Université Paris Descartes
UFR de Mathématiques et Informatique
45, rue des Saints-Pères 75270 Paris cedex 06

UNIVERSITÉ PARIS DESCARTES

MASTER 2

Apprentissage en grande dimension

TP3 Réseaux de neurones - classification - pytorch

TP noté - A remettre à la fin du cours (lundi 21 octobre à 11h45), sous la forme de :
- **un Jupyter Notebook Python (par mail à andres.almansa@parisdescartes.fr)**
- **Les calculs peuvent être rendus sur une feuille de papier, ou bien intégrés au notebook au format latex/markdown (si vous avez le temps)**

N'oubliez pas de mettre votre nom dans le nom du fichier.

Mise en route Ce TP doit être fait sous Python. Si vous voulez installer Python et ses principales librairies (numpy, scipy, matplotlib) sur votre machine personnelle, je vous conseille d'installer directement Anaconda (`http://anaconda.com/download/`). Anaconda vous permet notamment de créer un Notebook Jupyter et de le sauvegarder pour le partager.
Vous pouvez également utiliser le serveur Jupyter de l'UFR dédié à l'enseignement :
`https://jupyter.ens.math-info.univ-paris5.fr/`
Ce serveur est plus puissant que votre station de travail peut contenir des versions différentes de python et des bibliothèques logicielles associées.
Les commandes python de ce TP sont également disponibles via ce lien
`http://bit.ly/2019nntp3final`
Vous pouvez vous en servir pour éviter d'avoir à recopier du code.
On commence par importer les différentes libraries nécessaires au TP.

```
import numpy as np
import matplotlib.pyplot as plt
import math
```

Dans ce TP, on s'intéresse à un problème de classification supervisée dans $\mathbb{R}^2$. Etant donnée une base d'apprentissage de points du plan, classés en deux classes $\mathcal{C}_0$ et $\mathcal{C}_1$, on souhaite apprendre une fonction permettant de prédire la classe de n'importe quel point du plan (cette fonction segmente donc $\mathbb{R}^2$ en deux).

Données d'apprentissage. On crée dans ce qui suit un exemple simple de données du plan $\mathbb{R}^2$ divisées en deux classes bien distinctes mais pas linéairement séparables (voir le Figure 1).

```
N = 100 # number of points per class
p = 2   # dimensionality of data
K = 2   # number of classes
X = np.zeros((N*K,p)) # data matrix (each row = single example)
y = np.zeros(N*K) # class labels

ix = range(0,N)
r = 0.5 # radius
```

```
t =   np.random.rand(N)*math.pi-math.pi/2 # theta
X[ix] = np.c_[-0.3+2*r*np.cos(t), r*np.sin(t)] + np.random.randn(N,p)*0.02
y[ix] = 0
ix2 = range(N,2*N)
t =   np.random.rand(N)*math.pi+math.pi/2 # theta
X[ix2] = np.c_[+0.3+2*r*np.cos(t), -0.5+r*np.sin(t)]  + np.random.randn(N,2)*0.02
y[ix2] = 1
```

On visualise ensuite les données

```
fig = plt.figure()
plt.scatter(X[:, 0], X[:, 1], c=y, s=40, cmap=plt.cm.cool)
plt.show()
```

► **Question 1 :**

1. Expliquez comment les données sont créées.

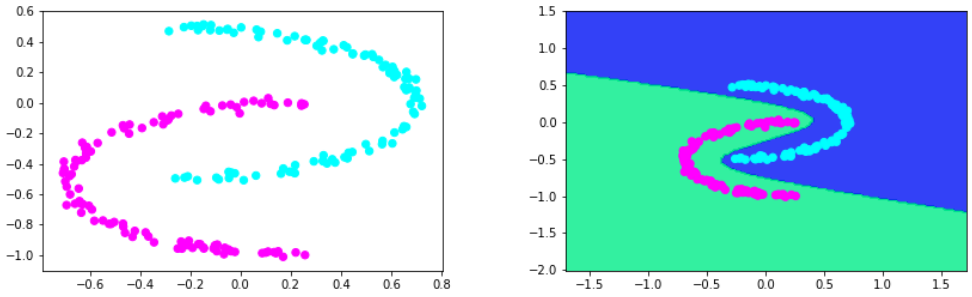

FIGURE 1 – Données d'apprentissage à gauche. Segmentation du plan apprise par le réseau de neurones à droite.

