'input $X[0][:]$ on utilise rng.normal+theta.

-Dans un deuximème temps on génère X qui sera une liste d'array, qui correspondra aux différentes couches de neuronnes. Cet à dire composé de l'input initial, des couches cachés et de l'output.

-Dans un troisème temps on calcule les predictions grâce à la fonction CalculeSortie.

-Dans un quatrième temps on évalue la distance entre $X[-1]$ (la prédiction) et theta(le resultat attendu à l'aide d'une Mse sur $X[-1]-\theta$.

-Dans un cinquième temps on va optimiser les coefficients a et b pour cela on effectue une descente de gradient avec le gradient $X[-1]-\theta$ et le $pas=1e-4$.

-Enfin, on repète ces operations 5000 fois, puis on evalue le réseau à l'aide d'une métric et on recommence ces opérations 50 fois grâce à une double boucle.

Modèle 2

Nous pouvons maintenant modifier le code pour l'adatpé à la prediction d'un echantillon suivant une loi uniform bruité (par du bruit gaussien).

-Dans un premier temps on définit theta avec np.uniform(-np.pi,np.pi,m) ce qui impactera également la loi de X.

-Dans un deuxième temps on calcule X_bar en utilisant np.mean sur la sortie de la fonction CalculSortie.

-Dans un troisème temps on implémente $g(\theta)$. Pour cela on génère une liste de deux éléments $cos(\theta)$ et $sin(\theta)$, puis on convertit la liste de array. On utilise la même méthode pour f(X_bar).

-Dans un quatrième temps on remplace le s du modèle 1 par la loss function du modèle 2 qu'on implémente en utilisant la fonction norm() de numpy.linalg.

-Dans un cinquième temps on change la fonction à minimiser pour cela on dispose du gradiant de la fonction qu'on cherche à minimiser cet à dire F-G.

-Enfin, on remplace le risque optimal par la formule de l'énoncé ce qui donne le code suivant.

