Rapport de Projet:
Projet Annuel 3 Big Data

RHERMINI Idriss

31/08/2020

Table des matières

Objectifs

Lors de ce rapport, je dois réaliser plusieurs objectifs. je dois implémenter les algorithmes sur le classification/régression linéaire ainsi que le perceptron multi couche pour la classification et la régression. Tous les algorithmes doivent valider un certain nombre de test. De tests simples comme le *OR* à plus complexe comme le *XOR* et le test de la croix à 3 classes.

Algorithme Linéaire

Classification Linéaire

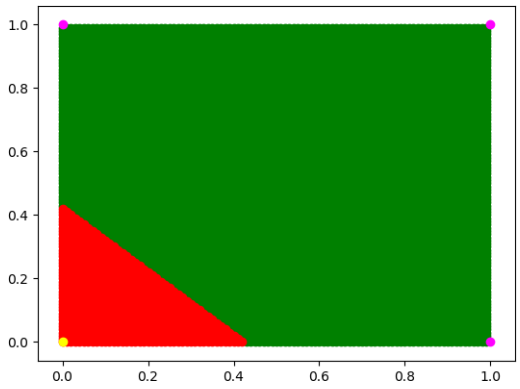
J'avais une structure trop complexe pour gérer la classification linéaire. Les résultats n'étaient pas mauvais, mais je voulais le refaire pour bien comprendre son implémentation.

Le principe reste inchangé, j'ai toujours un seul neurone avec n entrées auquel on rajoute un neurone de biais.

Pour chaque époque, je charge les entrées une a une. Je calcule la sortie du neurone, puis rectifie le poids de chaque entrée grâce à l'implémentation de Rosenblatt. La logique de cette implémentation est basée sur la distance entre la valeur calculée et la valeur attendue. Cette distance donne donc la direction et l'amplitude de la variation de la mise à jour des poids. Au début j'avais en chargement des inputs de façon linéaire, ce qui n'avait pas de gros impact sur les petits dataset, comme les tests case. Mais qui pouvait être un biais pour les datatset bien plus gros. J'ai donc intégré un système pour choisir aléatoirement une image. Ce n'est toujours pas parfait car l'ordre reste le même selon les Epochs. En effet j'ai stocké les index des inputs dans un vecteur que j'ai mélangé.

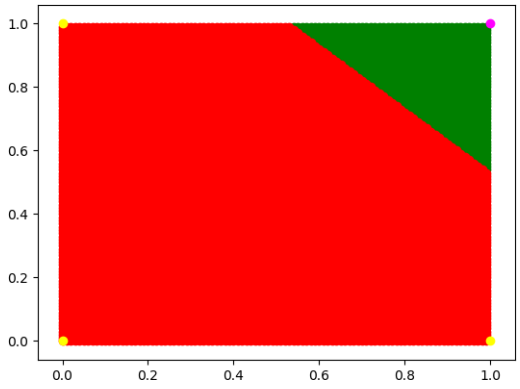
OR

Le cas de test du *OR* est le cas de test le plus simple. je dois séparer linéairement deux classes de points.



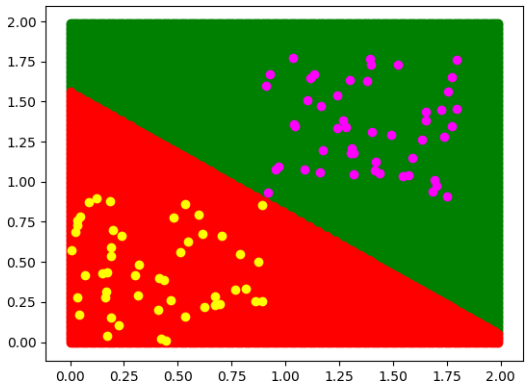
AND

Le cas de test du *AND* est identique au *OR*, cela me permets de confirmer le bon fonctionnement de mon neurone.



