Université Mohammed 1er

École Nationale de l'Intelligence Artificielle et du Digitale - Berkane

TP 03 : Programmer un Perceptron avec Python et Numpy

Module : Deep Learning

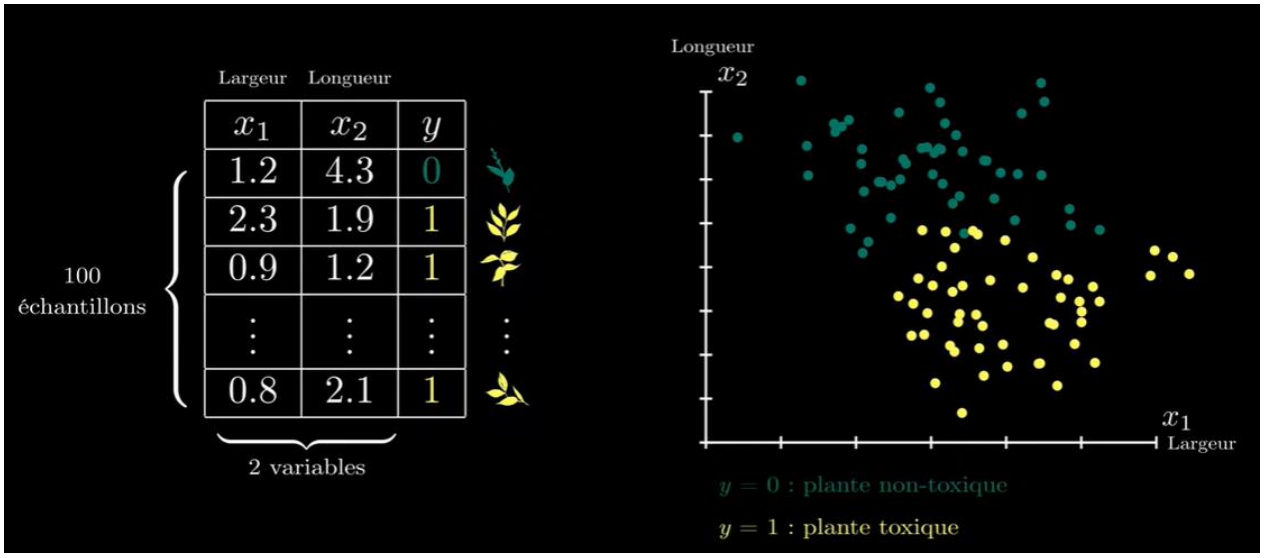
Année Universitaire : 2024/2025

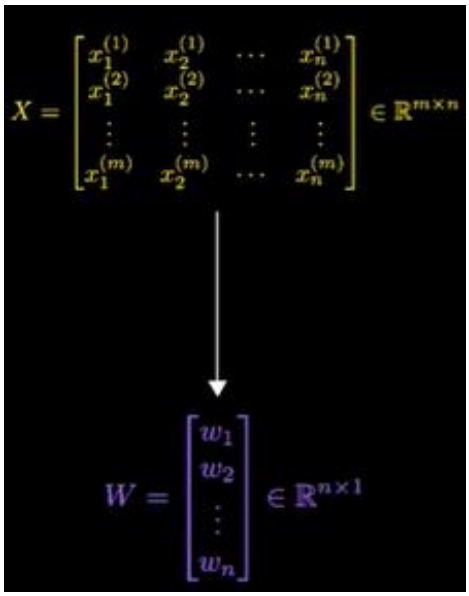
Filière : IA

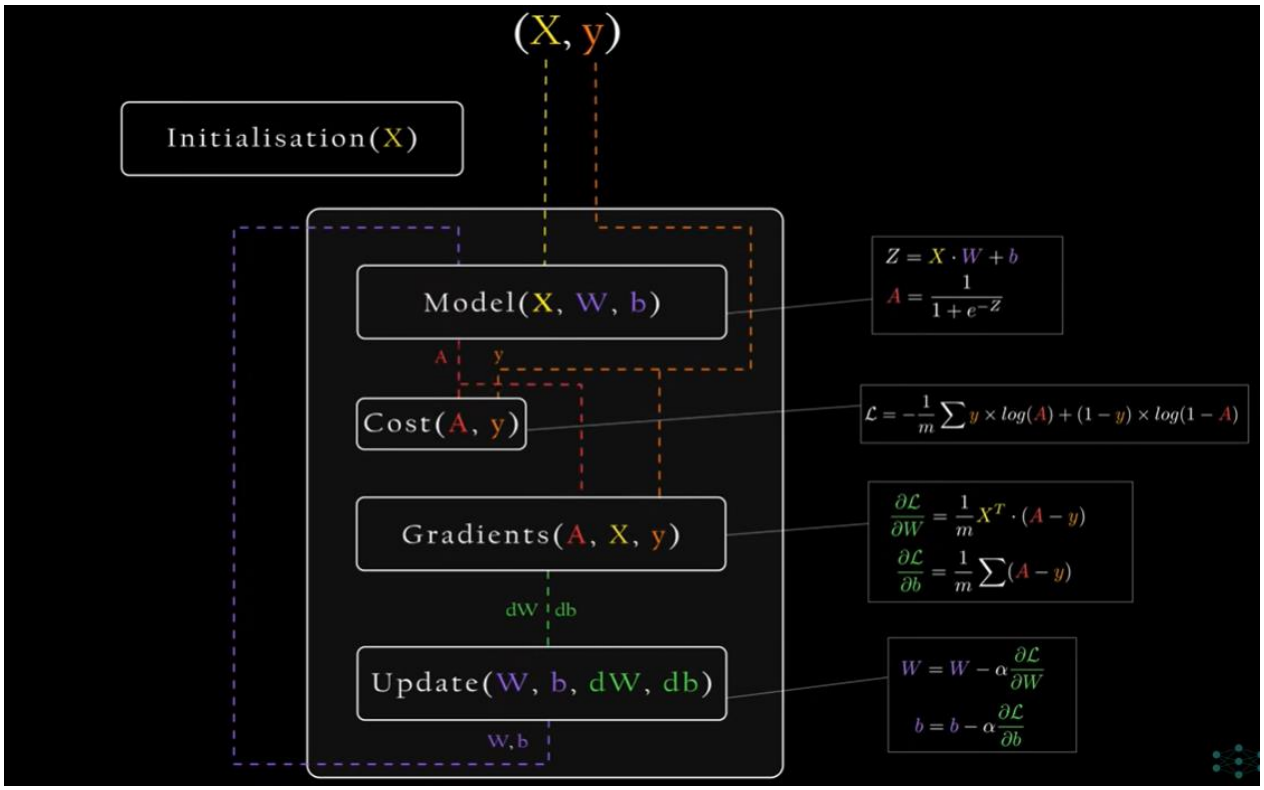
Réalisé par : Pr. ALAMI Nabil

Objectif

L'objectif de ce TP est de développer notre premier neurone artificiel en se servant uniquement de Python et Numpy sans utiliser un framework spécifique.







Alors pour développer notre programme de neurone artificiel nous allons partir d'un data x et y de 100 lignes et de deux colonnes si on veut on peut imaginer que ce dataset représentent des plantes avec la longueur et la largeur de leurs feuilles et notre but c'est d'entraîner un neurone artificiel à reconnaître les plantes toxiques des plantes non toxique grâce à ces données de référence. Alors pour faire cela nous allons devoir structurer notre code de la façon suivante:

Pour commencer nous aurons besoin d'une fonction d'initialisation qui comme son nom l'indique nous permettra d'initialiser les paramètres w et b de notre modèle. Dans cette fonction, nous ferons passer la matrice X car notre but c'est d'obtenir un vecteur W qui contient autant de paramètres que l'on trouve de variables dans la matrice X. Une fois cette étape d'initialisation effectué, nous écrirons ensuite un algorithme itératif dans lequel nous allons répéter en boucle les fonctions suivantes :

- Nous allons commencer par la fonction qui représente notre modèle de neurone artificiel. Celle dans laquelle on va retrouver la fonction linéaire Z = XW + b.
- Puis notre fonction d'activation.
- Ensuite nous aurons une fonction d'évaluation c'est à dire notre fonction coûts qui nous permettra d'évaluer la performance du modèle en comparant la sortie A aux données de référence y que l'on a gardé deux côtés.
- En parallèle nous pourrons également calculer les gradients de cette fonction coûts grâce aux formules que l'on a développé dans le cours.
- Et pour finir nous utiliserons ces gradients dans une dernière fonction qui permettra de mettre à jour les paramètres W et b de manière à réduire d'une petite quantité les erreurs de notre modèle.
- En répétant ainsi en boucle c'est quelques fonctions, nous arriverons à minimiser le coût de notre modèle. C'est ce qu'on appelle l'algorithme de la descente de gradient.

