Nom : Mathis Pezet

Compte-Rendu TP Perceptron

Introduction

Le but de ce TP était de comprendre comment marche le Perceptron. C'est un des premiers et des plus simples modèles de neurone artificiel.

On devait :

- Comprendre le fonctionnement d'un perceptron simple.
- Le coder en Python.
- Voir ses limites, surtout sur des problèmes qu'il n'arrive pas à résoudre.
- L'essayer sur des vraies données pour voir ce que ça donne.

Méthodes

Pour faire ça, j'ai utilisé Python et j'ai codé deux classes principales :

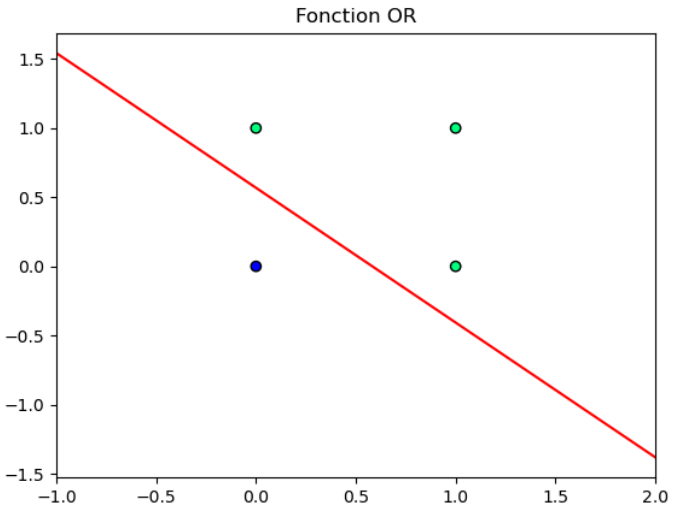
- C'est un seul neurone. Il sert à séparer des données en deux groupes. Il apprend en changeant ses poids quand il se trompe.
- Comme le premier ne gère que 2 classes, celui-là en gère plus. Il utilise plusieurs perceptron simple:: un pour chaque classe, qui apprend à la reconnaître par rapport à toutes les autres.

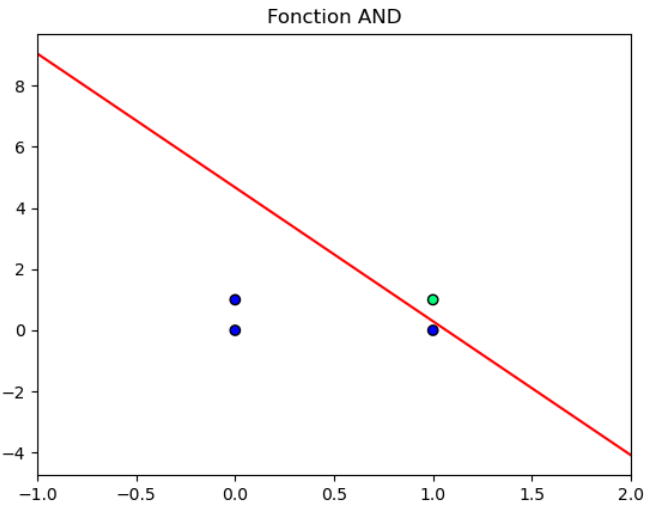
Résultats

Tests sur fonctions logiques

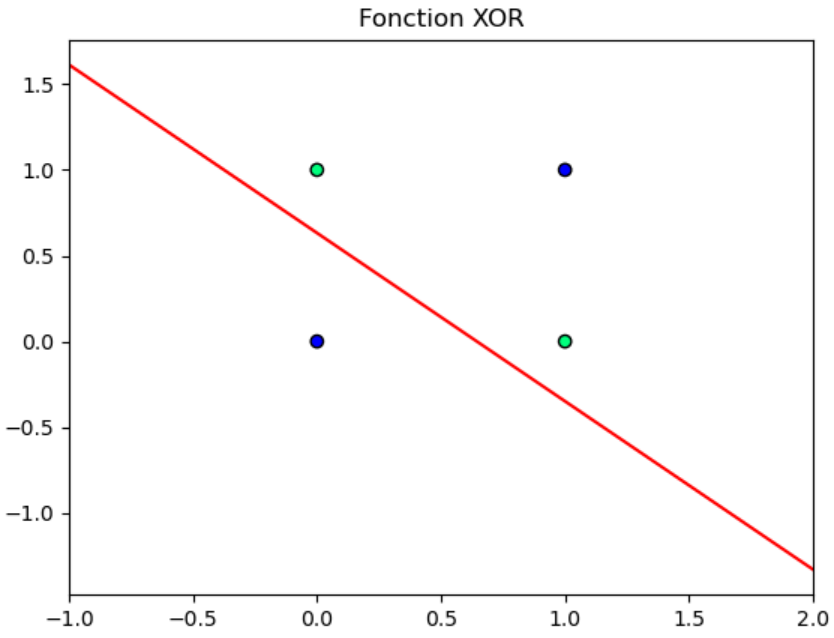
D'abord, j'ai testé le perceptron simple avec les fonctions logiques ET, OU et XOR.