In [8]:
```
import numpy as np
rng = np.random.default_rng()  # création du générateur de nombres aléatoires

class Rn:
    def __init__(self,shape, sigma):
        self.shape = shape

        self.b = [ np.zeros(n) for n in shape[1:] ]
        self.a = [ (rng.random((shape[ell+1],shape[ell])) - .5) * np.sqrt( 24/shape[ell] ) for ell in range(len(shape)-1) ]
        self.a[0] /= sigma
        self.a[-1] /= np.sqrt(2)

    def __str__(self):
        msb = [ (b.mean(), b.std()) for b in self.b]
        msa = [(a.mean(), a.std()) for a in self.a]
        return str(msb) + "\n" + str(msa)

    def Copy(self):
        rn = Rn(self.shape,1)
        rn.b = [ x.copy() for x in self.b ]
        rn.a =[x.copy() for x in self.a ]
        return rn

    def CalculSortie(self, X):
        for ell in range(len(self.shape)-2):
            X[ell+1][:] = np.maximum( 0,self.b[ell][:, np.newaxis] + self.a[ell] @ X[ell] )
        X[-1] = self.b[-1][:, np.newaxis] + self.a[-1] @ X[-2]

    def Retro(self, X, grad):
        for ell in range(len(self.shape)-2,-1,-1):
            aux = np.copy(grad)
            if ell < (len(self.shape)-2):
                grad *= ( X[ell+1]>0 )
            self.b[ell] -= grad.mean( axis=1 )
            self.a[ell] -= (grad[:,np.newaxis,:] * X[ell][np.newaxis,:,:]).mean(axis =2)
            grad = (self.a[ell][:,:,np.newaxis]*grad[:,np.newaxis,:]).sum(axis=0)

if __name__ == "__main__":

    def initialisation():
        m = 1
        X = [ np.zeros(shape = (n, m)) for n in shape ]

        for rep in range(5):
            Y = [ np.zeros(shape = (n, 1)) for n in shape ]
            for i in range(10000):
                theta = rng.uniform(-np.pi,np.pi,size = m) #loi de theta
                X[0][:] = theta[np.newaxis,:] + rng.normal(size = (n0,m))
                rn.CalculSortie(X)
                for j in range(len(Y)):
                    Y[j] += np.mean(X[j]**2, axis = 1)[:,np.newaxis]
            res = [  np.mean(y)/10000 for y in Y]
            for j in range(len(shape)-1):
                rn.a[j] /= np.sqrt(res[j+1])

    n0 = 20
    shape = (n0,n0,n0,2)
    rn = Rn( shape , np.sqrt(2))
    initialisation()
```

```
def test2():
    global rn
    m = 1

    X = [ np.zeros(shape = (n, m)) for n in shape ]

    taille_epi = 5000
    nb_epi = 50
    pas = 1e-10

    for rep in range(nb_epi):
        s = 0.0
        for i in range(taille_epi):

            theta = rng.uniform(-np.pi,np.pi,size = m) #loi de theta

            X[0][:] = theta[np.newaxis,:] + rng.normal(size = (n0,m))

            rn.CalculSortie(X)

