INSAT	Module : Deep Learning
Département Mathématiques et Informatique	Sections : RT4
Année Universitaire : 2021 - 2022	Enseignante : Sana Hamdi sana.hamdi@fst.utm.tn

TP N°1 : Perceptron mono-couche

Objectif :

Ce TP a pour objectif d'introduire les bases de la théorie des réseaux de neurones et d'introduire le cas particulier du perceptron. Nous allons examiner l'algorithme Perceptron, qui est le réseau de neurones à une seule couche le plus élémentaire utilisé pour la classification binaire. Tout d'abord, nous examinerons la fonction d'activation et verrons comment l'algorithme Perceptron fait la classification, puis nous examinerons la règle de mise à jour du perceptron. Enfin, nous tracerons la frontière de décision pour nos données. Nous utiliserons les données avec seulement deux caractéristiques, et il y aura deux classes puisque Perceptron est un classifieur binaire. Nous allons implémenter tout le code en utilisant Python NumPy et visualiser/tracer en utilisant Matplotlib.

I. Notion de neurones

Un neurone est un objet mathématique qui fut à l'origine introduit, entre autres choses, pour modéliser le fonctionnement du cerveau humain dans le cadre d'études de la cognition. En interconnectant plusieurs neurones, nous formons alors un réseau de neurones. Vous trouvez l'analogie entre un neurone biologique et artificiel dans le cours.

Le principe général est de retourner une information en sortie à partir de plusieurs informations en entrée. L'information entrante peut être, par exemple, issue de l'information sortante d'autres neurones dans le cadre d'un réseau. Plus précisément, nous notons $x_1, \ldots, x_n \in R$ les informations entrantes et, pour chaque $i \in \{1, \ldots, n\}$, nous associons un poids $w_i \in R$ à x_i. Nous introduisons également un poids $w_0 \in R$, appelé coefficient de biais, associé à une information virtuelle $x_0 = 1$ dont le rôle sera précisé ultérieurement. L'information traitée par le neurone n'est pas l'ensemble des x_i mais leur somme pondérée.

$$s = \sum_{i=0}^{n} w_i x_i$$

Le rôle d'un réseau de neurones est de fournir une réponse *y* entre 0 et 1 à partir de *s*. Nous utilisons pour cela une fonction g : R → [0, 1], appelée fonction d'activation. Lorsque la réponse est proche de 1, nous dirons que le neurone est actif et pour une réponse proche de 0, le neurone sera dit inactif. Nous avons donc la sortie du neurone qui est définie par :

$$y = g(s) = g(\sum_{i=0}^{n} w_i x_i)$$

La fonction de Heaviside :

$$\forall \boldsymbol{x} \in \boldsymbol{R}, \boldsymbol{g}(\boldsymbol{x}) = \begin{cases} \mathbf{1}\ \boldsymbol{si}\ \boldsymbol{x} > \mathbf{0} \\ \mathbf{0}\ \boldsymbol{sinon} \end{cases}$$

II. Travail à faire :

1. Algorithme :

Pour l'algorithme de base du perceptron vu dans le cours :

1.1.Quelle est sa complexité en temps ?
1.2. Quelle est sa complexité en espace ?

2. Dataset :

Essayons de comprendre l'algorithme Perceptron en utilisant les données suivantes comme exemple motivant.

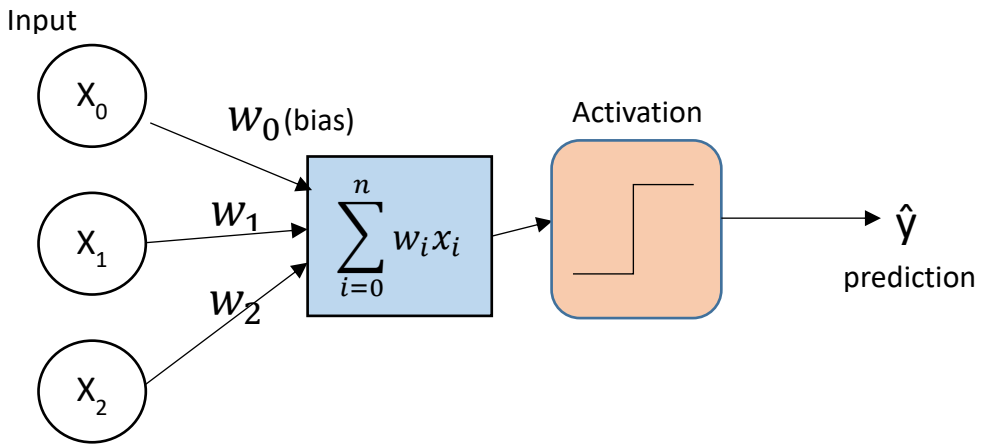
```
from sklearn import datasets

X, y = datasets.make_blobs (n_samples=150, n_features=2,
                            centers=2, cluster_std=1.05,
                            random_state=2)
#Plotting
fig = plt.figure(figsize=(10,8))
plt.plot(X[:, 0][y == 0], X[:, 1][y == 0], 'r^')
plt.plot(X[:, 0][y == 1], X[:, 1][y == 1], 'bs')
plt.xlabel("feature 1")
plt.ylabel("feature 2")
plt.title('Random Classification Data with 2 classes')
```

