Systèmes Intelligents

Rapport TD 1-2-3

Réalisé par :
Bou Saba Elie

Travail Présenté à :
Monsieur K. Naudin

1. TD 1 : Python et perceptron

1.1 Introduction

Dans ce premier TD, on a utilisé l'IDE Python PyCharm pour pratiquer quelques bases de programmation Python. On a créer 3 fichiers ayant chacun une extension .py qui sont : army.py, character.py et main.py où Army.py et character.py possèdent respectivement les classes Army et Character. En plus, on a utilisé le fichier CSV « characters » pour extraire les informations enregistrées dans ce fichier concernant les personnages de la série « The Witcher » : name, firstname, age, profession et boost de moral.

1.2 Travail réalisé

D'après le fichier main.py qui est le script principal, on va ouvrir et lire le fichier CSV « characters » et on va mettre les différentes informations des personnages qu'on a lu du fichier en sautant la première ligne dans une liste qui est initialement vide nommée character_list.

On va maintenant créer la classe Army et on va ajouter dans le constructeur un calcul d'une valeur de moral de base aléatoire en utilisant la méthode random
On va ajouter aussi une méthode get_total_morale() qui calcul la valeur de moral de base de l'armée multipliée par le coefficient de boost moral de son chef.

Après ça, on va créer dans le script principal une liste d'armée initialement vide nommée army_list puis on va attribuer à chaque personnage une armée et la mettre dans la liste. Après avoir bien insérer les armées de chaque personnage dans la liste army_list, on a calculé et affiché la valeur totale de moral de toutes les armées.

```
def TotalValeurArme(army_list):
    total = 0.0
    for army in army_list:
        total += army.get_total_morale()
    return total

totall = TotalValeurArme(army_list)
print("Le Boost total est: "+str(totall))
```

2. TD 2 : Python et perceptron

2.2 Introduction

Dans ce deuxième TD, nous avons réalisé notre premier apprentissage avec un perceptron dans le but que notre réseau de neurones apprenne la fonction AND tout en vérifiant que le modèle peut fonctionner grâce à la visualisation de sa surface d'erreur.

2.2 Travail réalisé

En premier lieu, on a créer une liste à 2 dimensions contenant les inputs possibles pour la fonction AND et ces valeurs sont : (0,0) , (0,1) , (1 ,0) , (1,1) et une liste à 1 dimension contenant les sorties attendues pour chaque paire d'inputs qui est : (0, 0, 0, 1)

En second lieu, on a mis en place une liste initialement vide nommé 'y' où on va mettre tous les résultats obtenus et on a créer aussi un tableau (11x11) nommé 'error' qu'on lui a attribué des valeurs de 0 partout au début.

Ensuite, on a appliqué une variation des valeurs de w1 et w2 qui sont les poids entre les valeurs -5 et 5 dans 2 boucles et pour chaque variation de la valeur de w on va réinitialiser la liste 'y' des résultats. Aussi, on a créer une boucle qui entre dans chaque paire de valeur de la liste 'a' qui contienne les inputs possibles et on va faire pour chaque paire le calcul suivant : w1*a1+w2*a2 et on test la valeur obtenue. Si la valeur obtenu est <=0, on ajoute 0 à la liste 'y', sinon on ajoute 1. De plus on a fait une autre boucle qui va calculer l'erreur à partir de la formule suivante :

$$E = \frac{1}{2}\sum_{\forall i}(y_i - t_i)^2$$

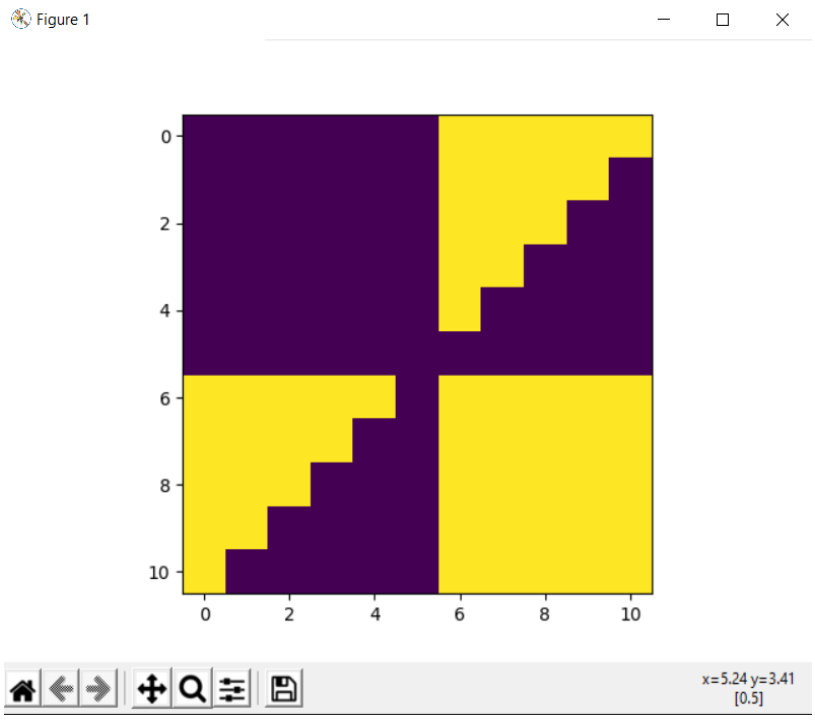
On va ajouter la valeur retournée dans le tableau 'error' et on incrément la valeur d'index pour traverser dans le tableau.

```
index = [0,0]
for w1 in range(-5,6):
   for w2 in range(-5,6):
      y = []
      for i in a:
         temp = w1*i[0]+w2*i[1]
         if(temp<=0):
            y.append(0)
```

```
            else:
                y.append(1)
        temp2 = 0.0
        for i in range(0,4):
            temp2 += 0.5*(y[i]-t[i])**2

        error[index[0]][index[1]] = temp2
        index[1] += 1
    index[1] = 0
    index[0] += 1
```

À la fin, avec l'aide de la librairie 'matplotlib', on a bien affiché l'image représentant la surface d'erreur et on voit bien dans la capture ci-dessous la surface d'erreur qu'on a obtenu après le calcul.