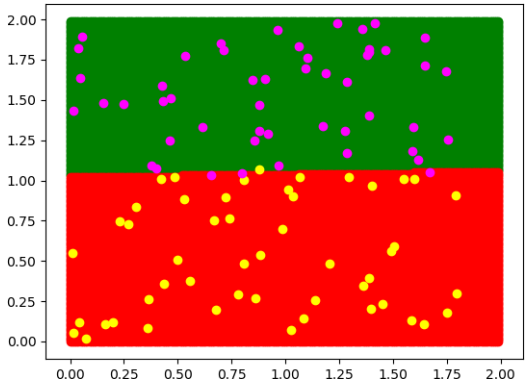
Cas simple

Ce cas de test est un test avec plus de valeurs. Le principe est le même que le *OR* et le *AND*, il permet de vérifier que le neurone marche bien avec des valeurs aléatoires.



Je dois donc créé une règle qui génère 100 points, la moitié appartient à une première classe et le reste à une autre classe. Lors de la génération des points, on s'assure qu'ils ne se chevauchent pas pour être sûr d'avoir une séparation linéaire.

Dans cet exemple j'ai rajouté du bruit à la génération des points pour observer la réaction de l'apprentissage.



On constate que quelques points sont du mauvais côté de la séparation. En effet le neurone montre sa limite, il est incapable d'aller chercher ses points, tout simplement car le jeu de données n'est pas séparable linéairement !

Cas simple à 3 classes

L'objectif de ce cas de test est de vérifier que je suis capables de séparer linéairement plusieurs classes. Or un neurone n'a qu'une seule sortie, il faut donc entrainer 3 neurones séparés afin produire 3 fonctions affines.

Si j'ai 3 classes, A, B, C. Alors il suffit d'entrainer un premier neurone à séparer la classe A des classes B et C regroupées. On répète ce processus

2 fois pour les classes B et C, puis on obtient 3 classifications linéaire. Il suffit maintenant de faire une prédiction sur les 3 neurones pour récupérer la bonne classe qui lui est associée.

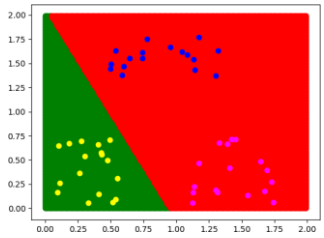


FIGURE 1 – Fit A

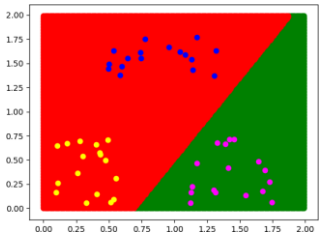


FIGURE 2 – Fit B

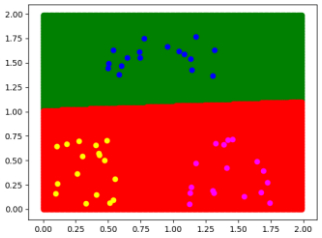
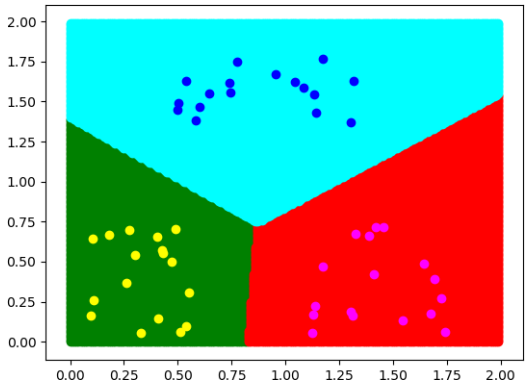


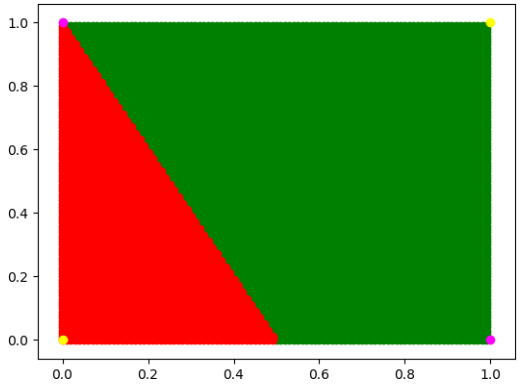
FIGURE 3 – Fit C

Une fois chaque classe classifiée individuellement, il suffit de prendre la prédiction la plus haute pour chacune des classes.



