Bach Antoine Lafond Théo

Rapport : Réseau de neurones — Application à la classification

Introduction **1**

I - Classification binaire **1**

a - Mono-couche 1

b - Deux couches 2

c - n-couches 3

II - Classification multi classe **4**

a - Mono-couche 4

b - Multi-couches 4

III - Analyse des performances **5**

a - Adaptatif 5

b - Comparaison des convergences 5

c - Robustesse 6

Conclusion **6**

Introduction

Ce projet a pour but de coder un simple réseau de neurones et sa descente de gradient afin de, dans un premier temps, faire une classification binaire de couples et dans un second temps, faire de la classification d'images de chiffres. Les études se feront avec un réseau mono couche et un réseau à deux couches afin de comparer les performances.

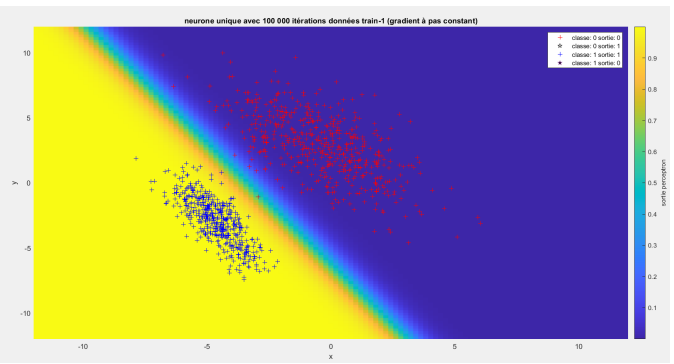
I - Classification binaire

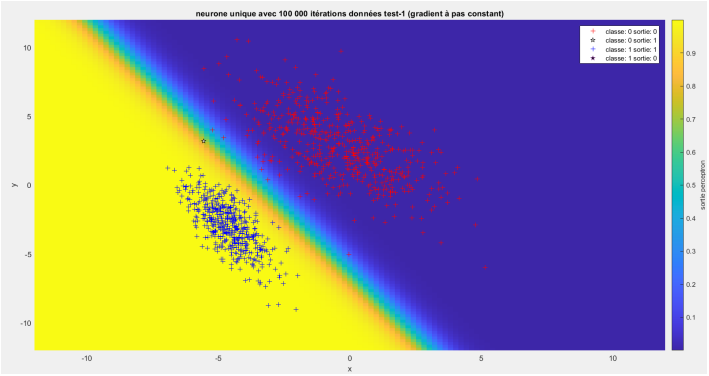
Nous avons à disposition deux jeux de données que nous appellerons problème 1 et problème 2. Chacun est composé de données d'entraînement et de données de test. Elles sont composées de deux données d'entrée répartis dans deux classes : 0 ou 1. Nous allons créer des réseaux de neurones entraînés sur les données d'entraînement que nous allons tester sur les données de test.

a - Mono-couche

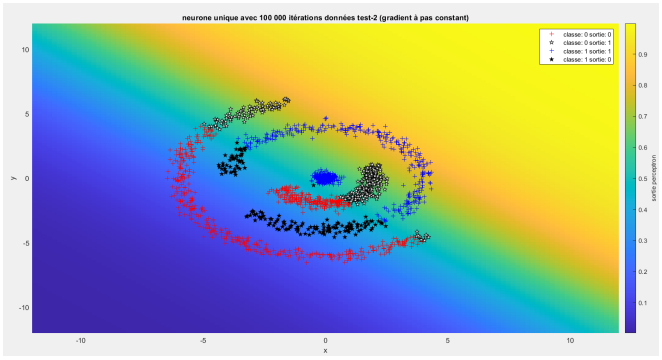
Tout d'abord nous créons un simple neurone à deux entrées que nous entrainons sur les données d'entrainement du problème 1. Nous entraînons notre neurone avec la méthode de la descente de gradient à pas fixe.

Avec 100 000 itérations et un pas de 1, nous arrivons à avoir 100% de bonne classification par notre perceptron. (Dans la suite, nous appellerons ce pourcentage la réussite). Sur les données de test nous obtenons 99,9% de réussite.