            X_bar = X[0].mean()

            F = np.exp(-1/(2*n0))*np.array([np.cos(X_bar),np.sin(X_bar)]).reshape(2,1) #dans le cas m=1

            G = np.array([np.cos(theta),np.sin(theta)])

            s += (np.linalg.norm((F - G))**2).mean()

            grad = pas*(F - G)

            rn.Retro(X, grad)

    print(f"Episode {rep}  pas ={pas}")
    print("\n")
    print('risque=', s / taille_epi, 'risque optimal =', 1-np.exp(-1/n0))
    print("\n")
    print(rn)
    print("\n")

test2()
```

```
Episode 49  pas =1e-10


risque= 0.049556700826643006 risque optimal = 0.048770575499285984


[(8.4578549785363e-09, 8.652233191299611e-08), (4.823131336874458e-09, 4.5807155677266376e-08), (5.963858312888062e-10, 9.515620789614583e
-09)]
[(0.0037377098988672576, 0.14771896369571913), (0.016017479250960635, 0.36321327418487837), (-0.03038975779853146, 0.37755829507141864)]
```

Modèle de prédiction Mnist

Cette fois ci on cherche à modifier le modèle pour classer des images dans 10 catégories différentes. Les images sont de dimension 28x28 et on dispose d'un ensemble d'entrainement, d'un ensemble d'image test, et également de leurs labels.

Tout d'abord on implémente une fonction qui calcule l'entropie qui nous sera utile plus tard.

Comme l'input est de dimension 28x28 et l'output contiendra 10 éléments, on a donc shape=[28*28,10].

Cette fois ci les theta ne sont pas simuler aléatoirement, ils sont donnés par les labels. Or les labels sont des int on va donc changer int en vecteur.

Pour cela on on genere un array à 10 éléments nulles, puis on change la valeur i par 1. Par exemple si (label = i) alors (theta[i] = 1).

(X[0][:]=l'input) sera égale à une nouvelle image à chaque nouvelle itération des boucles.

Donc on le définie comme train__images[i], puis on le reshape pour ajouter la dimension des colonnes, ce qui corrigera un problème de dimension survenu plus tard.

Ensuite on calcule l'output avec la fonction CalculSortie.

Puis on convertit cette sortie en proabilité avec la fonction softmax.

L'erreur est évaluer par l'entropy, on remplace donc s par l'entropy.

On ajoute un accumulateur W qui s'initialise à 0, à chaque nouvel épisode et qui qui augmente de 1 pour chaques predictions correctes.

Puis nous affichons W/taille_epi ce qui donne le nombre de bonnes predicions / nombres de predictions

Enfin on utilise notre réseau de neuronnes sur les image_test. Pour cela, on sort de la 1er boucle et refait une boucle qui fait plus ou moins la même choses mais cette fois ci sur les images_test et les label_test.