Un réseau à deux couches pour la classification à entraîner

Structure du réseau. Pour apprendre notre fonction de classification, on propose d'entraîner un réseau de neurones à deux couches et d neurones sur la couche intermédiaire. Ce réseau prend un point $x = (x_1, x_2)$ de $\mathbb{R}^2$ en entrée et donne un scalaire y entre 0 et 1 en sortie, représentant la probabilité que le point x appartienne à la classe $\mathcal{C}_0$.
La première couche du réseau est représentée par une matrice de poids $W^{[1]}$ de taille $d \times 2$ et un vecteur d'offset $b^{[1]}$ à d lignes. On utilise une fonction d'activation g (par exemple *tanh* ou ReLU). A la sortie de la première couche on observe donc

$$A^{[1]} = g(Z^{[1]}) \text{ avec } Z^{[1]} = W^{[1]}x + b^{[1]}$$

La matrice de poids $W^{[2]}$ permettant de passer de la couche cachée à la couche de sortie a 1 lignes et d colonnes, et le vecteur d'offset $b^{[2]}$ est un scalaire. On applique sur la couche de sortie une fonction d'activation

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

pour obtenir une valeur entre 0 et 1. La sortie $\widehat{y}$ est donc liée à $A^{[1]}$ par la relation

$$\widehat{y} = \sigma(Z^{[2]}) \text{ avec } Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}.$$

Pour simplifier les notations, on note dans ce qui suit $\theta = (W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]})$ l'ensemble des paramètres du réseau.

► **Question 2 :**

1. Ecrire la fonction complète $\widehat{y}(x, \theta)$ représentée par ce réseau.

On souhaite utiliser ce réseau pour faire de la régression à partir de M observations $\{(x^{(m)}, y^{(m)})\}_{m=1,\dots,M}$. Les $x^{(m)}$ sont des points de $\mathbb{R}^2$ et les $y^{(m)}$ valent 0 ou 1 selon que $x^{(m)}$ appartienne à la classe $\mathcal{C}_0$ ou $\mathcal{C}_1$. On cherche les paramètres $\theta = (W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]})$ qui minimisent la fonction de perte d'entropie croisée

$$\mathcal{L}(\theta) = -\sum_{m=1}^{M} \left(y^{(m)} \log \widehat{y}(x^{(m)}, \theta) + (1 - y^{(m)}) \log(1 - \widehat{y}(x^{(m)}, \theta)) \right).$$

Pour calculer le gradient de $\mathcal{L}$ par rapport aux différents paramètres de θ, on raisonne par backpropagation (on pourra s'inspirer fortement des réponses du TP1 pour faire cette partie).

► **Question 3 :**

1. Quelle est la dérivée de $\mathcal{L}$ par rapport à $\widehat{y}$?
2. Quelle est la dérivée de $\mathcal{L}$ par rapport à $Z^{[2]}$?
3. Quelle est la dérivée de $\mathcal{L}$ par rapport à $W^{[2]}$?
4. Quelle est la dérivée de $\mathcal{L}$ par rapport à $b^{[2]}$?
5. Quelle est la dérivée de $\mathcal{L}$ par rapport à $A^{[1]}$?
6. Quelle est la dérivée de $\mathcal{L}$ par rapport à $Z^{[1]}$?
7. Quelle est la dérivée de $\mathcal{L}$ par rapport à $W^{[1]}$?
8. Quelle est la dérivée de $\mathcal{L}$ par rapport à $b^{[1]}$?