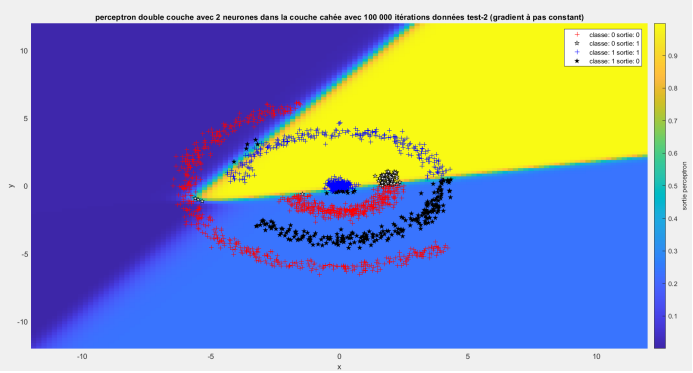
Nous avons une très bonne réussite avec le simple neurone pour le problème 1, voyons ce qu'il en est pour le problème 2.
Avec les mêmes paramètres, nous n'obtenons que 75,6% de réussite, et encore moins sur les données de test pour le problème 2. Ce résultat est très mauvais.



Sur le graph ci-dessus représentant les points du jeu de données d'entraînement et leur classification ainsi que la sortie du neurone en dégradé de couleur, nous remarquons qu'avec un seul neurons, la séparation des données par le neurone est linéaire : il ne peut séparer les classes que par une droite. Cela marchait pour le problème 1 car il était possible de séparer les classes par une droite mais avec des classes plus complexes comme dans le problème 2, c'est impossible. Il nous faut donc plus de neurones.

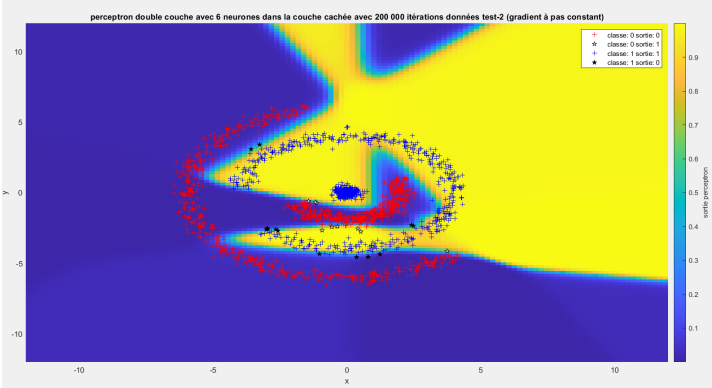
b - Deux couches

Nous entraînons un perceptron à 2 couches avec 2 neurones en couche 1 avec les données d'entraînement du problème 2. Le taux de réussite obtenu est de 84,2%, le perceptron a-t-il cette fois compris le problème 2 ? En observant le graph des points, ci-dessous nous voyons que malgré la réussite meilleure, le perceptron n'a pas du tout compris le problème.

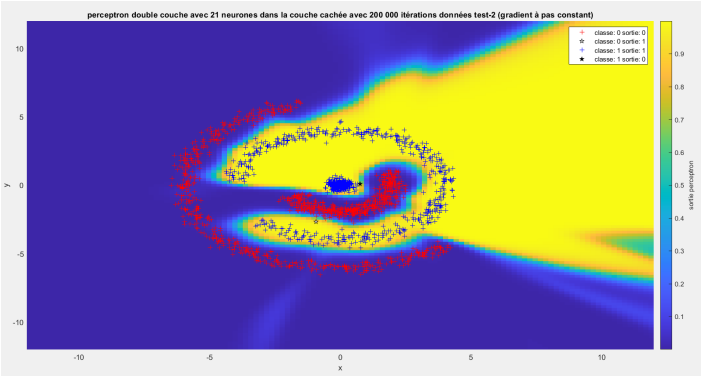


Nous faisons donc des test avec de plus en plus de neurones en couche 1 et remarquons que le nombre de neurones est égale au nombre de droites de segmentation dans le graph. Donc plus il y a de neurone, plus l'entraînement pourra créer un contours précis pour délimiter les classes. Mais l'augmentation du nombre de neurones augmente aussi le temps de calcul et l'énergie utilisée pour le

faire. Il faut trouver le juste milieu entre réussite du résultat et énergie pour faire le calcul. Par exemple on peut utiliser 6 neurones qui est le minimum pour que le perceptron ait une réussite correcte (99,2% sur données de test) et que les droites segmentent à peu près la classe comme on peut le voir ci-dessous :



Si on veut un résultat parfait, on peut avoir 100% de réussite sur les données d'entraînement et 99,9% sur les données de test avec 21 neurones.



