Asie – 2026 – sujet1 - Correction

Exercice 2 (6 points)

1. Somme et sortie du perceptron

somme = x1 × p1 + x2 × p2 + b = 1 × 0,7 + 3 × (−0,2) + 0,1 = 0,7 − 0,6 + 0,1 = **0,2**.

0,2 ≥ 0, donc la fonction seuil renvoie **sortie = 1**.

2. Une méthode et un attribut

Méthode : par exemple **get_biais** (ou set_biais, get_poids, set_poids, init).

Attribut : **biais** (ou poids).

3. Création et réglage du détecteur

```
detecteur = DetecteurSpam(2)
detecteur.set_poids([0.7, -0.2])
detecteur.set_biais(0.1)
```

👉*Commentaire : on passe par les méthodes set_poids et set_biais, comme demandé, plutôt que de modifier directement les attributs.*

4. Rôle de la ligne 16

L'assertion vérifie que la nouvelle liste de poids a **autant d'éléments** que l'ancienne, c'est-à-dire autant que d'entrées du détecteur. Sinon, le programme s'arrête avec une AssertionError au lieu d'enregistrer des poids incohérents.

5. Méthode fonction_activation

```
    def fonction_activation(self, valeur):
        if valeur >= 0:
            return 1
        else:
            return 0
```

6. Méthode prediction

```
    def prediction(self, entrees):
        somme = 0
        for i in range(len(entrees)):
            somme += self.poids[i] * entrees[i]
        somme += self.biais
```

```
        sortie = self.fonction_activation(somme)
        return sortie
```

👉*Commentaire : on vérifie l'exemple : 0,5 × 0,1 − 0,1 × 0,3 + 0,2 × 0,2 + 0,2 = 0,26 ≥ 0, la prédiction vaut bien 1.*

7. Coût de prediction

La boucle fait n tours, chacun en temps constant : le coût est **linéaire**, en **O(n)**.

8. Valeurs dans echantillon

echantillon[1] vaut **[0.2, 0.1]** (deuxième élément) et echantillon[2][1] vaut **0.8** (deuxième nombre du troisième élément [0.4, 0.8]).

9. Méthode entrainement (lignes 5, 7, 8 et 13)

```
            cible = cibles[i]                                   # ligne 5
            # calcul de la prédiction
            predict = self.prediction(entrees)                  # ligne 7
            if predict != cible:                                # ligne 8
                # prédiction incorrecte
                nb_erreurs += 1
                erreur = cible - predict
                for j in range(len(self.poids)):
                    self.poids[j] += taux * erreur * entrees[j]      # ligne 13
                self.biais += taux * erreur
```

👉*Commentaire : la ligne 13 applique la règle nouveau_poids = ancien_poids + taux × erreur × entrée.*

10. Entraînement sur « URGENT » ET « GRATUIT »

```
detecteur = DetecteurSpam(2)
emails = [[0, 0], [0, 1], [1, 0], [1, 1]]
spams = [0, 0, 0, 1]
epoque = 0
resultat = -1
while epoque < 1000 and resultat == -1:
    epoque = epoque + 1
    if detecteur.entrainement(emails, spams) == 0:
        resultat = epoque
print(resultat)
```

👉*Commentaire : on s'arrête à la première époque sans erreur. En exécutant ce programme, Python affiche 4 : les époques 1, 2 et 3 corrigent des erreurs, et la 4e n'en fait plus (poids 0,2 et 0,1, biais −0,2). On peut aussi répondre 3 (nombre d'époques d'apprentissage), en le précisant.*

👉*Commentaire : surprise des flottants : après trois corrections, le biais vaut −0.20000000000000004 et non −0,2. Pour l'email (1, 0), la somme 0,2 − 0,20000000000000004 est légèrement négative, donc la prédiction 0 est correcte. Avec des calculs exacts, cette somme vaudrait 0, la sortie serait 1 et il faudrait 6 époques.*

👉*Commentaire : le ET logique est « linéairement séparable » : une droite sépare le cas (1, 1) des trois autres, le perceptron peut donc l'apprendre. Pour le OU exclusif, ce n'est pas possible : le programme afficherait -1.*