Travaille à faire :

1) Pour commencer importer les bibliothèque necessaires

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
```

2) Nous allons générer un Dataset aléatoire X, y comprenant 100 lignes et deux variables grâce à la fonction make_blobs que l'on trouve dans sklearn :

```
X, y = make_blobs(n_samples=100, n_features=2, centers=2, random_state=0)
y = y.reshape((y.shape[0], 1))

print('dimensions de X:', X.shape)
print('dimensions de y:', y.shape)
```

3) En utilisant la bibliothèque matplotlib, afficher le dataset généré sur un plan 2D

4) Créer la fonction d'initialisation dans laquelle on va passer la matrice X qui va nous servir de connaitre la dimension du vecteur W

```
def initialisation(X):
    ######################
    ## Votre code ici  ##
    ######################
    return (W, b)
```

Tester cette fonction et afficher les résultats obtenus et les shape de W et b.

5) Implémenter la fonction du modèle (Agrégation + Activation sigmoid). Cette fonction va prendre en paramètre X, W et b

```
def model(X, W, b):
    ######################
    ## Votre code ici  ##
    ######################
    return ## Votre code ici  ##
```

Tester cette fonction et afficher la forme de la valeur retournée. Conclure.

6) Implémenter la fonction cout Log Loss

```
def log_loss(A, y):
   ######################
    ## Votre code ici  ##
    ######################
    return ## Votre code ici  ##
```

Tester cette fonction et afficher la forme de la valeur retournée (ses dimensions).

7) Créer la fonction du gradient

```
def gradients(A, X, y):
    ######################
    ## Votre code ici  ##
    ######################
     return ## Votre code ici  ##
```

Tester cette fonction et vérifier la forme de la valeur retournée.

8) Implémenter la fonction update qui va faire la mise à jour des poids. Quels sont les paramètres à passer à cette fonction

```
def update( ## Votre code ici  ##):
    ######################
    ## Votre code ici  ##
    ######################
    return ## Votre code ici  ##
```

Tester cette fonction et vérifier la forme de la valeur retournée.

9) Maintenant pour implémenter notre premier perceptron, nous allons créer une fonction appelée **artifitiel_neuron**. Le rôle de cette fonction est d'implémenter l'algorithme d'apprentissage de

notre premier neurone artificiel en rassemblant toutes les méthodes créées précédemment. Cette fonction prendra en paramètre les entrés X et les sorties y, le pas d'apprentissage ainsi qu'un paramètre supplémentaire qui va déterminer le nombre d'itération que l'algorithme va effectuer. A la fin, la fonction retournera nos paramètres optimisés du modèle W et b.

```
def artificial_neuron(X, y, learning_rate = 0.1, n_iter = 100):
    ######################
    ## Votre code ici  ##
    ######################
    return ## Votre code ici  ##
```

10) Modifier la fonction précédente de façon à pourvoir visualiser l'évolution de la fonction cout en fonction des itérations effectuées. Et ceci pour s'assurer que notre modèle est bien appris et que la phase d'apprentissage a été déroulé avec succès. Pour cela, utiliser une courbe matplotlib.
11) Tester la fonction artificial_neurone et afficher la courbe d'apprentissage qui représente l'évolution du cout.

Effectuer de futures prédictions

Alors maintenant que nous avons un modèle nous pouvons nous en servir pour effectuer des prédictions. Par exemple si je prends une nouvelle plante que je mesure la longueur et la largeur de ses feuilles et que j'entre ces informations dans mon modèle celui-ci va me retourner la probabilité que la plante soit toxique car rappelez-vous la sortie de notre modèle c'est une fonction sigmoïde que l'on peut voir comme une probabilité toujours comprise entre 0 et 1. donc ce qu'on fait en général c'est qu'à partir du moment où cette probabilité est supérieur à 0,5 on dit que la plante est toxique et qu'elle appartient à la classe y=1.

Alors pour implémenter cela il va donc nous falloir créer une nouvelle fonction appelée **predict** dans laquelle nous ferons passer des données x dont on voudra faire la prédiction, ainsi que les paramètres de notre modèle W et b. Alors la première chose à faire dans cette fonction c'est de calculer la sortie du modèle représentée par les activations grâce à x, W et b. Puis ensuite ces activations lorsqu'elles seront supérieures au seuil de 0,5 on retournera la valeur 1 qui est la valeur de la classe 1.

12) Ecrire la fonction **predict**

