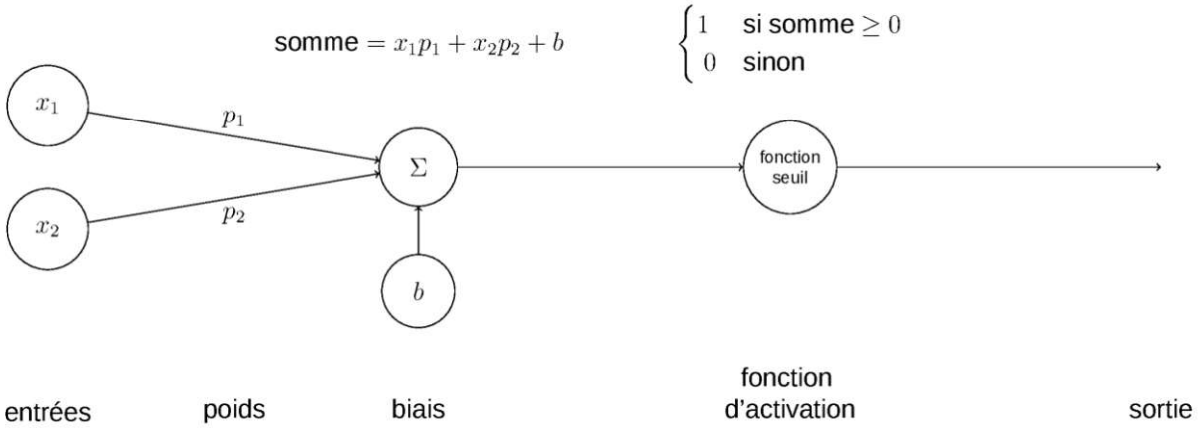
Exercice 2 (6 points)

Cet exercice porte sur les thèmes langages, programmation et algorithmique.

Dans le cadre d'un concours national intitulé "Jeunes Talents IA 2025", une équipe constituée d'élèves de terminale NSI est sélectionnée pour développer un système d'intelligence artificielle capable de détecter si un email est un spam ou non pour la messagerie d'un établissement scolaire. Après réflexion, l'équipe décide d'utiliser un perceptron pour classer les messages comme spam ou non spam.

Le perceptron est un neurone artificiel qui imite très simplement le fonctionnement d'un neurone biologique. L'équipe choisit d'entraîner le perceptron à l'aide d'un algorithme d'apprentissage supervisé : on présente au perceptron des exemples, avec les bonnes réponses, afin qu'il ajuste automatiquement ses poids dans le but d'améliorer ses prédictions.

Le fonctionnement d'un perceptron est décrit ci-après :



Le perceptron dispose d'un tuple de n poids $p = (p_1, p_2, p_3, \ldots, p_n)$ et d'un biais b. Il reçoit en entrée un tuple de n valeurs numériques d'entrée $x = (x_1, x_2, x_3, \ldots, x_n)$.

Il calcule la somme pondérée avec biais des valeurs d'entrées :
$somme = x_1 \times p_1 + x_2 \times p_2 + \ldots + x_n \times p_n + b$.

Puis il applique une fonction d'activation. Dans cet exercice la fonction d'activation est la fonction seuil qui renvoie 1 si la somme pondérée avec biais est supérieure ou égale à 0, et 0 sinon.

Le schéma ci-dessus illustre le cas $n = 2$.

Pour commencer, l'équipe considère un perceptron à deux entrées ayant pour caractéristiques les poids $p_1 = 0.7$ et $p_2 = -0.2$, et un biais $b = 0.1$.

1. Calculer la valeur $somme$ de la somme pondérée avec biais du perceptron lorsque les valeurs d'entrée sont $x_1 = 1$ et $x_2 = 3$ et en déduire la valeur de sortie $sortie$ du perceptron si la fonction d'activation est la fonction seuil (sortie à 1 si somme supérieure ou égale à 0, sinon 0).

Voici le début d'une classe `DetecteurSpam` basée sur un perceptron :

```
 1 class DetecteurSpam:
 2     def __init__(self, nb_entrees):
 3         self.biais = 0.0
 4         self.poids = [0.0 for i in range(nb_entrees)]
 5
 6     def get_biais(self):
 7         return self.biais
 8
 9     def set_biais(self, nouveau_biais):
10         self.biais = nouveau_biais
11
12     def get_poids(self):
13         return self.poids
14
15     def set_poids(self, nouveaux_poids):
16         assert len(nouveaux_poids) == len(self.poids)
17         self.poids = nouveaux_poids
```

2. Donner le nom d'une méthode et le nom d'un attribut de cette classe.

3. Donner une suite d'instructions Python permettant de créer un objet `detecteur` (instance de la classe `DetecteurSpam`) correspondant à un détecteur à deux entrées, puis de remplacer ses poids et son biais par les valeurs `p1 = 0.7`, `p2 = -0.2` et `b = 0.1` en utilisant les méthodes de la classe.