Nous sommes maintenant en mesure d'implémenter et d'entraîner le réseau. On commence par mettre les données au bon format et par initialiser les paramètres du réseau aléatoirement.

```
x = X.T
y = np.reshape(y,(1,2*N))
d = 10                          # essayer avec 10 neurones pour commencer
W1 = 0.01 * np.random.randn(d,p)
B1 = np.zeros((d,1))
W2 = 0.01 * np.random.randn(1,d)
B2 = np.zeros((1,1))
```

Pour entraîner le réseau, on utilise dans un premier temps un pas de 0.01.

```
learning_rate = 0.01
```

On entraîne ensuite le réseau (cette partie est très semblable à ce que l'on a fait dans le TP précédent.

```
for t in range(2000):
    # Forward pass: compute predicted y
    z = np.dot(W1,x)+B1
    a = np.tanh(z)
    z2 = ...
    y_pred = 1/(1+np.exp(-z2))  # Sigmoid Activation funtion


    # Compute and print loss
    loss= (-np.log(y_pred)*y-np.log(1-y_pred)*(1-y)).sum()

    # Backprop to compute gradients of  W1, W2, B1 and B2 with respect to loss
    grad_y_pred = ...
```

```
    grad_z2 = ...
    grad_W2 = ...
    grad_B2 = ...
    grad_a = ...
    grad_z = ...
    grad_W1 = np.dot(grad_z,x.T)
    grad_B1 = grad_z.sum(axis=1,keepdims=True)

    # Update weights
    W1 -= learning_rate * grad_W1
    W2 -= learning_rate * grad_W2
    B1 -= learning_rate * grad_B1
    B2 -= learning_rate * grad_B2
```

On peut maintenant calculer la précision de la fonction de classification apprise sur la base d'apprentissage.

```
predicted_class = (y_pred > 0.5)
print("training accuracy: %.2f",np.mean(predicted_class == y))
```

Puis visualiser la segmentation du plan résultante.

```
h = 0.02
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))
Z = np.dot(np.tanh(np.dot(np.c_[xx.ravel(), yy.ravel()], W1.T) + B1.T), W2.T) + B2.T
Z = (1/(1+np.exp(-Z))>0.5)
Z = Z.reshape(xx.shape)
fig = plt.figure()
plt.contourf(xx, yy, Z, cmap=plt.cm.winter, alpha=0.8)
plt.scatter(X[:, 0], X[:, 1], c=np.reshape(y,[y.size]), s=40, cmap=plt.cm.cool)
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.show()
```

► **Question 4 :**

1. Entraîner plusieurs fois le réseau, à partir d'initialisations différentes. Qu'observez-vous ?
2. Ajouter une ligne dans le code permettant d'afficher la valeur de la fonction d'erreur toutes les 100 itérations. Commentez les résultats.
3. Tenter de remplacer la fonction d'activation tanh par la fonction $\mathrm{ReLU}(x) = \max(x, 0)$. Quelles lignes du code doivent changer (attention de changer également le code pour visualiser le résultat) ? Qu'observez-vous ?

Initiation à Pytorch

On a codé le réseau précédent à la main, en utilisant seulement le package *numpy* de Python. Pour faciliter cette tâche, il existe plusieurs packages spécialisés sous Python : *Tensorflow, Theano, Keras, Pytorch* (voir `https://developer.nvidia.com/deep-learning-frameworks` pour une liste de librairies spécialisées en deep learning). On se propose d'utiliser *Pytorch* pour recoder le réseau précédent.
On commence par importer la librairie (il faut pour cela qu'elle soit installée sur votre ordinateur).

```
import torch
from torch.autograd import Variable
```

On garde les données d'apprentissage déjà créées au début du TP et on les met au bon format pour les utiliser avec *Pytorch* (cette librarie n'utilise pas des tableaux de valeurs comme *numpy*, il faut donc transformer tous les tableaux *numpy* dans un format approprié).

```
x = X
y = np.reshape(y,(2*N,1))
xvar = Variable(torch.from_numpy(x)).type(torch.FloatTensor)
yvar = Variable(torch.from_numpy(y)).type(torch.FloatTensor)
```

