IBI
Réseaux de neurones

AUBRET Arthur et BETTINELLI Mickael

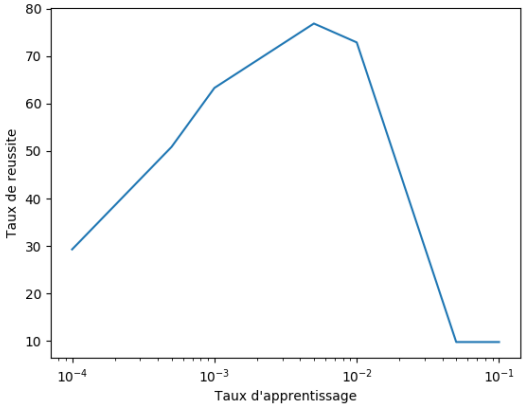
Université Lyon 1
M2 - 2017/2018

1 Perceptron

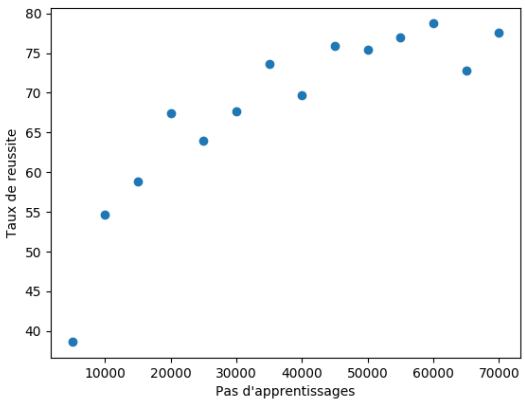
Nous allons d'abord étudier les résultats du Perceptron. Les paramètres par défaut que nous utilisons sont :

η : 0.005

Pas d'apprentissages : 70000



(a) Evolution du taux de réussite en fonction du pas d'apprentissage



(b) Evolution du taux de réussite en fonction du nombre de données apprises

Les poids initiaux sont déterminants pour la déterminer le temps de convergence de l'algorithme, si les poids initiaux sont proches des poids idéaux, l'algorithme partira d'un taux d'erreur bas et aura plus rapidement un taux d'erreur bas : Le temps de convergence change. Cependant ils n'influenceront pas, dans le cas du perceptron, la vitesse de convergence. Dans notre cas, nous générons des poids aléatoires.

Sur la figure 1a, nous affichons l'évolution du taux de bonnes prédictions en fonction du pas d'apprentissage. Nous voyons que celui-ci augmente, jusqu'à atteindre un sommet, puis redescend. Le taux de réussite augmente et se réduit de manière exponentielle. Pendant la phase ascendante, le pas d'apprentissage est trop bas et n'a pas le temps de converger, nous aurions besoin de plus de pas de temps. Pendant la phase descendante, le taux d'apprentissage est trop haut, le réseau n'arrive donc pas à converger et oscille autour de la solution optimale.

Sur la figure 1b, nous affichons l'évolution du taux de bonnes prédictions en fonction du nombre de pas d'apprentissage. Nous voyons que l'amélioration est de type logarithmique, plus le réseau est efficace, moins il s'améliore rapidement. On perçoit cependant qu'il converge vers 80% de réussite, ce qui semble être la limite induite par sa linéarité.

Le maximum de réussite que nous avons atteint est de 80%

2 Shallow network

Nous allons ici étudier les résultats du shallow network, c'est à dire d'un perceptron avec une couche cachée. Les paramètres par défaut que nous utilisons sont :