Nous avons observé l'effet d'avoir plusieurs neurones sur une couche cachée, voyons maintenant les conséquences d'avoir n couches.

c - n-couches

Pour le mono couche et le 2 couches, nous calculons séparément le gradient pour les deux couches avec deux formules différentes. Mais cette méthode n'est pas applicable pour un perceptron à un nombre indéterminé de couches. C'est pourquoi il faut trouver une manière pseudo récursive de calculer de gradient.

$$\frac{\partial f}{\partial w_{i,j}^{(l)}} = \frac{1}{2N}\sum_{n=1}^{N}\sum_{l_L=1}^{I_L}\cdots\sum_{l_{l+1}=1}^{I_{l+1}}\frac{\partial f_n}{\partial y_{l_L}^{(L)}}\frac{\partial y_{l_L}^{(L)}}{\partial y_{l_{L-1}}^{(L-1)}}\frac{\partial y_{l_{L-1}}^{(L-1)}}{\partial y_{l_{L-2}}^{(L-2)}}\cdots\frac{\partial y_{l_{l-1}}^{(l+1)}}{\partial y_{l}^{(l)}}\frac{\partial y_{l}^{(l)}}{\partial w_{i,j}^{(l)}}$$

$$= \frac{1}{2N}\sum_{n=1}^{N}\sum_{l_L=1}^{I_L}\cdots\sum_{l_{l+1}=1}^{I_{l+1}}\left(y_{l_L}^{(L)} - c(x_n)\right)\left(y_{l_L}^{(L)} - y_{l_L}^{(L)^2}\right)w_{l_L}^{(L)}\left(y_{l_{L-1}}^{(L-1)} - y_{l_{L-1}}^{(L-1)^2}\right)w_{l_{L-1}}^{(L-1)}\cdots\left(y_{l_{l+1}}^{(l+1)} - y_{l_{l+1}}^{(l+1)^2}\right)w_{l_{l+1}}^{(l+1)}\left(y_{l_l}^{(l)} - y_{l_l}^{(l)^2}\right)x_n$$

$$= \frac{1}{2N}\sum_{n=1}^{N}\sum_{l_L=1}^{I_L}\cdots\sum_{l_{l+1}=1}^{I_{l+1}} m^{(l+1)}\left(y_{l_l}^{(l)} - y_{l_l}^{(l)^2}\right)x_n \quad \text{où} \quad m^{(k)} = \left\{m^{(k+1)}\left(y_{l_k}^{(k)} - y_{l_k}^{(k)^2}\right)w_{l_k}^{(k)}\ si\ k \le L;\ \left(y_{l_k}^{(k)} - c(x_n)\right)\ sinon\right\}$$

Nous implémentons une méthode de calcul des gradients de chaque couche de manière pseudo récursive, c'est-à-dire avec des multiplications successives par m, en calculant entre chaque itération l de la multiplication le gradient de la couche $L - l$. Cette implémentation se fait avec la création de la classe perceptron_n qui permet de créer des objets de n'importe quel nombre de couches et n'importe quel nombre de neurones à chaque couche, de les entraîner, d'afficher leur graph s'ils sont de classification binaire à 2 entrées et enfin d'afficher leur matrice de confusion.

Nous testons sur des perceptrons 3 couches pour les comparer à des 2 couches (Plus que 3 couches est trop long à entraîner) sur le problème 2. Nous remarquons qu' un perception avec n_1 neurone en couche 1 et n_2 en couche 2 a le même comportement qu'un perceptron 2 couches avec $n_1 + n_2$ neurones en couche 1. C'est une conjecture que nous faisons sur quelques tests, nous aurions voulu en faire plus mais le temps d'entraînement pour chaque test à $n > 2$ couches est trop long.