Cas impossible : XOR

Je suis dans une classification linéaire, il est impossible pour un neurone seul de réaliser une séparation linéaire du cas du XOR. En effet il nous faut deux droites pour pouvoir séparer les classes. On observe bien ici l'infaisabilité du problème fourni au neurone. Le neurone ne sait que générer une fonction affine et non plusieurs.



On observe encore une fois la limite du neurone seul. Il est impossible de séparer les points $[1, 0]$ $[0, 1]$ dans une première classe et $[1, 1]$ $[1, 1]$ dans une seconde par une droite unique.

Régression Linéaire

La régression linéaire a pour but d'estimer de trouver une valeur plutôt que trouver l'appartenance, ou non, d'une classe. Pour cela j'ai suivi l'implémentation matricielle de la régression linéaire.

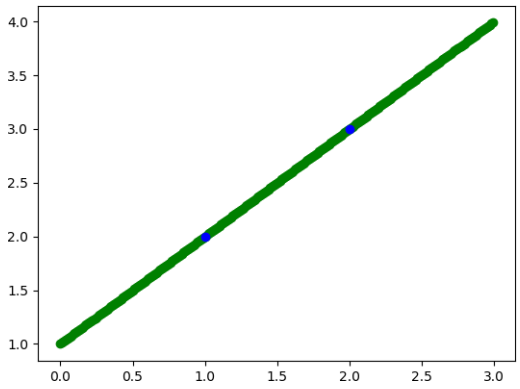
On trouve les poids de chaque entrée grâce à cette formule :

$$W = ((X^T X)^{-1} X^T) Y$$

Pour les calculs matriciels, j'ai utilisé la bibliothèque *Eigen*.

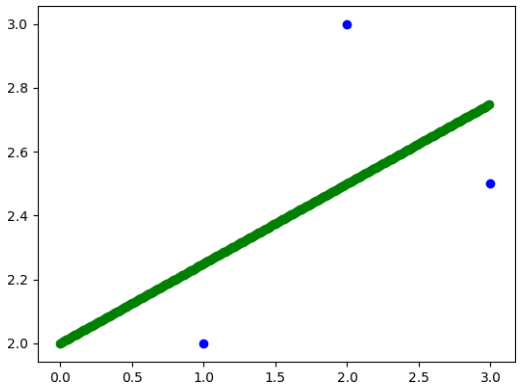
Simple 2D test

Le premier cas de test est un apprentissage sur 2 points. On est donc sûr d'avoir une unique droite qui passe par ses deux points en résultat.



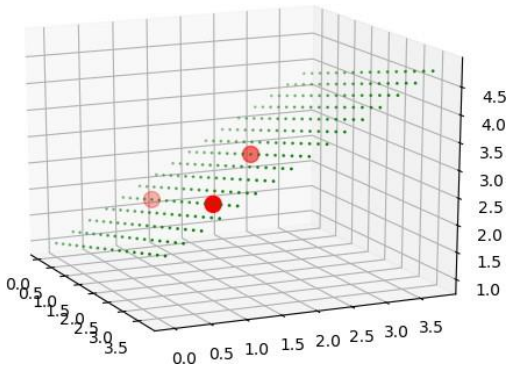
En effet j'ai bien une droite, de plus cet exemple me permet de voir le bon fonctionnement du biais, qui me permet d'avoir une fonction affine ne passant pas uniquement par l'origine.