On définit ensuite le réseau avec les lignes suivantes.

```
D_in, d, D_out =  p, 10, 1
model = torch.nn.Sequential(
    torch.nn.Linear(D_in, d),
    torch.nn.Tanh(),
    torch.nn.Linear(d, D_out),
    torch.nn.Sigmoid(),
)
```

On définit la fonction de perte

```
loss_fn = torch.nn.BCELoss()
```

Puis on entraîne le réseau.

```
learning_rate = 1
for t in range(1000):
     y_pred = model(xvar)
     loss = loss_fn(y_pred, yvar)
     if t % 100 == 0:
        print(t, loss.data[0])
    model.zero_grad()
    loss.backward()
    for param in model.parameters():
        param.data -= learning_rate * param.grad.data
```

► **Question 5 :** Expliquer ligne par ligne ce que font les morceaux de code précédents.

On peut à nouveau évaluer le score du réseau entraîné sur la base d'apprentissage et afficher le résultat.

```
scores = y_pred.data.numpy()
predicted_class = (scores>0.5)
print("training accuracy: %.2f",np.mean(predicted_class == y))

# plot the resulting classifier
h = 0.02
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),np.arange(y_min, y_max, h))
#Z = np.dot(np.maximum(0, np.dot(np.c_[xx.ravel(), yy.ravel()], W1.T) + B1.T), W2.T) + B2.T
```

```
X2  = np.zeros((xx.size,2))
X2[:,0] = np.reshape(xx,[xx.size])
X2[:,1] = np.reshape(yy,[yy.size])
X3 = Variable(torch.from_numpy(X2)).type(torch.FloatTensor)
scores = model(X3).data.numpy()
Z = (scores>0.5)
Z = Z.reshape(xx.shape)
fig = plt.figure()
plt.contourf(xx, yy, Z, cmap=plt.cm.winter, alpha=0.8)
plt.scatter(X[:, 0], X[:, 1], c=np.reshape(y,[y.size]),s=40, cmap=plt.cm.cool)
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.show()
```

Classification à 3 classes

On s'intéresse maintenant à un cas à 3 classes. On créé les données d'apprentissage $(x^{(m)}, y^{(m)})$ avec $y^{(m)} \in \{0, 1, 2\}$ de la manière suivante :

```
N = 100 # number of points per class
D = 2 # dimensionality
K = 3 # number of classes
X = np.zeros((N*K,D)) # data matrix (each row = single example)
y = np.zeros(N*K, dtype='uint8') # class labels
for j in range(0,K,1):
  ix = range(N*j,N*(j+1))
  r = np.linspace(0.0,1,N) # radius
  t = np.linspace(j*4,(j+1)*4,N) + np.random.randn(N)*0.2 # theta
  X[ix] = np.c_[r*np.sin(t), r*np.cos(t)]
  y[ix] = j
# lets visualize the data:
plt.scatter(X[:, 0], X[:, 1], c=y, s=40, cmap=plt.cm.Spectral)
plt.show()
```

► **Question 6 :** Créer à la main ou à l'aide de Pytorch un réseau de neurones à deux couches, d neurones sur la couche intermédiaire (prendre d entre 10 et 30 par exemple) et 3 neurones de sortie pour apprendre une fonction de classification à partir de ces données. Comme fonction d'activation sur la couche intermédiaire, on pourra utiliser tanh. On définit la fonction de perte de la manière suivante. Si $\widehat{y} = (\widehat{y}_1 \dots, \widehat{y}_3)$ est la sortie du réseau de neurones :

$$\mathcal{L}(\widehat{y}, y) = \sum_{m=1}^{M} -\log\left(\frac{e^{\widehat{y}_{y^{(m)}}}}{e^{\widehat{y}_1} + \cdots + e^{\widehat{y}_3}}\right).$$