- Pour ET et OU, ça a bien marché. J'ai vu que 30 époques suffisent pour un résultat constant. Par contre, la solution change un peu à chaque fois : Le perceptron tend à trouver une solution similaire à chaque fois mais jamais exactement là même.





- Pour XOR, le perceptron simple n'arrive jamais à le résoudre totalement, On a un score de 0.5 ou 0.75 dans le meilleur des cas, mais jamais de 1. C'est normal parce que on ne peut pas résoudre le XOR avec une application linéaire (une seule droite tracée). On a besoin d'au moins 2 droites, le perceptron simple ne peut donc pas résoudre le problème à cause de ça.



Analyse de convergence

Ici, j'ai regardé l'effet du taux d'apprentissage (la vitesse à laquelle le modèle apprend).

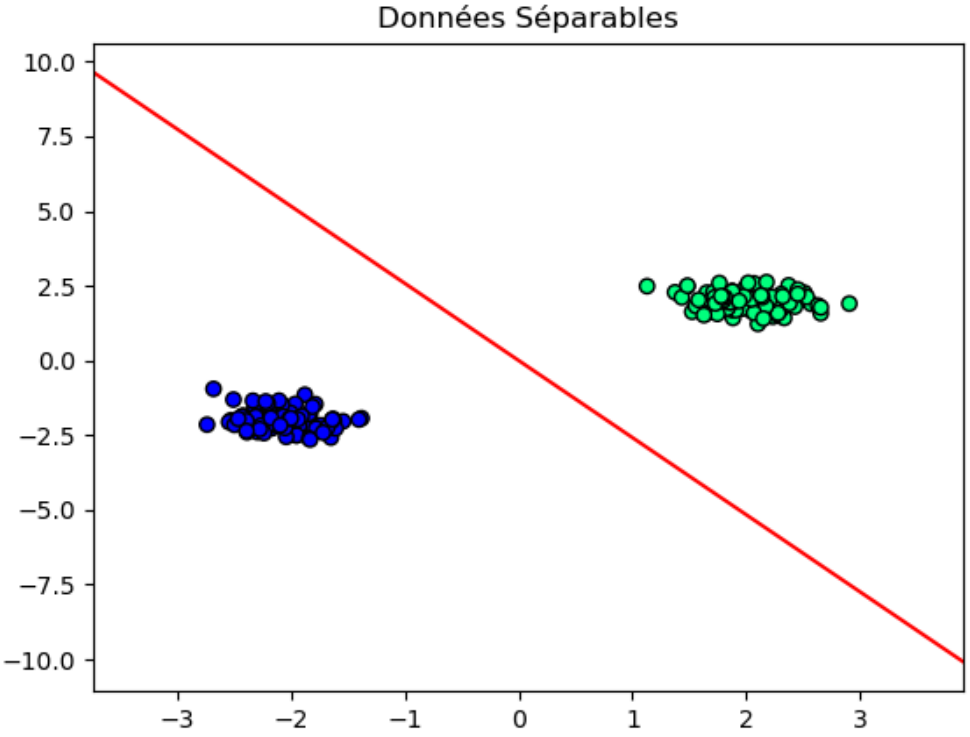
Q1) Si le taux est trop grand ? Si n est trop grand le perceptron n'a pas le temps d'apprendre de son itération.

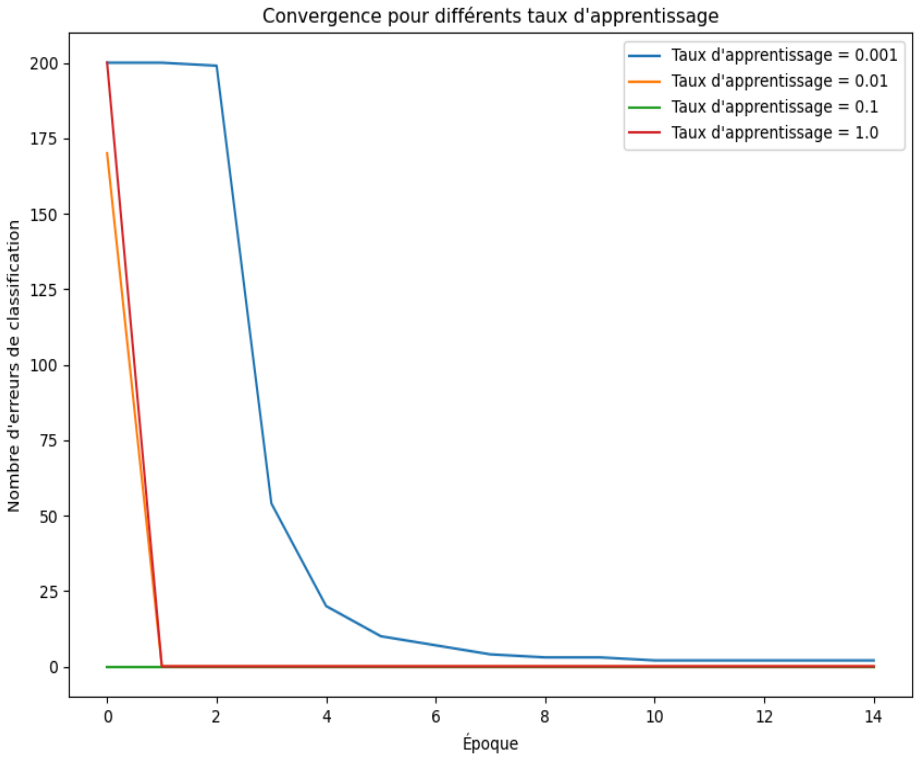
Q2) S'il est trop petit ? S'il est trop petit il apprendra trop lentement.