In [2]:
```
import numpy as np
from scipy import special

rng = np.random.default_rng()  # création du générateur de nombres aléatoires



#-------------------- inport de l'image-------------------------
dt = np.dtype('uint32')
dt = dt.newbyteorder('>')  # big-endian, à commenter si besoin

f = open('train-images-idx3-ubyte', mode = 'rb')
x = f.read(16)
y = np.frombuffer(x, dtype = dt)
truc, nb_im, nb_rows, nb_cols = y
x = f.read(nb_rows*nb_cols*nb_im)
f.close()
train_images = np.frombuffer(x, dtype = 'ubyte').reshape(nb_im, nb_rows, nb_cols)

f = open('train-labels-idx1-ubyte', mode = 'rb')
x = f.read(8)
y = np.frombuffer(x, dtype = dt)
truc, nb_exemples = y
x = f.read(nb_exemples)
f.close()
train_labels = np.frombuffer(x, dtype = 'ubyte')

f = open('t10k-images-idx3-ubyte', mode = 'rb')
x = f.read(16)
y = np.frombuffer(x, dtype = dt)
truc, nb_im, nb_rows, nb_cols = y
x = f.read(nb_rows*nb_cols*nb_im)
f.close()
test_images = np.frombuffer(x, dtype = 'ubyte').reshape(nb_im, nb_rows, nb_cols)

f = open('t10k-labels-idx1-ubyte', mode = 'rb')
x = f.read(8)
y = np.frombuffer(x, dtype = dt)
truc, nb_exemples = y
x = f.read(nb_exemples)
f.close()
test_labels = np.frombuffer(x, dtype = 'ubyte')



#----------------------Calcul de l'entropie--------------
def D(P,Q):
    r = 0
    x = np.sum(Q.T, axis=0)
```

```
    for i in range(len(P)):
        if P[i] != 0:
            r += P[i]*np.log((P[i]/x[i]))

    return r

#-------------------------------------------------------------
class Rn:
    def __init__(self,shape, sigma):
        self.shape = shape

        self.b = [ np.zeros(n) for n in shape[1:] ]
        self.a = [ (rng.random((shape[ell+1],shape[ell])) - .5) * np.sqrt( 24/shape[ell] ) for ell in range(len(shape)-1) ]
        self.a[0] /= sigma
        self.a[-1] /= np.sqrt(2)

    def __str__(self):
        msb = [ (b.mean(), b.std()) for b in self.b]
        msa = [(a.mean(), a.std()) for a in self.a]
        return str(msb) + "\n" + str(msa)

    def Copy(self):
        rn = Rn(self.shape,1)
        rn.b = [ x.copy() for x in self.b ]
        rn.a =[x.copy() for x in self.a ]
        return rn

    def CalculSortie(self, X):
        for ell in range(len(self.shape)-2):
            X[ell+1][:] = np.maximum( 0,self.b[ell][:, np.newaxis] + self.a[ell] @ X[ell] )
        X[-1] = self.b[-1][:, np.newaxis] + self.a[-1] @ X[-2]

    def Retro(self, X, grad):
        for ell in range(len(self.shape)-2,-1,-1):
            aux = np.copy(grad)
            if ell < (len(self.shape)-2):
                grad *= ( X[ell+1]>0 )
            self.b[ell] -= grad.mean( axis = 1 )
            self.a[ell] -= (grad[:,np.newaxis,:] * X[ell][np.newaxis,:,:]).mean(axis = 2)
            grad = (self.a[ell][:,:,np.newaxis]*grad[:,np.newaxis,:]).sum(axis = 0)


if __name__ == "__main__":

    def initialisation():
        m = 1
        X = [ np.zeros(shape = (n, m)) for n in shape ]

        for rep in range(5):
            Y = [ np.zeros(shape = (n, 1)) for n in shape ]
            for i in range(10000):
                theta = np.zeros(10)
                theta[train_labels[i]] = 1 #loi de theta
                X[0][:] = train_images[i].reshape(784,1)
                rn.CalculSortie(X)
                for j in range(len(Y)):
                    Y[j] += np.mean(X[j]**2, axis = 1)[:,np.newaxis]
            res = [  np.mean(y)/10000 for y in Y]

            for j in range(len(shape)-1):
                rn.a[j] /= np.sqrt(res[j+1])

    n0 = 20
    shape = (28*28,10)
    rn = Rn( shape , np.sqrt(2))
    initialisation()



    def test2():
        global rn
        m = 1
        X = [ np.zeros(shape = (n, m)) for n in shape ]

        nb_epi = 10
        taille_epi = 60000
        pas = 1e-8


        for rep in range(nb_epi):
            s = 0.0
            W = 0
```

```
        for i in range(taille_epi):
            theta = np.zeros(10)
            theta[train_labels[i]] = 1 #loi de theta
            X[0][:] = train_images[i].reshape(784,1)
            rn.CalculSortie(X)
            proba = special.softmax(X[-1])
            entropy = D(theta,proba)
            s += entropy
            if np.argmax(proba) == np.argmax(theta):
                W += 1

            grad = pas*(proba - theta[:,np.newaxis])


            rn.Retro(X, grad)

    total = 0
    for j in range(0,test_images.shape[0],m):
        img_shp = test_images[j:j+m].shape
        X[0][:] = test_images[j:j+m].reshape(img_shp[0], img_shp[1]*img_shp[2]).transpose()
            #on envoie m image dans le reseau et on calcul la sortie
        rn.CalculSortie(X)

        g_alpha = special.softmax(X[-1].transpose())
            #on compte le nombre de fois ou le reseau donne le bon label en sortie
        count = (g_alpha.argmax(axis=1) == test_labels[j:j+m]).sum(dtype=int)
        total += count

    print(("Perfomance sur les images test :" +str(total/test_images.shape[0])))

test2()
```