On va créer maintenant une classe perceptron. Le constructeur de cette classe prend le nombre d'inputs du perceptron, le nombre d'epochs et le learning rate. Tous les poids ont initialement une valeur de poids de 0, ayant aussi un biais de valeur 1.
On va écrire une méthode predict prenant en argument une paire d'inputs. Cette méthode rend 1 si la valeur calculée est supérieure à 0.5 et rend 0 sinon. Cette méthode est écrite sous la forme suivante :

```
def predict(self, a):
    y = np.dot(self.Poids, a)
    if y > 0.5:
        return 1
    else:
        return 0
```

y = np.dot(self.Poids, a) se traduit mathematiquement par :
y = a [0] * self.poids[0] + a[1] * self.poids[1] + a[2] * self.poids[2]

On va réaliser maintenant une méthode train qui prend la liste des inputs possibles et les sorties attendues pour chaque paire d'inputs. Autant de fois qu'il y a d'epochs on va Faire une prédiction pour chaque paire d'inputs et corriger à la fin la valeur des poids en utilisant la loi de Widrow-Hoff

```
def train(self, x, y):
    t = self.calculPredictionTotal(x)
    erreur = self.calculErreur(t, y)

    for epoch in range(self.NombreEpochs):

        for i in range(len(x)):
            x2 = x[i]
            t2 = self.predict(x2)
            for j in range(len(x2)):
                temp = (self.LearningRate * (y[i] - t2) * x2[j])
                self.Poids[j] += temp

        t = self.calculPredictionTotal(x)
        erreur = self.calculErreur(t, y)
        print(erreur)
```

```
def calculPredictionTotal(self, x):
    t = []
    for i in x:
        t.append(self.predict(i))

    return t


def calculErreur(self, t, y):
    return np.sum((np.subtract(t, y)) ** 2) / 2
```

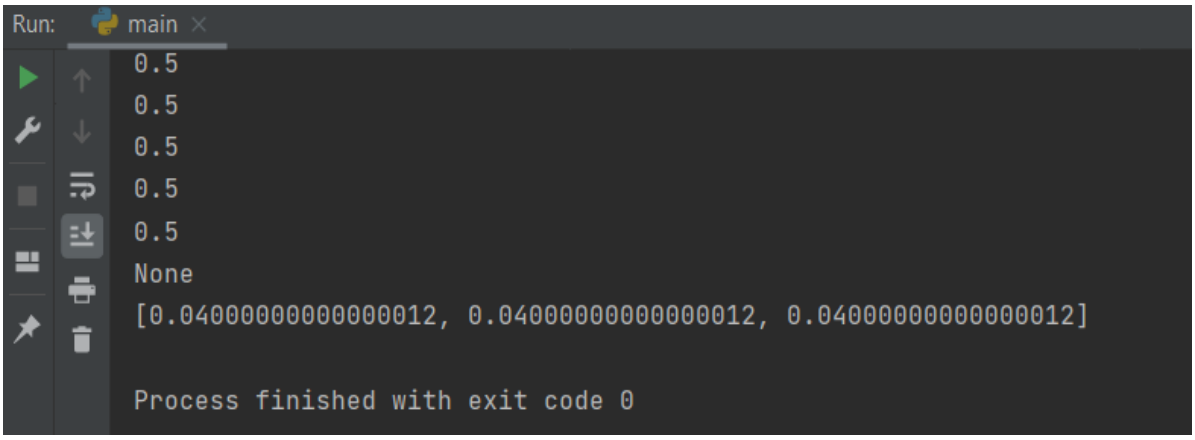
Dans le script principale main, on a créer un perceptron avec un nombre d'inputs égale à 3, un nombre d'epochs de 400 et un Learning rate de 1e-4. Après, on a initialisé les inputs dans x et les valeurs attendus dans y. A la fin, on a sauvegardé les valeur des poids dans le fichier 'data.csv'

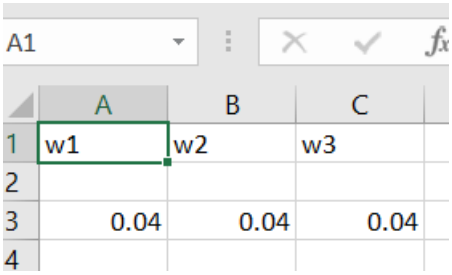
```
perceptron = Perceptron(3, 400, 1e-4)
x = [
    [1, 0, 0],
    [1, 0, 1],
    [1, 1, 0],
    [1, 1, 1]
]
y = [0, 0, 0, 1]

print(perceptron.train(x, y))

print(perceptron.get_poids())

with open("data.csv", "w") as file:
    writer = csv.writer(file)
    writer.writerow(["w1", "w2", "w3"])
    writer.writerow(perceptron.get_poids())
```





Les captures illustrées ci-dessus affiche bien les valeurs des poids trouvés et ces poids ont été bien enregistrés dans le fichier ‘data.csv’