Il y a deux classes, rouge et bleue, et nous voulons les séparer en traçant une ligne droite entre elles. Ou, plus formellement, nous voulons apprendre un ensemble de paramètres w_i pour trouver un hyperplan optimal (ligne droite pour nos données) qui sépare les deux classes.

3. Implémentation :

Nous pouvons visuellement comprendre le Perceptron en regardant la figure ci-dessous. Pour chaque exemple d'apprentissage, nous prenons d'abord le produit scalaire des caractéristiques et paramètres d'entrée,w. Ensuite, nous appliquons la fonction d'activation pour faire la prédiction (y_hat).



1. Coder la fonction d'activation de Heaviside

```
def acti_func(z):
    ...
```

2. Implementer l'algorithme de perceptron

```
def perceptron(X, y, lr, epochs):

    # X --> Inputs.
    # y --> labels/target.
    # lr --> learning rate.
    # epochs --> Number of iterations.

    # m-> number of training examples
    # n-> number of features
    m, n = X.shape

    # Initializing parameters(theta) to zeros.
    # +1 in n+1 for the bias term.
    w = np.zeros((n+1,1))

    # Empty list to store how many examples were
    # misclassified at every iteration.
    n_miss_list = []

    # Training.
    for epoch in range(epochs):

        # variable to store #misclassified.
        n_miss = 0

        # looping for every example.
        for idx, x_i in enumerate(X):
```

```
            # Insering 1 for bias, X0 = 1.
            x_i = np.insert(x_i, 0, 1).reshape(-1,1)

            # Calculating prediction/hypothesis.
            y_hat = acti_func(np.dot(x_i.T, w))

            # Updating if the example is misclassified.
            if (np.squeeze(y_hat) - y[idx]) != 0:
               ....

               # Incrementing by 1.
               ...

      # Appending number of misclassified examples
      # at every iteration.
      n_miss_list.append(n_miss)

   return w, n_miss_list
```

3. Tracez la limite de décision trouvée par votre algorithme.

```
def plot_decision_boundary(X, w):

    # X --> Inputs
    # w --> parameters

    # The Line is y=mx+c
    # So, Equate mx+c = w0.X0 + w1.X1 + w2.X2
    # Solving we find m and c
    x1 = [min(X[:,0]), max(X[:,0])]
    m = ...
    c = ...
    x2 = m*x1 + c

    # Plotting
    ...
```

III. Compte rendu

1. Instructions :

- Vous devez implémenter ce qui est demandé dans cette section from scratch.
- Envoyez-moi votre notebook Jupiter **richement commenté** avant le 04/03/2022.

2. Dataset :

Considérons le data set $S = \{(x, y)\}_{i=1}^{250}$ composé de 250 points $x_i = (x_1, x_2)$ et leur classes y_i.

Les premières 125 x_i sont classées $y_i = -1$ et sont générées selon une distribution gaussienne $x_i \sim N(\mu_1, \sigma_1^2)$, où

$$\mu_1 = \begin{bmatrix} -1 \\ 0 \end{bmatrix}$$

Les dernières 125 x_i sont classées $y_i = 0$ et sont générées selon une distribution gaussienne $x_i \sim N(\mu_2, \sigma_2^2)$, où

$$\mu_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Après faire mélanger le dataset, on va le diviser en train and test sets, contenant 80 % et 20 % du dataset (utiliser des méthodes de shuffling et de splitting existantes).

