COMPTE RENDU TD°1

Intelligence Artificielle & Optimisation

DUNAND Tom - 5A ILIA

Entraînement du programme pour la porte AND

Opération effectuée

Lancement du programme fourni tel quel afin d'analyser le résultat de celui-ci.

Observations

Le programme implémente l'entraînement d'un perceptron sur la fonction logique AND. La sigmoid permet de retourner une sortie entre 0 et 1. Un graphe matplotlib permet de visualiser l'évolution des poids au cours de l'entraînement. Un fichier CSV enregistre les poids finaux.

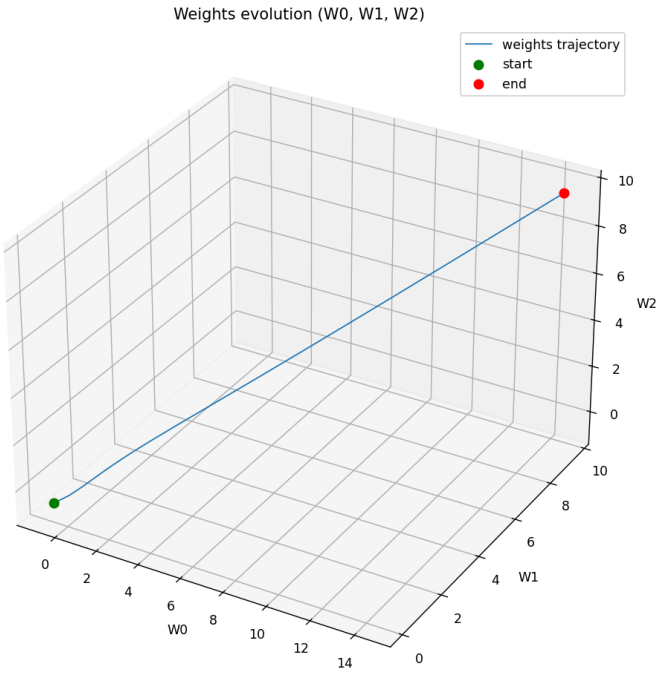
Visualisation de l'évolution des poids

Opérations effectuées

Ajout d'un tableau "weights_history" pour enregistrer l'évolution des poids à chaque itération et création d'un graphique 3D pour visualiser l'évolution dans les itérations des poids.

Observations relevées

La visualisation 3D montre une trajectoire convergente depuis les poids aléatoires initiaux vers un point fixe. La trajectoire n'est plutôt linéaire mais montre tout de même les ajustements successifs lors de la présentation des différents exemples d'entraînement.



Commentaires

La visualisation 3D permet de comprendre intuitivement comment l'algorithme de descente de gradient ajuste progressivement les poids. La convergence vers un point stable montre que l'algorithme trouve effectivement un minimum.

Test de différents hyperparamètres

Opérations effectuées
Tests effectués en faisant varier :

- Learning rate : 0.1, 0.5, 0.7, 1.5
- Itérations : 100, 500, 1000, 2000

Observations relevées
Variation du learning rate :

- 0.1 : Convergence très lente, nécessite plus d'itérations pour atteindre une solution acceptable
- 0.5 : Convergence stable et progressive
- 0.7 : Bon compromis, convergence rapide et stable
- 1.5 : Oscillations importantes, risque d'instabilité, la convergence est moins stable

Variation du nombre d'itérations :

- 100 itérations : Souvent insuffisant, les poids n'ont pas complètement convergé
- 500 itérations :Suffisant pour converger
- 1000 itérations : Convergence complète assurée
- 2000 itérations : Aucun gain significatif après convergence

Commentaires
Le coefficient d'apprentissage est important : trop faible, l'apprentissage est lent, trop élevé, il provoque des oscillations et peut empêcher la convergence. La valeur de 0.7 représente un bon équilibre pour ce problème. Pour le nombre d'itérations, 500 suffisent, mais 1000 itérations garantissent une convergence stable quelle que soit l'initialisation aléatoire des poids. Au-delà, on observe une phase de plateau où les poids ne changent presque plus.

Modification pour une porte OR

Opérations effectuées
Modification de la ligne définissant la sortie désirée :

```
desired_out = np.array([0, 1, 1, 1])
```

Exécution avec les mêmes paramètres qu'auparavant (lr=0.7, 1000 itérations).

Observations relevées

La convergence est tout aussi rapide que pour la porte AND. L'erreur diminue rapidement et se stabilise
.

Commentaires
La porte OR est également linéairement séparable et donc facilement apprise par le perceptron.

Modification pour une porte XOR

Opérations effectuées

Modification de la ligne définissant la sortie désirée :

```
desired_out = np.array([0, 1, 1, 0])
```

Exécution avec les mêmes paramètres (lr=0.7, 1000 itérations).

Observations relevées

Le réseau ne converge pas vers une solution correcte :

- L'erreur est négative et augmente
- Les poids continuent de varier sans atteindre de valeur stable
- La fonction ne parvient pas à modéliser correctement XOR

Des tests avec différents hyperparamètres donnent le même résultat : échec de la convergence.

Commentaires

Cet échec est attendu. La fonction XOR n'est pas linéairement séparable : il est impossible de tracer une seule droite qui sépare correctement les exemples positifs des exemples négatifs dans l'espace des entrées.