Le cas suivant est le cas général de la régression linéaire.



Simple 3D test

Le cas 3D me permet de savoir si ma régression linéaire est capable de traiter plusieurs entrées pour un même individu.



On se retrouve donc avec un plan dans l'espace créé par 3 dimensions.

Réseau de Neurones

Classification

Cas de tests basiques

Voici les résultats du AND, OR et XOR avec un réseau de neurones simpliste de 2 neurones en couche d'entrée et 1 neurone de sortie.

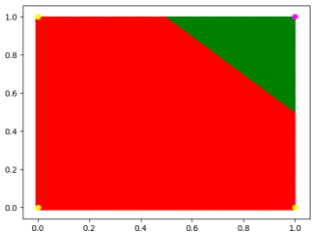


FIGURE 4 – MLP AND

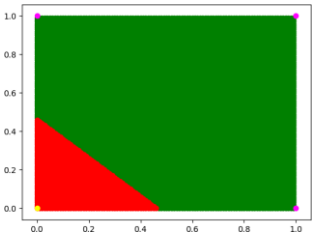


FIGURE 5 – Fit C

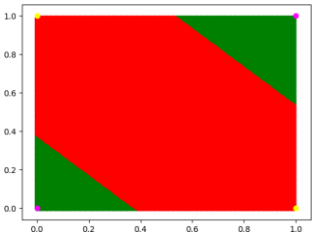


FIGURE 6 – XOR 1

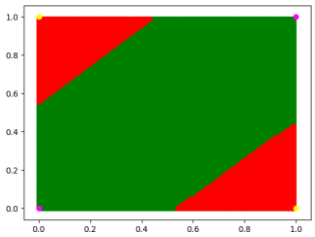
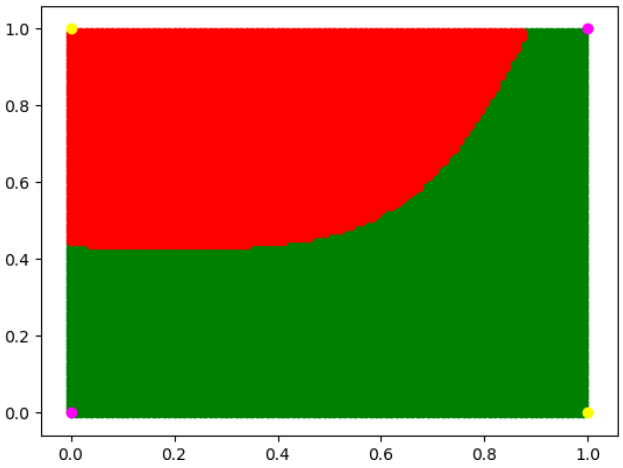


FIGURE 7 – XOR 2

D'après les résultats ci-dessus, tout semble correct, or si on fait plusieurs XOR avec les mêmes hyperparamètres, on peut obtenir des résultats différents.



On suppose que dans le premier cas mon Modèle soit en sous apprentissage, même si on émet des doutes, car les nombres d'epochs et le pas d'apprentissage reste inchangé pour tous les cas du *XOR*. Il se peut aussi que cela soit normal, dû à la génération aléatoire des poids. En effet je suis censé trouver deux droites affines parallèle, mais il se peut que mes poids initiaux créés une tendance qui ne favorise pas la distinction des deux droites affines. On a donc une combinaison de deux droites qui essayent de garder lesdeux extrémité

Haute-Droite et Basse-Gauche dans le vert, et le reste dans le rouge. D'où le coude formé en rouge qui tente de se rapprocher du point Bas-Droite. On note aussi que lors de ses résultats, on obtient des valeurs de *Loss* très hautes à la normale et sont presque impossible à corriger. On suppose que je suis dans un minimum local et que je n'ai pas encore les outils pour pouvoir sortir de ses cas.

