Compte Rendu TD1 : Entraînement d'un Perceptron sur des Portes Logiques

Louis Thin

7 décembre 2025

1 Introduction

Dans ce TD, on devait implémenter et étudier un perceptron simple pour voir s'il pouvait apprendre différentes portes logiques (AND, OR, XOR). L'objectif était de mieux comprendre comment un modèle aussi élémentaire peut pourtant capturer certains comportements logiques, mais aussi où se situent ses limites. En travaillant sur plusieurs types de portes, on peut comparer la dynamique d'apprentissage et l'évolution du perceptron selon que la fonction est linéairement séparable ou non.

2 Entraînement du programme pour la porte AND

2.1 Opérations effectuées

Avant de procéder aux expérimentations, il est important de comprendre la structure générale du perceptron utilisé. Le programme `perceptron_learn.py` contient un perceptron à une seule couche avec :

- 2 entrées (x1, x2)
- un biais
- 3 poids (W0, W1, W2)
- une fonction d'activation sigmoïde $\sigma(y) = \frac{1}{1+e^{-y}}$
- une sortie désirée : $[0, 0, 0, 1]$
- un taux d'apprentissage : $\epsilon = 0.7$

L'entraînement consiste à présenter successivement les quatre combinaisons possibles de la porte AND. À chaque itération, le perceptron calcule une prédiction, puis mesure l'erreur entre cette sortie et la valeur attendue. Cette erreur sert ensuite à ajuster les poids selon la règle de mise à jour :

$$W_i = W_i + \epsilon \cdot \text{entrée}_i \cdot \delta$$

où δ représente l'écart entre la sortie désirée et la sortie produite. Ainsi, le perceptron apprend progressivement les bons paramètres pour séparer les données en deux classes.

2.2 Observations

En exécutant le code :

- l'erreur diminue assez vite ;

— la convergence arrive généralement avant 1000 itérations ;
— la prédiction pour (1,1) devient proche de 1 tandis que les autres restent proches de 0.

Ces résultats montrent que le perceptron s'adapte rapidement à la structure linéaire du problème. Le fait que la sortie pour (1,1) augmente en dernier est logique, car c'est la seule entrée positive attendue, ce qui nécessite un ajustement subtil des poids.

2.3 Commentaires

Comme prévu, la porte AND est linéairement séparable, donc un perceptron suffit à l'apprendre. Le taux d'apprentissage de 0.7 fonctionne plutôt bien ici : il permet un apprentissage rapide mais peut parfois provoquer de légères oscillations selon l'initialisation. Ce comportement illustre l'équilibre délicat entre stabilité et vitesse d'apprentissage.

3 Visualisation de l'évolution des poids

3.1 Opérations effectuées

Afin de mieux comprendre comment le perceptron ajuste ses paramètres, j'ai utilisé `perceptron_weights_evolution.py`. Ce script permet d'observer non seulement la convergence des poids, mais aussi leur trajectoire au cours du temps. Pour cela, j'ai ajouté :

— un tableau qui stocke les poids à chaque itération ;
— un graphique montrant l'évolution de W0, W1 et W2 ;
— un export CSV pour visualiser les valeurs en dehors du programme.

Cette approche rend le processus d'apprentissage beaucoup plus transparent.

3.2 Observations

En observant le graphique :

— les poids se stabilisent après un certain temps ;
— W1 et W2 deviennent souvent proches, ce qui est logique puisque les entrées jouent le même rôle dans AND ;
— W0 reste négatif, servant de seuil.

On observe souvent une phase initiale où les poids évoluent rapidement, suivie d'une zone de plus faible amplitude : c'est typique des algorithmes de descente de gradient.

3.3 Commentaires

On voit bien que le modèle ajuste surtout les poids au début, puis les modifications deviennent de plus en plus petites. Cela confirme que l'algorithme descend progressivement vers un minimum. Visualiser ces courbes permet de mieux appréhender la dynamique interne du perceptron, souvent invisible lorsqu'on se contente des valeurs finales.

4 Test de différents hyperparamètres

4.1 Opérations effectuées

Dans un second temps, je me suis intéressé à l'impact du taux d'apprentissage. Le script `test_hyperparameters.py` teste plusieurs taux : 0.1, 0.3, 0.5, 0.7, 1.0 et 1.5.

Pour chaque cas, le programme réinitialise les poids, effectue l'apprentissage et trace l'erreur. Cette méthodologie permet de comparer objectivement les comportements selon les réglages.

4.2 Observations

Les comportements observés :

— **Taux faible** : apprentissage lent mais stable.

— **Taux moyen** : bon compromis vitesse/stabilité.

— **Taux élevé** : parfois des oscillations ou une non-convergence.

On constate que la valeur du taux d'apprentissage conditionne entièrement la vitesse et la stabilité de la convergence.

4.3 Commentaires

Ce test montre bien l'importance du taux d'apprentissage : trop petit, on avance doucement ; trop grand, on saute partout sans converger. Dans notre cas, 0.5–0.7 fonctionne le mieux. Cette expérimentation met en lumière que les hyperparamètres ne sont pas de simples détails, mais des éléments déterminants dans la réussite de l'apprentissage.

5 Modification pour la porte OR

5.1 Opérations effectuées

Pour OR, j'ai modifié uniquement la sortie désirée :

$$[0, 1, 1, 1]$$

Le code et les mécanismes d'apprentissage restent identiques, ce qui permet de comparer directement les trajectoires de convergence entre AND et OR.

5.2 Observations

Les résultats montrent que :

— l'erreur diminue encore plus vite que pour AND ;

— les poids convergent vers des valeurs différentes (W0 peut devenir positif) ;

— les prédictions deviennent correctes assez rapidement.

La fonction OR étant encore plus simple à séparer linéairement, le perceptron s'y adapte naturellement.

5.3 Commentaires

Comme AND, OR est linéairement séparable. Le perceptron n'a donc aucun problème à apprendre cette fonction. Cet exercice montre que différentes portes peuvent être apprises sans modifier le modèle, uniquement en changeant les données cibles.

6 Modification pour la porte XOR

6.1 Opérations effectuées

Pour XOR, j'ai mis comme sorties désirées :

$$[0, 1, 1, 0]$$

Cette fois encore, le code reste le même, ce qui permet d'observer directement l'échec du perceptron dans sa configuration simple.

6.2 Observations

Contrairement aux deux autres portes :
— l'erreur ne converge jamais vraiment ;
— les prédictions restent incorrectes ;
— les poids continuent de bouger sans trouver une solution stable.
Ce comportement erratique est typique des problèmes non linéairement séparables traités par un modèle trop simple.

6.3 Commentaires

Le perceptron échoue car XOR n'est pas linéairement séparable. Pour capturer une telle relation, il faudrait ajouter une couche cachée et utiliser un réseau de neurones plus avancé. C'est une limite connue des perceptrons simples, et cette expérience permet d'observer concrètement cette contrainte théorique.