Perfomance sur les images test :0.9169

Optimisation du réseau neuronne

Dans cette partie on va chercher à optimiser le reseau neuronne. Nous savons que le nombre de couches de neuronnes (shape) et nombre_epi , taille_epi, et le pas dans la descente de gradient sont des hyperparametre.

Essayons de les optimiser.

Meilleur valeurs pour le nombre d'epi

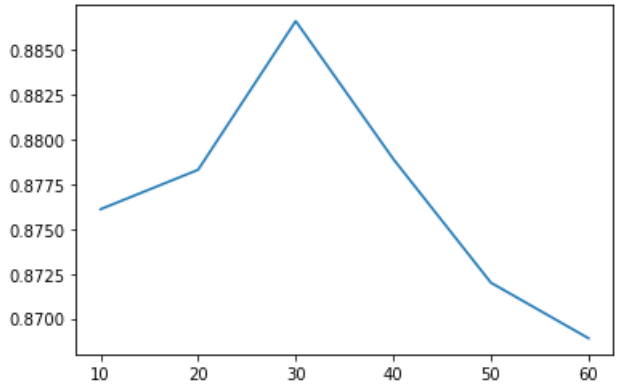
Cherchons le nombre d'episodes optimal.

In [4]:
```
import numpy as np
from scipy import special
from matplotlib import pyplot as plt
rng = np.random.default_rng()  # création du générateur de nombres aléatoires


#---------- import de l'image----------
dt = np.dtype('uint32')
dt = dt.newbyteorder('>')  # big-endian, à commenter si besoin

f = open('/Users/williammeheust/Desktop/M2_ISIFAR/Semestre_1/PYTHON/PROJET/train-images-idx3-ubyte', mode = 'rb')
x = f.read(16)
y = np.frombuffer(x, dtype = dt)
truc, nb_im, nb_rows, nb_cols = y
x = f.read(nb_rows*nb_cols*nb_im)
f.close()
train_images = np.frombuffer(x, dtype = 'ubyte').reshape(nb_im, nb_rows, nb_cols)

f = open('/Users/williammeheust/Desktop/M2_ISIFAR/Semestre_1/PYTHON/PROJET/train-labels-idx1-ubyte', mode = 'rb')
x = f.read(8)
y = np.frombuffer(x, dtype = dt)
truc, nb_exemples = y
x = f.read(nb_exemples)
f.close()
train_labels = np.frombuffer(x, dtype = 'ubyte')

f = open('/Users/williammeheust/Desktop/M2_ISIFAR/Semestre_1/PYTHON/PROJET/t10k-images-idx3-ubyte', mode = 'rb')
x = f.read(16)
y = np.frombuffer(x, dtype = dt)
truc, nb_im, nb_rows, nb_cols = y
x = f.read(nb_rows*nb_cols*nb_im)
f.close()
test_images = np.frombuffer(x, dtype = 'ubyte').reshape(nb_im, nb_rows, nb_cols)

f = open('/Users/williammeheust/Desktop/M2_ISIFAR/Semestre_1/PYTHON/PROJET/t10k-labels-idx1-ubyte', mode = 'rb')
x = f.read(8)
```

```
y = np.frombuffer(x, dtype = dt)
truc, nb_exemples = y
x = f.read(nb_exemples)
f.close()
test_labels = np.frombuffer(x, dtype = 'ubyte')

if __name__ == "__main__":
    import matplotlib.pyplot as plt
    #fig, ax = plt.subplots(3,3)
    #for i in range(9):
        #ax[i%3,i//3].imshow(test_images[-i,:,:], cmap = plt.cm.gray)
        #ax[i%3,i//3].set_title(str(test_labels[-i]))
        #ax[i%3,i//3].set_axis_off()

    #plt.show()
#----------------------Calcul de l'entropie--------------
def D(P,Q):
    r = 0
    #Q.reshape(10)
    x = np.sum(Q.T, axis=0)


    for i in range(len(P)):
        if P[i] != 0:
            #r += P[i]*np.log((P[i]/Q[i]))
            r += P[i]*np.log((P[i]/x[i]))

    return r

#-----------------------------------------------------------------
class Rn:
    def __init__(self,shape, sigma):
        self.shape = shape

        self.b = [ np.zeros(n) for n in shape[1:] ]
        self.a = [ (rng.random((shape[ell+1],shape[ell])) - .5) * np.sqrt( 24/shape[ell] ) for ell in range(len(shape)-1) ]
        self.a[0] /= sigma
        self.a[-1] /= np.sqrt(2)

    def __str__(self):
        msb = [ (b.mean(), b.std()) for b in self.b]
        msa = [(a.mean(), a.std()) for a in self.a]
        return str(msb) + "\n" + str(msa)

    def Copy(self):
        rn = Rn(self.shape,1)
        rn.b = [ x.copy() for x in self.b ]
        rn.a =[x.copy() for x in self.a ]
        return rn

    def CalculSortie(self, X):
        for ell in range(len(self.shape)-2):
            X[ell+1][:] = np.maximum( 0,self.b[ell][:, np.newaxis] + self.a[ell] @ X[ell] )