4. Expliquer le rôle de la ligne 16 du code (`assert len(nouveau_poids) == len(self.poids)`).

5. Écrire la méthode `fonction_activation` qui prend en paramètres `valeur` un nombre et qui renvoie 1 si `valeur` est supérieur ou égal à 0, et 0 sinon.

La méthode `prediction` ci-après :

- prend en paramètre une liste `entrees` de nombres ;
- calcule la somme pondérée avec biais des valeurs de la liste `entrees` ;
- applique la fonction d'activation ;
- renvoie la valeur de sortie de la fonction d'activation.

```
1     def prediction(self, entrees):
2         somme = 0
3         for i in range(len(entrees)):
4             somme += ...
5         somme += ...
```

```
6           sortie = self.fonction_activation(...)
7           return ...
```

On considère que le nombre d'éléments de la liste `entrees` est identique au nombre d'entrées (ou paramètres) du détecteur. Exemple d'utilisation :

```
>>> # exemple avec un détecteur à trois entrées
>>> detecteur.get_biais()
0.2
>>> detecteur.get_poids()
[0.5, -0.1, 0.2]
>>> detecteur.prediction([0.1, 0.3, 0.2])
1
```

6. Recopier et compléter la méthode `prediction`.

7. Donner le coût d'exécution temporel de la méthode `prediction` en fonction de la taille n des entrées.

La méthode `entrainement` ci-après permet d'ajuster les poids et le biais du perceptron selon un algorithme d'apprentissage supervisé. Cette méthode prend en paramètres une liste `echantillon` d'entrainement constituée de listes de nombres, une liste `cibles` de bonnes réponses attendues, un taux d'apprentissage `taux` et renvoie le nombre d'erreurs obtenues à la fin de l'entrainement.

```
1       def entrainement(self, echantillon, cibles, taux=0.1):
2           nb_erreurs = 0
3           for i in range(len(echantillon)):
4               entrees = echantillon[i]
5               cible = ...
6               # calcul de la prédiction
7               predict = ...
8               if ...:
9                   # prédiction incorrecte
10                  nb_erreurs += 1
11                  erreur = cible - predict
12                  for j in range(len(self.poids)):
13                      self.poids[j] += ... * ... * entrees[j]
14                  self.biais += taux * erreur
15          return nb_erreurs
```

Pour l'ajustement des poids et du biais, on applique la méthode `prediction` pour chaque élément de la liste `echantillon` :

- si la prédiction est correcte, on ne change rien ;
- si la prédiction est incorrecte :
  - on calcule l'erreur `erreur = cible - prediction` ;

- on ajuste les poids selon la règle : `nouveau_poids = ancien_poids + taux * erreur * entree` ;
- puis on ajuste le biais selon la règle : `nouveau_biais = ancien_biais + taux * erreur`.

Lorsque tous les éléments de la liste `echantillon` ont été traités, on dit que l'on a réalisé une **époque** d'entrainement.

Un des membres de l'équipe propose les listes `echantillon` et `cibles` suivantes pour tester la méthode `entrainement`.

```
1 echantillon = [[0.5, 0.3], [0.2, 0.1], [0.4, 0.8], [0.7,
0.9]]
2 cibles = [0, 1, 1, 1]
```

8. Donner les valeurs de `echantillon[1]` et `echantillon[2][1]`.
9. Recopier et compléter les lignes 5, 7, 8 et 13 du code de la méthode `entrainement`.

L'équipe veut maintenant créer un détecteur de spam à 2 caractéristiques d'entrée et l'entraîner sur les données du tableau ci-dessous pour apprendre à détecter les emails avec "URGENT" ET "GRATUIT".

Contient "URGENT" ?	Contient "GRATUIT" ?	Est un SPAM ?
0 (non)	0 (non)	0 (non)
0 (non)	1 (oui)	0 (non)
1 (oui)	0 (non)	0 (non)
1 (oui)	1 (oui)	1 (oui)

10. Écrire un programme permettant de créer et d'entrainer le détecteur sur les listes `emails = [[0, 0], [0, 1], [1, 0], [1, 1]]` et `spams = [0, 0, 0, 1]`. pendant au maximum 1000 époques, puis d'afficher dans la console le nombre d'époques nécessaires pour que le détecteur ne fasse plus d'erreurs (si c'est possible), et -1 sinon.