η : 0.01

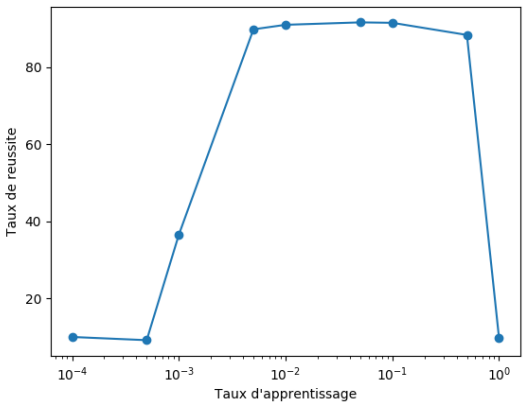
Pas d'apprentissages : 70000

Nombre de neurones dans la couche cachée : 16

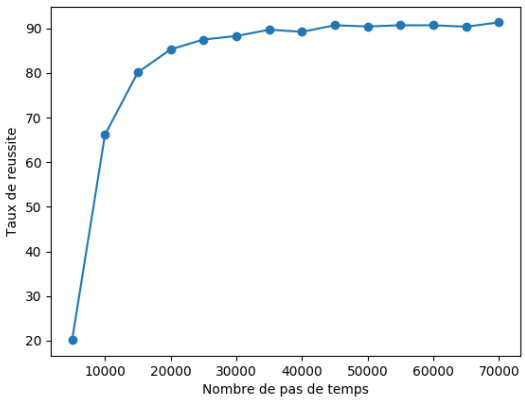
Nous obtenons environ 90% de réussite avec ces paramètres. Cependant nous avons déjà fait de meilleurs résultats avec d'autres paramètres.

Par rapport au perceptron, les poids réagissent différement. Les poids de la couche finale ont des caractéristiques similaires. Les poids de chaque couche ne peuvent pas être identiques, sinon les poids de la couche cachée seront toujours modifiés de manière identique et aucun de ses neurones ne se différenciera des autres. Il faut donc générer aléatoirement une des deux couches. Par ailleurs, de manière à éviter d'avoir des produits scalaires trop importants, mal gérés par la fonction sigmoïde (Il sera difficile de converger si tous les neurones renvoient 1 dès le début, il est difficile d'approximer le déplacement aux bornes infinies), les poids doivent être, soit être faibles et répartis uniformément autour de 0 (Poids négatifs et positifs), soit très faibles (On peut diviser par le taille de l'entrée).

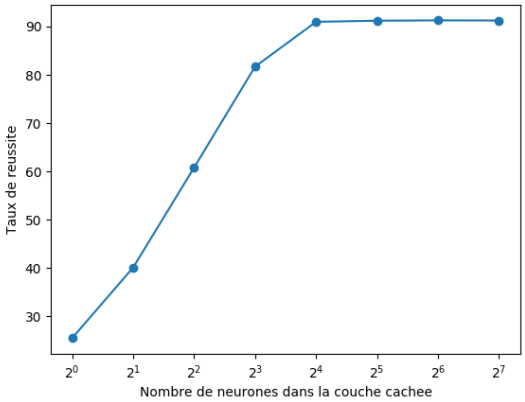
Sur la figure 2a, nous voyons que le pas influe énormément sur les résultats. Il augmente puis baisse très rapidement avec un pic au milieu. Le pic est plus large qu'avec le perceptron, la multiplication des couches le rend plus résilient aux oscillations autour d'une très bonne répartition des poids.



(a) Evolution du taux de réussite en fonction du pas d'apprentissage



(b) Evolution du taux de réussite en fonction du nombre de pas de temps



(c) Evolution du taux de réussite en fonction du nombre de neurones dans la couche cachée

Sur la figure 2b, nous voyons que le temps influe de la même manière que le perceptron : La taux évolue de manière logarithmique par rapport au nombre de données testées.

Nous voyons sur la figure 2c que le nombre de neurone est un paramètre primordial. Il semble y avoir une évolution quasiment linéaire jusqu'au plafond atteignable (Majoré par le pas d'apprentissage). Pendant la phase ascendante, la couche du milieu n'a pas suffisamment de neurones pour représenter abstraitement les zones importantes de l'image. Typiquement, un neurone pourrait représenter un trait commun entre le 3 et le 8. Une fois le maximum atteint, les neurones ajoutés apportent peu de choses. Si nous augmentons fortement le nombre de neurones (Plus que sur la figure), nous aurons tendance à apprendre par coeur les données de test et perdrons alors la faculté de généralisation.

