

TP 1

Implémentation du Perceptron avec la Descente de Gradient

Objectifs pédagogiques

- Comprendre les bases mathématiques du perceptron.
- Implémenter progressivement un perceptron binaire avec la descente de gradient.
- Renforcer les concepts de la fonction coût et des mises à jour des paramètres.
- Développer des compétences en programmation Python.

Étape 1 : Concepts de base

1. Question conceptuelle

- Expliquez en une phrase ce qu'est un perceptron.

2. Exercice manuel

- Une donnée $X = [2, 3]$, un vecteur de poids $w = [0.5, 1.5]$ et un biais $b = -2$ sont donnés. Calculez la sortie du perceptron avec une fonction sigmoïde.

$$z = w^T X + b$$

$$y_{pred} = a(z)$$

Question : Que se passe-t-il si vous augmentez le biais b ? Essayez avec $b = 0$ et $b = 2$.

Étape 2 : Génération et visualisation des données

1. Générer des données linéairement séparables avec `make_blobs`.
 - Paramètres suggérés : `n_samples=100`, `centers=2`, `cluster_std=1.5`.
2. Affichez ces données avec `matplotlib`.

Questions :

- Combien de points appartiennent à chaque classe ?
- Pensez-vous que le perceptron peut séparer ces données ?

Étape 3 : Implémentation de la fonction sigmoïde

1. Implémentez la fonction **sigmoïde** :

```
def sigmoid(z):  
    # Complétez le calcul de la fonction sigmoïde  
    return
```

2. Testez-la avec une série de valeurs de `z`.

Questions :

- Quelle est la sortie pour $z = 0$? Pourquoi ?
- Tracez la courbe de la sigmoïde pour z entre -10 et 10. Que remarquez-vous ?

Étape 4 : Fonction coût

1. Implémentez la fonction coût logistique binaire :

```
def cost_function(y, y_pred):  
    # Calcule la fonction coût logistique  
    return
```

2. Testez-la avec :

$y = [1, 0, 1, 0]$ et $y_{pred} = [0.9, 0.1, 0.8, 0.2]$.

Question :

- Que se passe-t-il si les prédictions y_{pred} sont proches des étiquettes y ? Et si elles sont très éloignées ?

Étape 5 : Descente de gradient

1. Implémentez une étape de la descente de gradient. Utilisez les formules pour mettre à jour w et b .

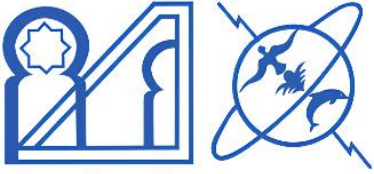

```
def gradient_descent_step(X, y, w, b, alpha):  
    m = len(y)  
    z = np.dot(X, w) + b  
    y_pred = sigmoid(z)  
    # Complétez le calcul de dw  
    dw = ...  
    # Complétez le calcul de db  
    db = ...  
    # Mettez à jour w et b  
    w -= ...  
    b -= ...  
  
    return w, b
```

Question :

- Qu'arrive-t-il aux poids si le taux d'apprentissage α est trop grand ?
Essayez $\alpha = 0.1, 1$, et 10 .

Étape 6 : Entraînement du perceptron

1. Implémentez une boucle complète d'entraînement pour optimiser les poids et biais sur plusieurs itérations.

 <p>Faculté des Sciences Université Mohammed V de Rabat</p>	<p>Deep Learning MSID 2024 / 2025</p>	 <p>جامعة محمد الخامس كلية العلوم الرباط</p>
--	---	---

Question :

- Après combien d'itérations le coût commence-t-il à se stabiliser ? Pourquoi ?

Étape 7 : Évaluation

- Implémentez une fonction `predict` pour prédire les étiquettes.
- Évaluez la précision sur le jeu de données.

Questions :

- Votre perceptron est-il parfait ? Pourquoi pourrait-il échouer dans certains cas ?
- Comment pourriez-vous améliorer ce modèle ?