Cross

Le cas du *Cross* à le même comportement que le *XOR*, il marche la majorité du temps, mais de façon aléatoire on se retrouve avec un modèle qui ne marche pas.

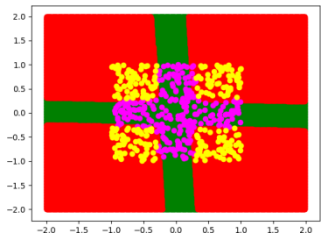


FIGURE 8 – Cross

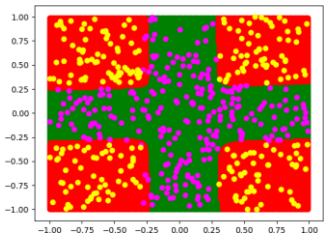


FIGURE 9 – CROSS

On suppose que ce sont des comportements normaux qui dépendent de la position initiale (les poids) de la recherche de l'optimisation. Il suffirait de jouer avec les hyperparamètres pour sortir de ses cas particuliers pour obtenir le bon résultat. C'est comme si j'étais dans un minimum local, et que la distance entre ma prédiction et mes résultats ont une influence trop restrictive sur l'orientation de la recherche.

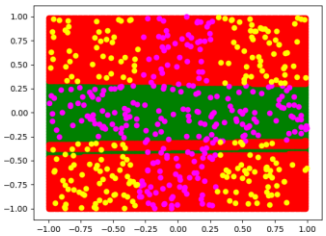


FIGURE 10 – Modèle du Cross

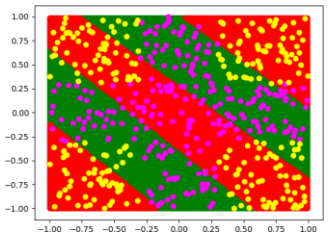
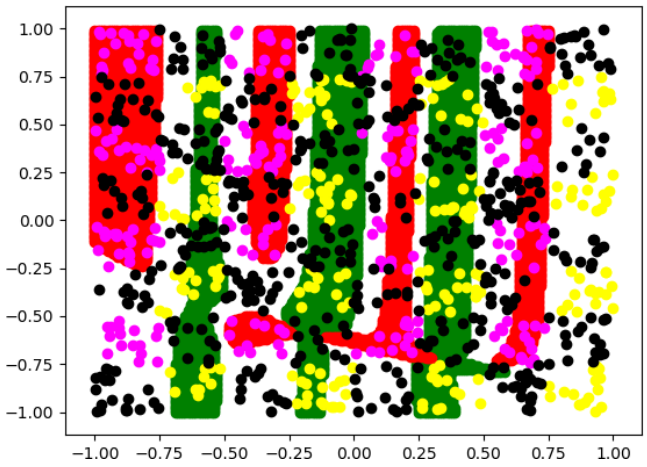


FIGURE 11 – CROSS

En effet on observe bien que je possède les deux features du réseau de neurones [2, 1], pour résoudre ce problème, il me faudrait un outil de tempéra- ture capable d'autoriser de s'éloigner de mon objectif pour mieux s'y rediriger, ou alors relancer l'apprentissage du Modèle jusqu'à trouver une Loss correcte.

Multi Cross

Le cas du multi cross est complexe, à l'heure actuelle mon `Loss` n'est pas encore capable de donne un résultat qui reflète un bon apprentissage selon plusieurs classes. En effet je ne faisais le calcul que d'un neurone de sortie et non de tous les neurones de sortie de la dernière couche.



Dans ce cas, on ne sait pas si les prédictions en rouge appartiennent à la classe des points noirs ou des points magentas. De même pour les prédictions en vert pour les points en noirs ou en jaune. Finalement grâce à la dernière prédiction en blanc, on peut supposer que mon réseau de neurones essayait d'assigner les prédictions en rouge pour les points en magentas, le vert aux points jaunes et le blanc aux points noirs. Ici aussi nous pensons que je me retrouve dans des cas de minimum locaux qui me interdisent de faire des erreurs pour franchir une étape. Il me manque la possibilité de rater pour mieux réussir.