3 Deep network

Nous essayons maintenant de rajouter des couches à notre réseau et modifions le calcul de l'erreur. Nous utilisons ici la fonction de sigmoïde de torch. Les paramètres par défaut que nous utilisons sont :

η : 0.25

Pas d'apprentissages : 100000

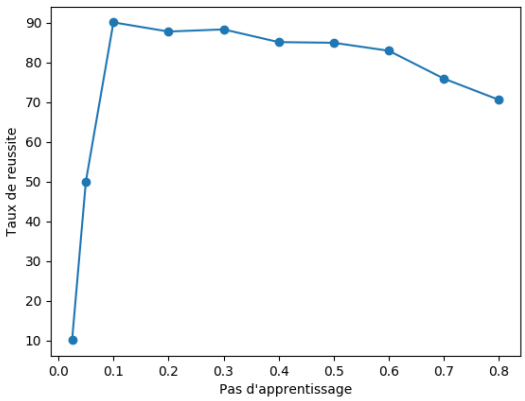
Nombre de couches : 2

Nombre de neurones par couche : 20

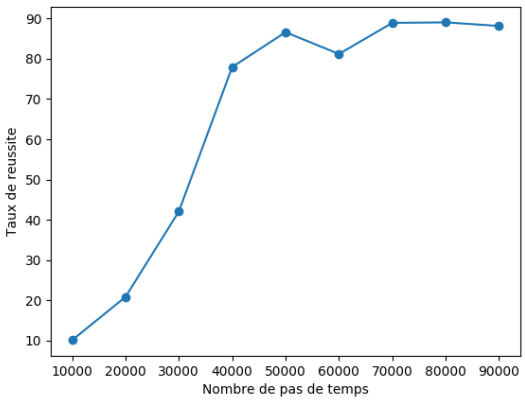
Nous obtenons 90% réussite avec ces paramètres, mais nous pouvons obtenir plus avec d'autres paramètres. Nous avons essayé de diversifier les paramètres sans avoir un programme trop long à executer.

Nous voyons sur les figures 3a, 3b, 3d que l'évolution du taux de réussite en fonction du pas d'apprentissage, du nombre de pas de temps et du nombre de neurones par couches est similaire aux réseaux précédents.

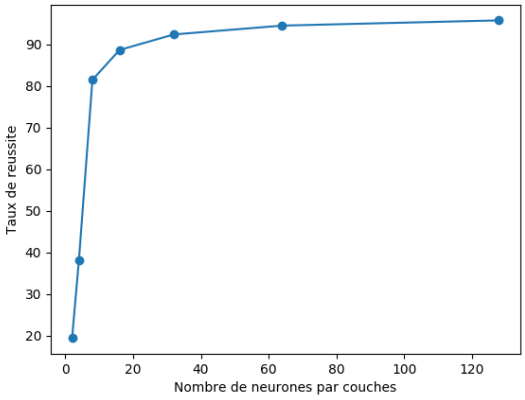
La principale nouveauté provient du nombre de couches, on voit que plus nous rajoutons des couches, moins le réseau apprend efficacement. Il faut en fait remarquer que la propagation du gradient fait que les poids



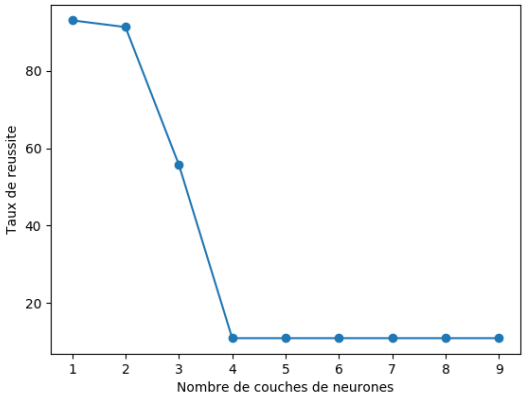
(a) Evolution du taux de réussite en fonction du pas d'apprentissage



(b) Evolution du taux de réussite en fonction du nombre de pas de temps



(c) Evolution du taux de réussite en fonction du nombre de neurones dans les couches cachées



(d) Evolution du taux de réussite en fonction du nombre de couches de 20 neurones

seront bas dans les premières couches. Il est ainsi un peu plus difficile de converger avec plusieurs couches. Par ailleurs, notre problème n'a pas besoin de plus de couches, du moins dans un réseau entièrement connecté car une deuxième couche d'abstraction est peu utile. D'après nos différents tests, le réseau qui contient plusieurs couches va seulement mettre plus longtemps à converger vers des résultats identique au réseau avec une seule couche.

4 Optionnal

4.1 Fonctions d'activation

Nous avons essayé trois fonctions d'activations :
— Sigmoïd
— ReLU
— Tanh

Les paramètres de la fonction Sigmoïd sont :

η : 0.1

Pas d'apprentissages : 50000

Nombre de couches : 1

Les paramètres de la fonction ReLU et Tanh sont :

η : 0.01

Pas d'apprentissages : 50000

Nombre de couches : 1

Le nombre de neurones par couche est variable afin de voir l'évolution du taux de réussite en fonction de la taille des couches.