3. Implémentation :

1. Implémenter l'algorithme de perceptron

2. **Expérience 1 :**

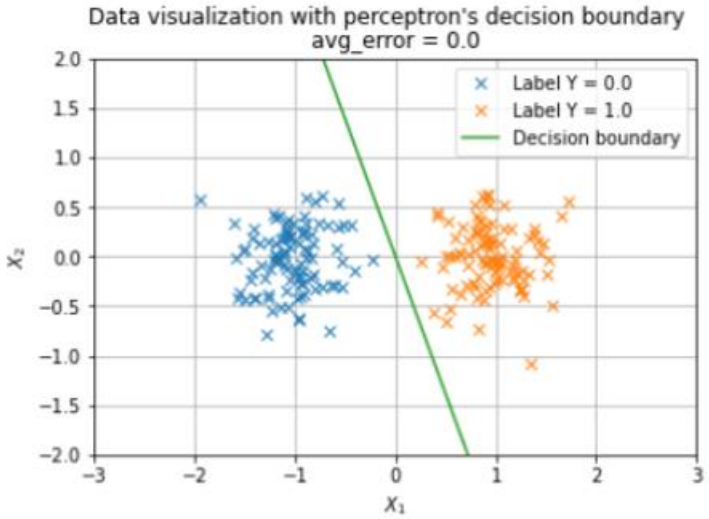
Générer un dataset pour $\sigma_1^2 = \sigma_2^2 = 0.25$

a. L'algorithme converge-t-il ? Pourquoi ?
b. Tracez la limite de décision trouvée par votre algorithme. Cette limite de décision est-elle unique ? La modification de l'initialisation modifie-t-elle le résultat de l'algorithme ?
c. Calculer la justesse (accuracy) de la classification sur l'ensemble de test. Tracez la limite de décision sur l'ensemble de test.

3. **Expérience 2 :**

Générer un dataset pour $\sigma_1^2 = \sigma_2^2 = 0.75$

d. L'algorithme converge-t-il ? Pourquoi ?
e. Tracez la limite de décision trouvée par votre algorithme. Cette limite de décision est-elle unique ? La modification de l'initialisation modifie-t-elle le résultat de l'algorithme ?
f. Calculer la justesse (accuracy) de la classification sur l'ensemble de test. Tracez la limite de décision sur l'ensemble de test.



4. Expérience 3 :

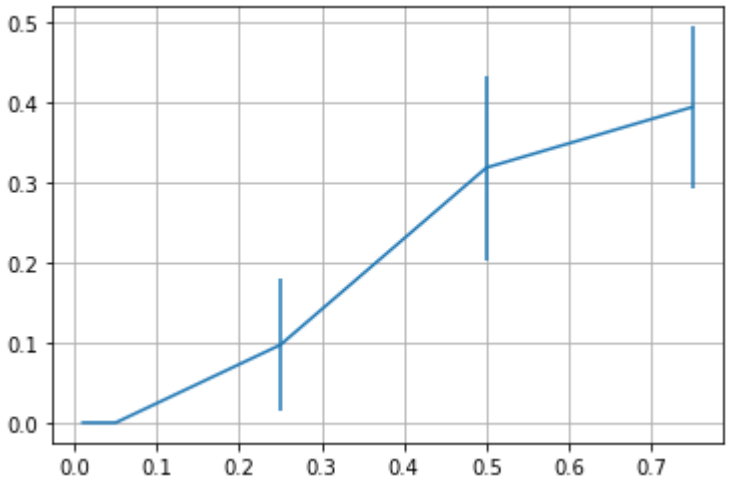
Nous définissons une expérience comme suit

• Générez les données et entraînez votre modèle.

• Calculez l'erreur sur l'ensemble de test.

Afin d'étudier l'impact de la variation de σ_2^1 et σ_2^2 sur les performances du système, nous stockons l'erreur sur plusieurs expériences (prendre nb expérience = 30). Puis on calcule la moyenne et la variance des erreurs stockées. Pour chaque σ_1^2 et $\sigma_2^2 \in [0.01, 0.1, 0.5, 0.7]$ calculez la moyenne et la variance puis tracez les résultats en utilisant **matplotlib.pyplot.errorbar**.

La figure suivante devrait être similaire au résultat attendu :



Commenter le résultat.