Voici d'autres tentatives avec des réseaux de neurones plus profonds en largeur et/ou en hauteur.

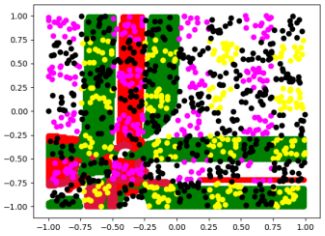


FIGURE 12 – Modèle : [8, 8, 3]

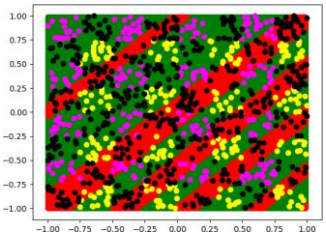


FIGURE 13 – Modèle : [16, 16, 3]

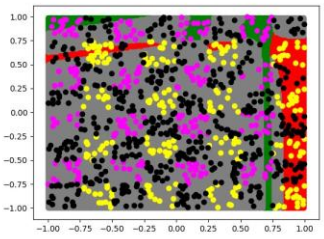


FIGURE 14 – Modèle : [24, 24, 24, 24, 3]

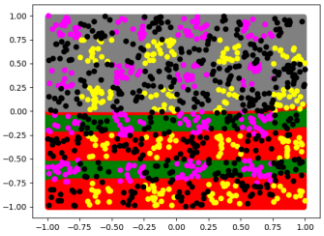
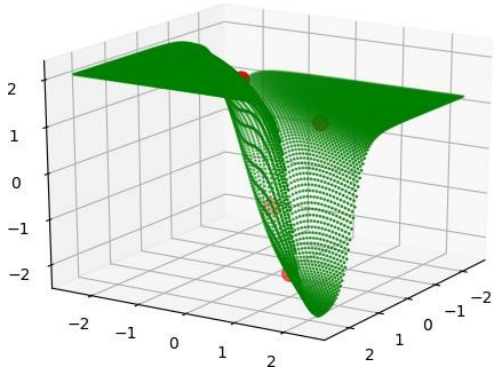


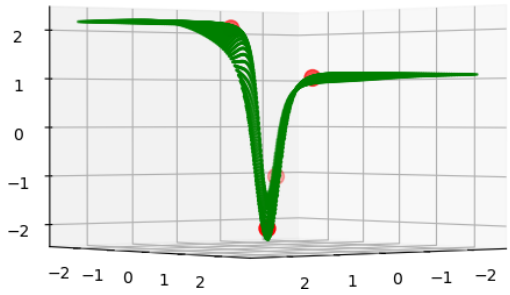
FIGURE 15 – Modèle : [8, 8, 8, 8, 3]

Régression

L'implémentation de la régression a été facile, une fois la classification réussie, je n'ai qu'à faire de légère modification concernant le calcul du delta de la dernière couche, mais aussi du calcul de la sortie du réseau de neurones qui n'est rien de plus qu'une somme pondérée. Dans la classification c'est une somme pondérée injecté dans une fonction Tangente hyperbolique.

Voici le résultat d'un cas simple d'une régression 3D qui n'est réalisable qu'avec un réseau de neurones. En effet, un model linéaire ne serait pas adapté pour résoudre ce problème.





Conclusion

Je prédis mes classes grâce à mon algorithme de réseau de neurones. Cependant j'ai vu que je n'étais pas capables de réaliser en permanence de bon résultat. Je pense que cela est dû à une correction trop sévère des poids lorsqu'on ne s'écarte rien que d'un petit peu de mon objectif. En effet, je corrige les poids en fonction de la distance de mes prédictions et de mes points attendus, or il est impossible de se rapproche de ce point si j'ai un obstacle entre. J'ai donc coincés dans ce que l'on pense être un minimum local.