Afin d'obtenir des taux de réussite satisfaisant, les paramètres des réseaux ne sont pas les même. En effet, la descente de gradient diffère par rapport aux fonctions d'activation. La fonction Tanh a par exemple un gradient plus fort que celui de la Sigmoïd. Il est donc impossible d'obtenir les même taux de réussite sans re-paramétrer le réseau. Puisque le résultat de la fonction d'activation est directement utilisé pour le calcul des poids, le pas d'apprentissage est le principal paramètre à adapter lors d'un changement de fonction d'activation.

Le pas d'apprentissage de la fonction ReLU a donc été recherché afin de la tester correctement. Comme le montre le graphique ci dessus, le taux de réussite est élevé sur un pas de 0.01. Bien que pas optimal, il nous assure de trouver un taux de réussite correct.

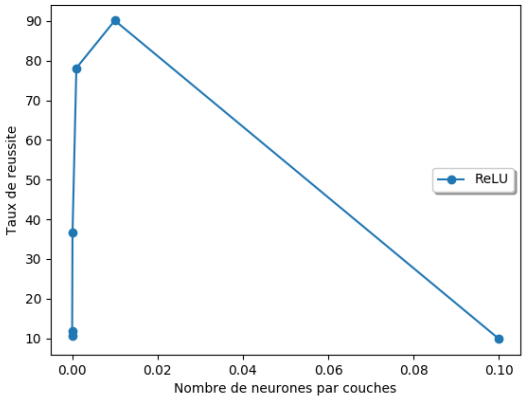


FIGURE 4 – Evolution du taux de réussite en fonction du pas

Nous avons réalisé le même travail avec la fonction Tanh. Avec un paramétrage correct des réseaux, nous obtenons le graphique ci-dessous.

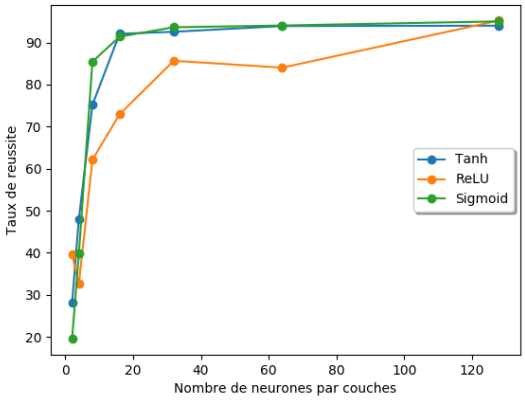


FIGURE 5 – Evolution du taux de réussite en fonction du nombre de couches et de la fonctions d'activation

Il ne semble pas évident de dire quelle fonction d'activation utiliser dans le cas de la base d'apprentissage MNIST. Comme la figure ci-dessus le montre, un bon paramétrage permet d'obtenir des résultats équivalents. Les fonctions Sigmoïd et Tanh ont l'avantage d'être bornées entre 0 et 1 ce qui permet - contrairement à la fonction ReLU - d'éviter une explosion de l'activation (car bornée en +inf). Cependant, ReLU permet de faire une descente de gradient plus efficace. En effet, les dérivées de Sigmoïd et Tanh tendent vers 0 aux extrémités. Ainsi, la descente de gradient et la convergence peuvent devenir très lentes. Enfin, ReLU à l'avantage d'être moins couteuse en opérations que les autres fonctions car moins compliquée mathématiquement.

4.2 Méthodes de gradient

Nous allons essayer les algorithmes d'optimisation Adam et Adagrad. Nous pouvons comparer sur les figures les différents algorithmes d'optimisation pour modifier les poids. Chaque algorithmes d'optimisation adapte implémente une rétro-propagation de gradient différente.

Sur la figure 7a, Nous avons utilisé une sigmoïde de 3 couches de 20 neurones. Le réseau s'est entrainé sur 20000 données. Nous avons vu précédemment que c'était très difficile de converger avec autant de couches.