Q3) Y a-t-il une valeur idéale ? Non, Il n'y a pas vraiment de valeurs idéale car cela va dépendre de l'évolution de l'apprentissage du modèle.

Q4) Peut-on faire varier le taux ? Oui on peut faire varier n, c'est même incontournable dans la grande majorité des cas.

Q5) Une stratégie ? Plus le gradient diminue plus on ralentit la vitesse d'apprentissage (on diminue n) pour affiner le modèle et plus le gradient augmente plus on augmente n pour optimiser la vitesse d'apprentissage.





Réponses aux autres questions

Exercice 7

Q1) On constate que la droite de séparation est différente à chaque exécution, mais elle sépare toujours correctement les deux nuages de points. Cela confirme que l'initialisation des poids influence la solution finale trouvée.

Exercice 8

Q1) On observe que la courbe d'erreur diminue très lentement et de manière très lisse. Il faut beaucoup d'époques pour atteindre zéro erreur.

Q2) L'erreur fluctue énormément d'une époque à l'autre. La courbe de convergence est très bruitée et le modèle peut même diverger (l'erreur augmente).

Q3) Oui, par expérimentation, on peut trouver une plage de valeurs optimales. Par exemple, une valeur comme 0.01 ou 0.1 semble souvent offrir un bon compromis entre vitesse de convergence et stabilité.

Q4) Si les données sont très bruitées ou si les classes se chevauchent, un taux d'apprentissage plus petit est souvent nécessaire pour éviter que le modèle ne soit trop influencé par les points mal classés et pour trouver une frontière plus stable.

Exercice 9

Q1) On choisit la classe correspondant au perceptron qui a donné le score le plus élevé (la valeur de la somme pondérée w*x+b avant l'activation). Ce score est utilisé comme une mesure de confiance.

Q2) La règle reste la même : on choisit la classe avec le score le plus élevé, même si tous les scores sont négatifs. Le "moins négatif" est considéré comme le meilleur choix.

Q3) L'approche de base ne le gère pas bien, c'est l'un de ses inconvénients. Chaque classifieur binaire est entraîné sur un jeu de données déséquilibré, ce qui peut le rendre moins performant sur la classe minoritaire.

Évaluation sur données réelles

Pour tester le perceptron multi classe, j'ai utilisé les données Iris (avec 3 classes de fleurs).

> Si plusieurs perceptrons disent "oui" ? On choisit le score de sortie le plus élevé, on peut considérer que c'est notre niveau de confiance (en quelque sorte).
>
> Si aucun ne dit "oui" ? On fait la même chose: on garde le score le plus élevé de tous.
>
> Le problème avec cette méthode : Elle ne le gère pas [le déséquilibre des classes] et c'est justement son principal défaut. En confrontant une seule classe à beaucoup d'autres, le perceptron devient mécaniquement moins performant sur la classe qui est seule.

Discussion

Limites du perceptron

Ce TP a bien montré les limites du perceptron :

1. Il ne peut pas tout séparer : Sa plus grosse limite, c'est qu'il ne marche que si on peut séparer les données avec une seule ligne droite. Le problème du XOR le montre bien.
2. Il n'aime pas le bruit : Le bruit peut empêcher les données d'être séparées correctement car le perceptron va essayer de classer les points bruités et n'y arrivera pas.
3. Il est mauvais avec des classes déséquilibrées : Le modèle sera très mauvais pour prédire les classes minoritaires comparé aux classes majoritaires ou il sera très bon.

Cas d'usage appropriés

Même avec ses limites, le perceptron peut être utile.

> Il est bien pour les problèmes simples où on sait que les données sont séparables par une seule droite (linéairement).
>
> Il est rapide et facile à coder, donc c'est un bon premier test.
>
> Pour qu'il marche mieux, il faut normaliser les données avant l'entraînement car ça aide l'algorithme à apprendre plus vite.

Conclusion : Synthèse des apprentissages

En conclusion, ce TP m'a bien aidé à comprendre comment le Perceptron fonctionne. J'ai bien vu que sa grosse limite, c'est qu'il a besoin que les données soient séparables par une seule ligne droite. C'est pour ça qu'il a échoué sur le XOR.

Ça montre pourquoi on a inventé des réseaux plus compliqués après. J'ai aussi compris que des trucs comme le taux d'apprentissage sont super importants à bien régler.

Le fait de coder soit même les algorithmes aide (en tout cas personnellement) à mieux se représenter leur force / faiblesse et surtout leur limite.