        X[-1] = self.b[-1][:, np.newaxis] + self.a[-1] @ X[-2]

    def Retro(self, X, grad):
        for ell in range(len(self.shape)-2,-1,-1):
            aux = np.copy(grad)
            if ell < (len(self.shape)-2):
                grad *= ( X[ell+1]>0 )
            self.b[ell] -= grad.mean( axis = 1 )
            self.a[ell] -= (grad[:,np.newaxis,:] * X[ell][np.newaxis,:,:]).mean(axis = 2)
            grad = (self.a[ell][:,:,np.newaxis]*grad[:,np.newaxis,:]).sum(axis = 0)


if __name__ == "__main__":

    def initialisation():
        m = 1
        X = [ np.zeros(shape = (n, m)) for n in shape ]
        #print(X[-1])
        for rep in range(5):
            Y = [ np.zeros(shape = (n, 1)) for n in shape ]
            for i in range(10000):
                theta = np.zeros(10)
                theta[train_labels[i]] = 1 #loi de theta
                X[0][:] = train_images[i].reshape(784,1)
                rn.CalculSortie(X)
                for j in range(len(Y)):
                    Y[j] += np.mean(X[j]**2, axis = 1)[:,np.newaxis]
            res = [  np.mean(y)/10000 for y in Y]
            #print(res)
```

```
        for j in range(len(shape)-1):
            rn.a[j] /= np.sqrt(res[j+1])

n0 = 20
shape = (28*28,10)
rn = Rn( shape , np.sqrt(2))
initialisation()
#print(rn.b)


def test2():
    global rn
    m = 1
    X = [ np.zeros(shape = (n, m)) for n in shape ]

    nb_epis = [60,50,40,30,20,10]
    passs = [1e-6,1e-5,1e-4,1e-3]
    l=[]
    for k in range(len(nb_epis)):
    #for a in range(len(passs)):
        nb_epi=nb_epis[k]
        taille_epi=60000
        pas=1e-4

        for rep in range(nb_epi):
            s = 0.0
            W = 0
            j=0
            for i in range(taille_epi):
                theta = np.zeros(10)
                theta[train_labels[i]] = 1 #loi de theta
                X[0][:] = train_images[i].reshape(784,1)
                rn.CalculSortie(X)
                proba = special.softmax(X[-1])
                entropy = D(theta,proba)
                s += entropy
                if np.argmax(proba) == np.argmax(theta):
                    W += 1


                #else :
                    #print(proba.T)


                #if i == 5:
                    #print(proba.T- theta[np.newaxis,:])
                    #print(proba.T)

                grad = pas*(proba - theta[:,np.newaxis])


                rn.Retro(X, grad)

            j+=1
        total = 0
        for j in range(0,test_images.shape[0],m):
            img_shp = test_images[j:j+m].shape
            X[0][:] = test_images[j:j+m].reshape(img_shp[0], img_shp[1]*img_shp[2]).transpose()

            rn.CalculSortie(X)

            g_alpha = special.softmax(X[-1].transpose())
                #on compte le nombre de fois ou le reseau donne le bon label en sortie
            count = (g_alpha.argmax(axis=1) == test_labels[j:j+m]).sum(dtype=int)
            total += count


        l.append(total/test_images.shape[0])
    print(l)
    plt.figure()
    plt.plot(nb_epis,l)
test2()
```

```
/var/folders/ny/wpshbz_s4y9f923ylrzb3j300000gn/T/ipykernel_77176/2117328925.py:62: RuntimeWarning: divide by zero encountered in double_scalars
  r += P[i]*np.log((P[i]/x[i]))
/var/folders/ny/wpshbz_s4y9f923ylrzb3j300000gn/T/ipykernel_77176/2117328925.py:62: RuntimeWarning: overflow encountered in double_scalars
  r += P[i]*np.log((P[i]/x[i]))
[0.8689, 0.872, 0.8789, 0.8866, 0.8783, 0.8761]
```