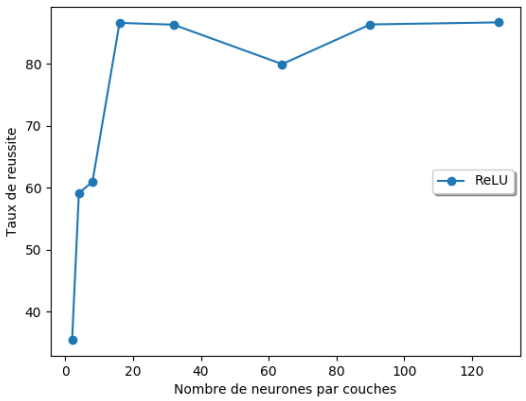
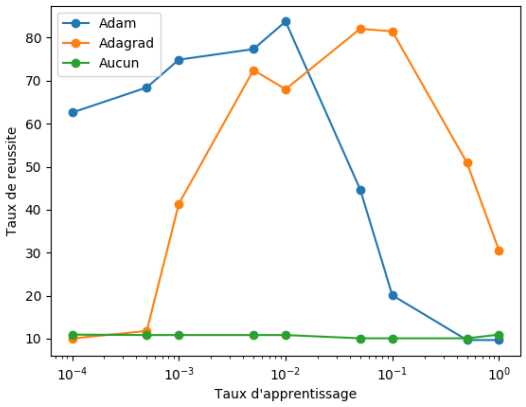
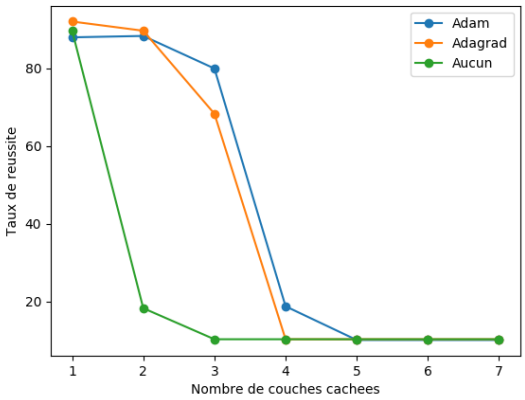


FIGURE 6 – Evolution du taux de réussite en fonction du nombre de neurones par couche



(a) Evolution du taux de réussite en fonction du pas d'apprentissage et de l'algorithme d'optimisation



(b) Evolution du taux de réussite en fonction du nombre de couches

On observe que Adam et Adagrad s'adaptent bien mieux au pas d'apprentissage que la rétro-propagation de gradient classique. Le paramétrage est bien plus facile à trouver, malgré des maximums.

Dans notre deuxième expérience, figure 7b, nous utilisons une sigmoïde avec des couches de 20 neurones. Le réseau s'entraine sur 40000 données, mais le taux d'apprentissage change pour chaque algorithme. Nous voyons que Adam est le plus résilient au nombre de couches, bien qu'Adagrad soit également très performant.

4.3 Mini-batchs

Jusque le nous utilisions un batch de taille 1, le gradient utilisé était donc une grosse approximation du gradient réel et variait beaucoup, car propre à une seule donnée. Nous essayons ici plusieurs batch. Nous utilisons l'algorithme de modification de poids d'Adam et une couche cachée de 16 neurones. Nous faisons varier les autres paramètres pour chaque taille de lot.

Nous n'avons pas fait de graphique, mais avons observé que l'algorithme est beaucoup plus rapide lorsque la taille du lot est grande, mais nécessite beaucoup plus de mémoire. Par ailleurs, le gradient réel est beaucoup mieux approché avec des lots que lorsque nous n'utilisons pas de lot (Taille de lot égale à 1). Assez rapidement cependant, il atteint son maximum, il est donc inutile d'utiliser des lots trop importants. Des lots de taille 10 conviennent parfaitement à notre réseau.

4.4 Résultat final

Finalement, notre paramétrage optimal de réseau entièrement connecté permet d'obtenir 97 à 98% de réussite sur la base de test dans un temps raisonnable.

Nous utilisons l'algorithme d'Adam, avec 128 neurones répartis dans une seule couche et une taille de lot de 10 neurones. Comme fonction d'activation, nous préférons utiliser ReLu, laquelle est rapide à calculer.

Nous aurions aimé essayer d'utiliser un réseau convolutionnel, mais n'avons pas eu le temps. Nous aurions sûrement pu nous approcher encore plus des 100% de réussite.