II - Classification multi classe

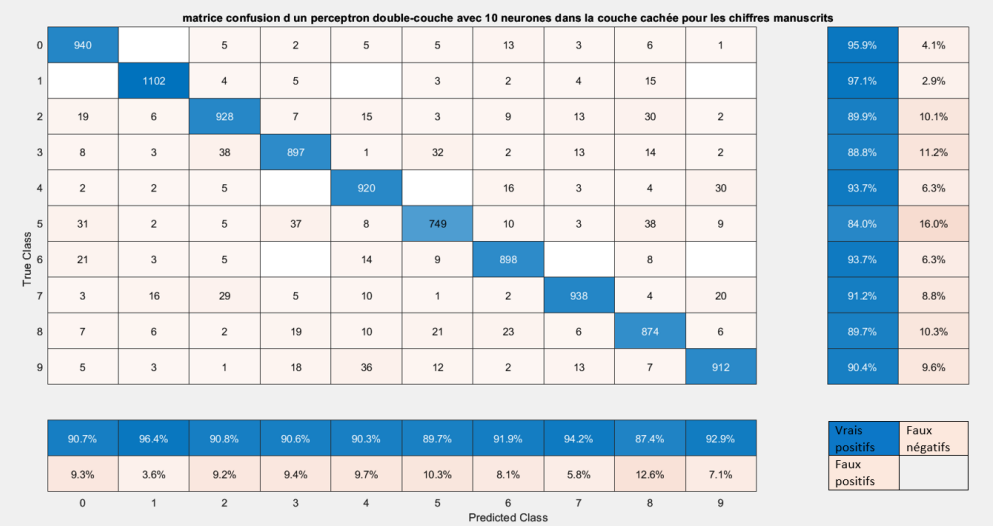
Maintenant que l'on a vu que l'on pouvait séparer deux classes à l'aide de nos réseaux de neurones, on va traiter les problèmes non binaires, id est multi-classe. Ici, on va essayer de différencier des images de chiffres manuscrits à l'aide d'un perceptron monocouche et d'un perceptron multi-couche.

a - Mono-couche

Pour le perceptron mono-couche, nous avons entraîné notre perceptron avec 20 000 itérations, on obtient 78,24% sur les données d'entraînement et 77,59% sur les données tests. Pour voir plus en détail, les résultats de notre perceptron nous avons tracé la matrice de confusion qui nous permet de déterminer les vrais positifs, les faux positifs et les faux négatifs par rapport aux résultats attendus (voir l'image en dessous). On peut observer que les résultats sont très satisfaisants pour tous les chiffres sauf le 8. En effet, le 8 est confondu avec le 5, le 3 et le 2 (respectivement 35,8%, 16.1% et 14.7%). Avec le mono-couche, on a testé plusieurs fois avec des poids différents pour observer la matrice de confusion et nous avons observé que l'on a toujours un seul chiffre quasiment pas reconnu (ce sont généralement le 0, 5 et le 8). Nous n'avons pas réussi à déterminer l'origine de ce phénomène.

b - Multi-couches

Pour le perceptron multi-couche, nous avons pris un perceptron à deux couches avec 10 neurones dans la couche cachée. Après avoir entraîné notre perceptron avec 20 000 itérations, on obtient 89,41% sur les données d'entraînement et 88,78% sur les données tests. Pour voir plus en détail, les résultats de notre perceptron nous traçons la matrice de confusion qui nous permet de déterminer les vrais positifs, les faux positifs et les faux négatifs par rapport aux résultats attendus (voir l'image en dessous). On peut observer que les résultats sont très satisfaisants comme on avait pu le voir avec le pourcentage de réussite. On peut remarquer que la classe la moins bien reconnue est le 5 qui a beaucoup plus de faux négatifs que les autres chiffres manuscrits. Le 5 est confondu avec le 8, le 3 et le 0 (respectivements 4,26%, 4.15% et 3,48%)