D'après le plot le meilleur nombre d'episode est 10.

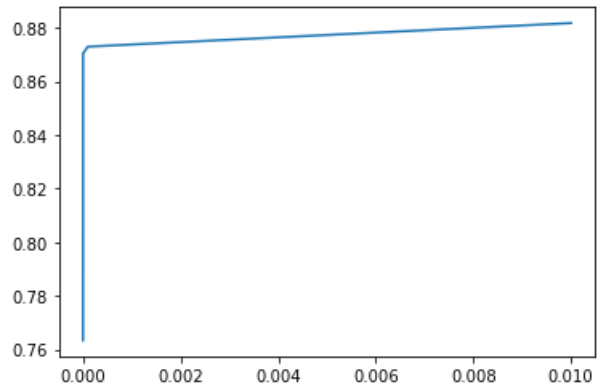
Recherche du meilleur pas

In [2]:
```
import numpy as np
from scipy import special
from matplotlib import pyplot as plt
rng = np.random.default_rng()  # création du générateur de nombres aléatoires


#---------- import de l'image----------
dt = np.dtype('uint32')
dt = dt.newbyteorder('>')  # big-endian, à commenter si besoin

f = open('/Users/williammeheust/Desktop/M2_ISIFAR/Semestre_1/PYTHON/PROJET/train-images-idx3-ubyte', mode = 'rb')
x = f.read(16)
y = np.frombuffer(x, dtype = dt)
truc, nb_im, nb_rows, nb_cols = y
x = f.read(nb_rows*nb_cols*nb_im)
f.close()
train_images = np.frombuffer(x, dtype = 'ubyte').reshape(nb_im, nb_rows, nb_cols)

f = open('/Users/williammeheust/Desktop/M2_ISIFAR/Semestre_1/PYTHON/PROJET/train-labels-idx1-ubyte', mode = 'rb')
x = f.read(8)
y = np.frombuffer(x, dtype = dt)
truc, nb_exemples = y
x = f.read(nb_exemples)
f.close()
train_labels = np.frombuffer(x, dtype = 'ubyte')

f = open('/Users/williammeheust/Desktop/M2_ISIFAR/Semestre_1/PYTHON/PROJET/t10k-images-idx3-ubyte', mode = 'rb')
x = f.read(16)
y = np.frombuffer(x, dtype = dt)
truc, nb_im, nb_rows, nb_cols = y
x = f.read(nb_rows*nb_cols*nb_im)
f.close()
test_images = np.frombuffer(x, dtype = 'ubyte').reshape(nb_im, nb_rows, nb_cols)

f = open('/Users/williammeheust/Desktop/M2_ISIFAR/Semestre_1/PYTHON/PROJET/t10k-labels-idx1-ubyte', mode = 'rb')
x = f.read(8)
y = np.frombuffer(x, dtype = dt)
truc, nb_exemples = y
x = f.read(nb_exemples)
f.close()
test_labels = np.frombuffer(x, dtype = 'ubyte')

if __name__ == "__main__":
    import matplotlib.pyplot as plt
    #fig, ax = plt.subplots(3,3)
    #for i in range(9):
        #ax[i%3,i//3].imshow(test_images[-i,:,:], cmap = plt.cm.gray)
        #ax[i%3,i//3].set_title(str(test_labels[-i]))
        #ax[i%3,i//3].set_axis_off()

    #plt.show()
#----------Calcul de l'entropie----------
def D(P,Q):
    r = 0
    #Q.reshape(10)
    x = np.sum(Q.T, axis=0)
```

```
    for i in range(len(P)):
        if P[i] != 0:
            #r += P[i]*np.log((P[i]/Q[i]))
            r += P[i]*np.log((P[i]/x[i]))

    return r

#-------------------------------------------------------------
class Rn:
    def __init__(self,shape, sigma):
        self.shape = shape

        self.b = [ np.zeros(n) for n in shape[1:] ]
        self.a = [ (rng.random((shape[ell+1],shape[ell])) - .5) * np.sqrt( 24/shape[ell] ) for ell in range(len(shape)-1) ]
        self.a[0] /= sigma
        self.a[-1] /= np.sqrt(2)

    def __str__(self):
        msb = [ (b.mean(), b.std()) for b in self.b]
        msa = [(a.mean(), a.std()) for a in self.a]
        return str(msb) + "\n" + str(msa)

    def Copy(self):
        rn = Rn(self.shape,1)
        rn.b = [ x.copy() for x in self.b ]
        rn.a =[x.copy() for x in self.a ]
        return rn

    def CalculSortie(self, X):
        for ell in range(len(self.shape)-2):
            X[ell+1][:] = np.maximum( 0,self.b[ell][:, np.newaxis] + self.a[ell] @ X[ell] )
        X[-1] = self.b[-1][:, np.newaxis] + self.a[-1] @ X[-2]

    def Retro(self, X, grad):
        for ell in range(len(self.shape)-2,-1,-1):
            aux = np.copy(grad)
            if ell < (len(self.shape)-2):