```
def predict(X, W, b):
    ######################
    ## Votre code ici  ##
    ######################
    return ## Votre code ici  ##
```

13) Au niveau de la fonction **artificial_neurone**, après avoir fini l'apprentissage du modèle, calculer les prédictions **y_pred** pour toutes les données X de notre dataset. Ensuite afficher la

performance du modèle (accuracy). Pour cela, utiliser la fonction **accuracy_score** de la bibliothèque **sklearn** (package **sklearn.metrics**). Qu'est-ce que vous remarquez par rapport au performance du modèle ?

14) Utiliser la fonction **predict** pour faire la prédiction sur une nouvelle plante dont la longueur est 2 et la largeur est 1 (2, 1). Essayer d'afficher cette plante sur le graphique et afficher le résultat de la fonction **predict**.
15) Afficher la probabilité associée à cette plante. Conclure.
16) Tracer la frontière de décision qui sépare les deux classes. Rappelez-vous que la frontière de décision c'est l'ensemble des points pour lesquels z est égal à zéro. C'est aussi l'endroit pour lequel les probabilités sont supérieurs à 50%. Or, le point z pour lequel A est égal à 0,5 c'est à dire la fonction sigmoïde est égal à 0,5 c'est lorsque z est égal à zéro (z=0 ➔ a(z)=0.5). Donc pour connaître l'équation de la droite que l'on voudrait tracer ici il suffit de dire que c'est le l'ensemble des points pour lesquels z est égal à zéro c'est à dire l'ensemble des points x1, x2 pour lesquels w1*x1 + w2 *x2 + b = 0. Donc x2 = (- W1 * x1 - b) / W2.
    a. Tracer cette droite sur le repère x1, x2 (utiliser un intervalle entre -2 et 5 pour le repère x1).
    b. Interpréter le graphe et donner une conclusion par rapport aux erreurs faites par le modèle.

Partie 2

L'objectif de cette deuxième partie du TP est de développer un neurone artificiel qui soit capable de reconnaître s'il y a un chat ou un chien sur une photo. Nous avons un dataset contenant des photos de chats et de chiens. Ce que je vous propose de faire, c'est d'écrire un programme qui soit capable de reconnaître un chat ou un chien à partir de telles photos. Pour cela, vous allez utiliser tout le code qu'on a écrit jusqu'à maintenant.

1. Pour commencer nous allons charger le dataset utilisé en suivant les étapes ci-dessous :

a. Installer h5py : h5py vous permet d'ouvrir les fichiers au format hdf5

```
!pip install h5py
```

b. Utiliser le code suivant pour charger le dataset sous forme de train set et test set :

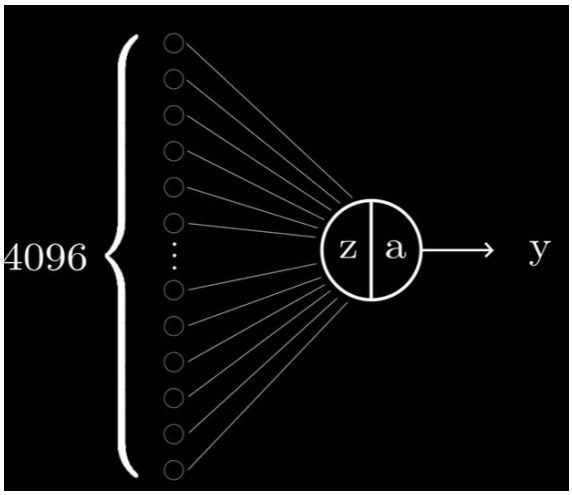
```python
import h5py
import numpy as np


def load_data():
    train_dataset = h5py.File('datasets/trainset.hdf5', "r")
    X_train = np.array(train_dataset["X_train"][:]) # your train set features
    y_train = np.array(train_dataset["Y_train"][:]) # your train set labels

    test_dataset = h5py.File('datasets/testset.hdf5', "r")
    X_test = np.array(test_dataset["X_test"][:]) # your train set features
    y_test = np.array(test_dataset["Y_test"][:]) # your train set labels