III - Analyse des performances

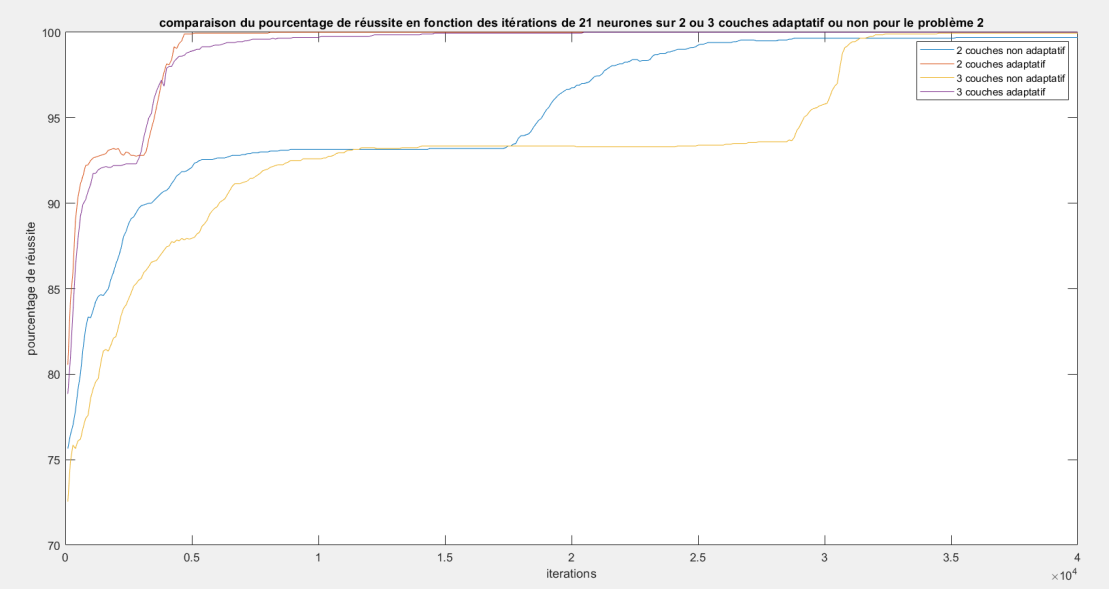
a - Adaptatif

On peut optimiser notre calcul de gradient en mettant en place un gradient à pas variable. En effet, plus notre fonction de coût diminue, plus le gradient est faible et donc plus l'évolution est lente. Pour contrecarrer cette diminution on augmente le pas c'est à dire rho_adaptatif: c'est le principe du pas variable.

b - Comparaison des convergences

Nous avons voulu comparer le nombre d'itérations nécessaire à entraîner un réseau de neurones sans et avec pas adaptatif. Pour cela nous avons implémenté le pas adaptatif et avons tracé le pourcentage de réussite en fonction des itérations sur le problème 2 pour un neurone 2 couches avec 21 neurones en couche 1. Nous avons choisi cette configuration car nous savons qu'il pouvait atteindre 100% de réussite. Résultat : -83% d'itérations entre non adaptatif et adaptatif pour atteindre 100%.

Précédemment, nous avons fait la conjecture qu'un perceptron à n couches et k neurones en tout avait les mêmes résultats qu'un perceptron deux couches avec k neurones en couche 1. Nous voulons vérifier si un perceptron à n couches k neurones en tout a un entraînement qui converge plus vite qu'un neurone 2 couches à k neurones en couche 1. Pour voir s'il y a un intérêt à faire plus de couches. Et résultat : non, il n'y a pas de différence significative, ni en adaptatif ni en non adaptatif.



c - Robustesse

Le choix des poids de départ est très important. En effet, lors de nos tests, nous avions de prime abord fixé nos poids à 1. Cela marchait pour les classes binaires mais lorsque nous sommes passés aux classes multiples nos résultats ne dépassent pas 50%. Ce qui se passe c'est que le calcul de la sigmoïde prend en entrée:

$$z = (\sum_{k=1}^{N} xk * w_k) + b = \sum_{k=1}^{N} xk$$

Or avec 400 entrées comprises entre 0 et 1, $z >> 1$ et par conséquent, la sigmoïde nous renvoyait toujours 1. Pour remédier à ce problème, nous devons mettre soit mettre tous les poids à 0 au départ (option 1) soit distribués autour de 0 (option 2)pour avoir

$$\frac{1}{N}\sum_{k=1}^{N} w_k \simeq 0$$

Néanmoins, si l'option 1 est choisie , elle a pour conséquence d'imposer une égalité sur nos poids dans les neurones de même couches et donc d'imposer une symétrie des neurones d'une même couche ce que nous ne voulons pas. C'est pour cette raison que nous adoptons l'option 2.

Conclusion

Nous avons pu observer lors de ce projet que le dimensionnement d'un réseau de neurones est important. Pour de la classification binaire un simple neurone peut suffire si les données peuvent se séparer linéairement mais si la classification est plus complexe, une deuxième couche s'impose. Nous avons remarqué qu'ajouter plus de deux couches ne change pas si le nombre total de neurones reste le même. Pour la classification multi classe pour la lecture de chiffres manuscrits, avec une couche, le résultat est sommaire mais pas impertinent. En ajoutant une première couche de dix neurones nous obtenons un bon résultat. L'entraînement des neurones est très long, nous avons donc utilisé la méthode du pas adaptatif qui nous a fait gagner du temps et de l'énergie.