                grad *= ( X[ell+1]>0 )
            self.b[ell] -= grad.mean( axis = 1 )
            self.a[ell] -= (grad[:,np.newaxis,:] * X[ell][np.newaxis,:,:]).mean(axis = 2)
            grad = (self.a[ell][:,:,np.newaxis]*grad[:,np.newaxis,:]).sum(axis = 0)


if __name__ == "__main__":

    def initialisation():
        m = 1
        X = [ np.zeros(shape = (n, m)) for n in shape ]
        #print(X[-1])
        for rep in range(5):
            Y = [ np.zeros(shape = (n, 1)) for n in shape ]
            for i in range(10000):
                theta = np.zeros(10)
                theta[train_labels[i]] = 1 #loi de theta
                X[0][:] = train_images[i].reshape(784,1)
                rn.CalculSortie(X)
                for j in range(len(Y)):
                    Y[j] += np.mean(X[j]**2, axis = 1)[:,np.newaxis]
            res = [  np.mean(y)/10000 for y in Y]
            #print(res)
            for j in range(len(shape)-1):
                rn.a[j] /= np.sqrt(res[j+1])

    n0 = 20
    shape = (28*28,10)
    rn = Rn( shape , np.sqrt(2))
    initialisation()
    #print(rn.b)


    def test2():
        global rn
        m = 1
        X = [ np.zeros(shape = (n, m)) for n in shape ]

        passs = [1e-10,1e-6,1e-4,1e-2] #liste de pas a tester
        l=[] #liste qui va stocker les resultat
        for a in range(len(passs)):
            nb_epi=10
            taille_epi=60000
            pas=passs[a]
```

```
        for rep in range(nb_epi):
            s = 0.0
            W = 0
            j=0
            for i in range(taille_epi):
                theta = np.zeros(10)
                theta[train_labels[i]] = 1 #loi de theta
                X[0][:] = train_images[i].reshape(784,1)
                rn.CalculSortie(X)
                proba = special.softmax(X[-1])
                entropy = D(theta,proba)
                s += entropy
                if np.argmax(proba) == np.argmax(theta):
                    W += 1


                #else :
                    #print(proba.T)


                #if i == 5:
                    #print(proba.T- theta[np.newaxis,:])
                    #print(proba.T)

                grad = pas*(proba - theta[:,np.newaxis])


                rn.Retro(X, grad)

            j+=1
        total = 0
        for j in range(0,test_images.shape[0],m):
            img_shp = test_images[j:j+m].shape
            X[0][:] = test_images[j:j+m].reshape(img_shp[0], img_shp[1]*img_shp[2]).transpose()

            rn.CalculSortie(X)

            g_alpha = special.softmax(X[-1].transpose())
                #on compte le nombre de fois ou le reseau donne le bon label en sortie
            count = (g_alpha.argmax(axis=1) == test_labels[j:j+m]).sum(dtype=int)
            total += count


        l.append(total/test_images.shape[0])
    print(l)
    plt.figure()
    plt.plot(passs,l)
test2()
```

```
/var/folders/ny/wpshbz_s4y9f923ylrzb3j300000gn/T/ipykernel_77176/2244617883.py:62: RuntimeWarning: divide by zero encountered in double_scalars
  r += P[i]*np.log((P[i]/x[i]))
/var/folders/ny/wpshbz_s4y9f923ylrzb3j300000gn/T/ipykernel_77176/2244617883.py:62: RuntimeWarning: overflow encountered in double_scalars
  r += P[i]*np.log((P[i]/x[i]))
[0.7634, 0.8705, 0.873, 0.8818]
```



Le meilleur pas semble etre 1e-10.