    return X_train, y_train, X_test, y_test
```

```python
X_train, y_train, X_test, y_test = load_data()
```

Le dataset se trouve dans le dossier **datasets** et se compose de données d'entrainement et de données de teste. X_train représente les photos et y_train représente la classe qu'on cherche à prédire (0 pour chat et 1 pour chien). N'oublier pas de mettre le dossier **datasets** dans votre répertoire de travail.

c. Examiner la dimension des tableaux contenant les données d'entrainement et de teste et vérifie si le dataset est équilibré ou non (train et test).
d. Afficher les 10 premières photos du dataset d'entrainement.
e. Exécuter la fonction artificial_neurone sur les données X_train et y_train. Qu'est-ce que vous remarquez ?

2. Notre dataset se compose de photos. Alors la question qui se pose c'est comment est-ce qu'on va faire pour fournir une photo à notre modèle de neurones artificiels. Parce que jusqu'à présent nous avons travaillé avec des données X et y. Alors une photo se compose d'un ensemble de pixel. Les pixels d'une photo peuvent être considérés comme des variables avec une valeur numérique comprise entre 0 et 255. Le pixel tout en haut à gauche c'est la première variable et le pixel tout en bas à droite est la dernière variable donc ici sur une photo de 64 pixels par 64 pixels nous avons en réalité 4096 variable. Alors pour les présenter à la machine c'est très simple, ce qu'on va faire c'est que nous allons aplatir nos photos, donc prendre chaque couche de pixels et les mettre sur une seule et même ligne. Une fois qu'on a fait cela, on peut exécuter tout le code qu'on a développée dans ce TP sans aucun problème.



Transformez le tableau X_train en un tableau (1000, 4096) en le nomant X_train_new

Ré-exécuter la fonction **artificial_neurone** sur les données X_train_new et y_train. Qu'est-ce que vous remarquez ?

Analyser les logs puis chercher et expliquez la cause du problème.

Corriger le problème et réexécuter la fonction

Donc votre mission pour cette partie du TP, consiste à développer un neurone artificiel qui soit capable de reconnaître s'il y a un chat ou un chien sur une photo. Pour faire cela, voici la démarche que vous allez suivre :

1. Normaliser le train_set et le test_set (0-255 -> 0-1) : En fait, que ça soit en machine learning ou en deep learning, il faut toujours normaliser nos données lorsqu'on utilise un algorithme de descente de gradient. Cela implique de mettre sur une même échelle toutes les variables dans un dataset afin d'éviter que les plus grandes valeurs écrasent les plus petites. Quand les variables sont sur une même échelle, la fonction cout évolue de façon similaire sur tous ses paramètres. Cela permet une bonne convergence de l'algorithme de la Descente du Gradient. Mais si une valeur est plus importante que l'autre, alors la fonction cout est compressée car le paramètre w associé à cette variable aura un grand impact sur la sortie A(Z) ce qui complique la convergence de la Descente du Gradient.

   En effet, dans notre cas, chaque image est codée en 8 bits, c'est à dire que, chaque pixel a une valeur comprise entre 0 et 255, 0 représentant le noir et 255 le blanc. Et bien ce qu'on veut faire c'est normaliser ses valeurs pour que chaque pixel soit en fait compris entre 0 et 1.

Maintenant, comment faire pour effectuer cette normalisation. Alors il existe différentes techniques comme la standardisation ou la normalisation min/max qui met toutes les variables sur une échelle de 0 à 1. La formule à appliquer pour la standardisation min max est la suivante :

$$X = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Dans notre cas est comme les valeur des pixels sont comprise entre 0 (noir) et 255 (blanc), alors la formule deviendra :

$$X = \frac{X - 0}{255 - 0} = \frac{X}{255}$$

Ecrire le code python qui permet de faire cette opération sur les deux ensembles train et test en divisant par la valeur max qui existe dans le train set.

2. flatten() les variables du train_set et du test_set (64x64 -> 4096) : Aplatir les images en les passant de de 64*64 pixels à des images qui sont représentés par une séquence de 4096 variable.

   Combien nous avons de paramètre dans le vecteur W ?

   Ecrire l'équation de Z avec les valeurs réelles des variables X et donner sa valeur. Ensuite remplacer les valeurs réelles de X par les valeurs normalisées et afficher la valeur de Z. Qu'est-ce que vous remarquez ?

3. Entrainer le modele sur le train set : Utiliser le code que nous avons développé précédemment. Dans cette étape, il faut tracer la courbe d'apprentissage (courbe de la fonction log loss) et trouver les bons hyper-params du modèle en modifiant le nombre d'itération et choisissant différentes valeurs du paramètre learning rate.

   Entrainer le modèle dans un premier temps avec un learning rate (lr) = 0.1 (En affichant toujours l'évolution de la fonction cout)

   Entrainer le modèle dans un deuxième temps avec un lr = 0.01

   Que'est ce que vous remarquez ?

   Changer le nombre d'itération à 1000 et Entrainer le modèle

   Calculer l'accuracy du modèle pour chaque itération et afficher sa la courbe d'évolution

   Entrainer maintenant le modèle avec 10000 itérations. Qu'est-ce que vous remarquez ?

4. Évaluer le modèle sur le test set (tracer également la courbe de Loss pour le test set)
5. Tirer des conclusions à propos de la qualité du modèle en comparant les deux courbes (modèle en overfitting ou underfitting)
6. Cimment faire pour améliorer notre modèle

Remarque : si vous rencontrez un problème avec le log_loss, utiliser la fonction de sklearn à la place.

Partie 3

Ecrire le code que nous avons fait de façon orienté objet

Bon courage.