Introduction à l'apprentissage profond

Rapport

-

Mathieu Lefort

BRIGNONE Jean p1709655

ABDRABO Khaled p1713323

Partie 1 : Perceptron

Dans ce TP, nous avons travaillé sur plusieurs méthodes d'implémentation de perceptrons. Notre fichier de base, **mnist.pkl**, comporte 70 000 images labellisées représentant des chiffres écrits à la main, allant de 0 à 9. Ces images sont réparties en un jeu d'entraînement de 63 000 entrées et un jeu test comportant les 7 000 autres entrées. Ces images sont composées de 784 pixels (28 × 28).

1.1 Tenseurs

Les différents tenseurs présents dans le fichier perceptron_pytorch.py sont les suivants :

- Data_train : Ce tenseur est de taille 63 000 × 784. Il comporte la totalité des entrées qui composent le jeu d'entraînement. Il y a 63 000 entrées dans ce jeu, et 784 pixels pour chacune des entrées.

- Label_train : Ce tenseur est de taille 63 000 × 10. Il comporte la totalité des labels correspondant aux entrées du jeu d'entraînement. Il y a 63 000 entrées dans ce jeu, et 10 sorties possibles, une pour chaque chiffre dessiné.

- Data_set : Ce tenseur est de taille 7 000 × 784. Il comporte les entrées du jeu de test qui est composé de 7 000 entrées. Chaque entrée est composée de 784 pixels dans le jeu de test.

- Label_test : Ce tenseur est de taille 7 000 × 10. Il comporte les labels qui correspondent aux entrés du jeu de test. Il est composé de 7 000 entrées pour 10 sorties possibles.

- Y : Ce tenseur est de taille 5 × 10. Il correspond à la prédiction du modèle pour chaque entrée de notre batch. Chaque batch est composé de 5 images, et les sorties possibles sont au nombre de 10. Chaque sortie aura une valeur qui aura été calculée, c'est la prédiction.

- W : Ce tenseur est de taille 784 × 10. Il représente les poids (weights) des connexions et est composé d'autant de lignes que de neurones sur la couche d'entrée et d'autant de colonnes que de neurones sur la couche de sortie.

- Grad : Ce tenseur est de taille 5 × 10. Il correspond au gradient des entrées, qui est la différence entre le label réel et la sotie prédite par le modèle. Pour chaque batch. Chaque batch est composé de 5 images et il y a 10 sorties possibles.

- B : Ce tenseur est de taille 1 × 10. Il correspond au biais associé à chaque sortie.

- X : Ce tenseur est de taille 5 × 784. Il correspond aux entrées que l'on donne au modèle. Le fichier sur lequel nous travaillons fonction avec des batchs de taille 5. Les exemples sont donc donnés au modèle par groupes de 5 images, comportant chacune 784 pixels.

- T : Ce tenseur est de taille 5 × 10. Il correspond aux labels des entrées données au modèle. Chaque batch est composé de 5 images et il y a 10 sorties possibles.

1.2 Paramètres

1.2.1 Taux d'apprentissage η (eta)

Nous avons 784 neurones dans la couche d'entrée. Les images sont composées de 28 × 28 pixels, ce qui fait 784. Chaque image représentant un chiffre écrit à la main, il y a 10 sorties possibles, ce qui explique que la couche de sortie soit composée de 10 neurones.

Il nous reste à déterminer une valeur correcte de η (eta), le taux d'apprentissage.

Si l'on considère un **eta** de **0**, la nouvelle valeur des poids, d'une itération à une autre, sera toujours de 0. Le réseau n'apprend jamais.

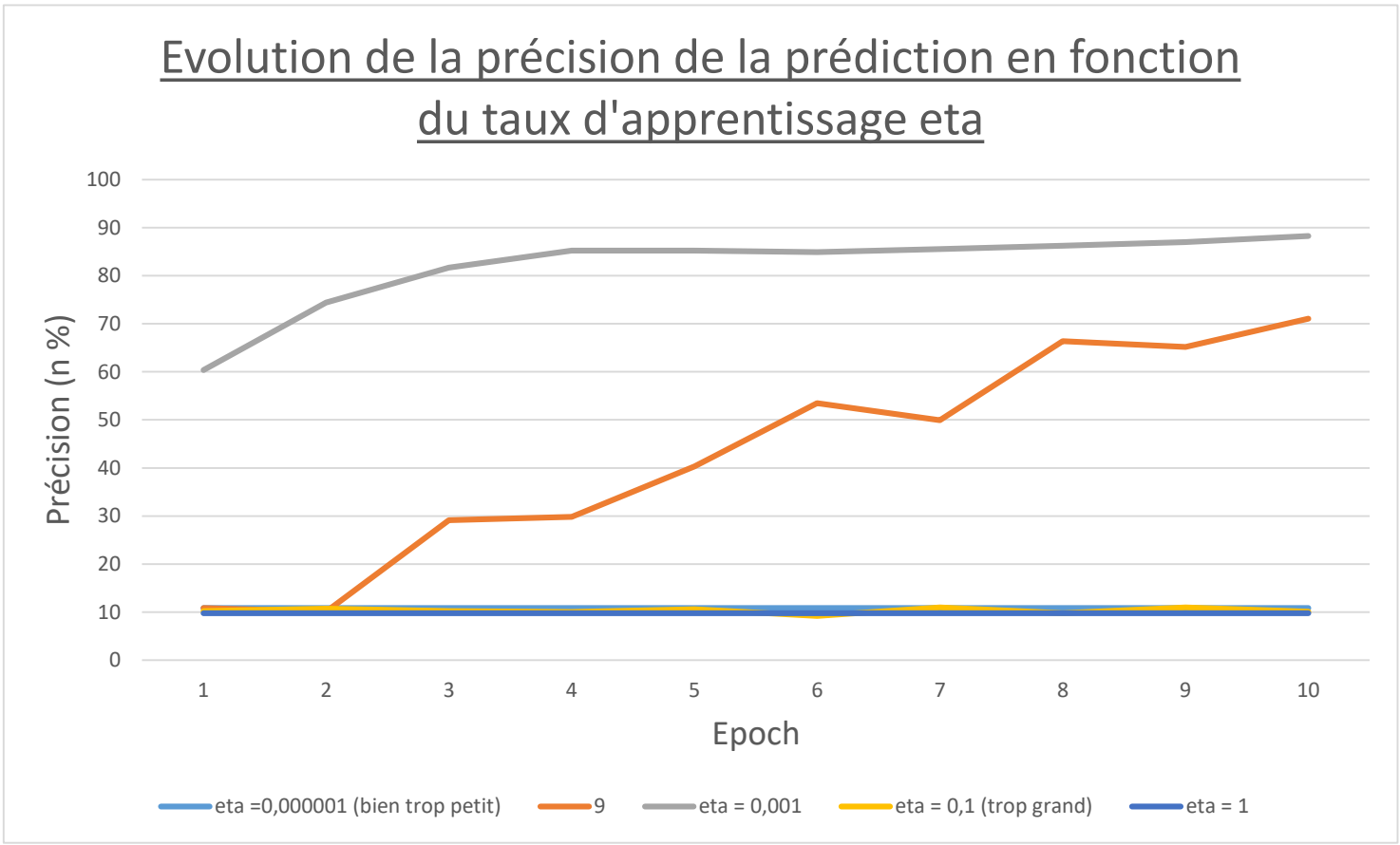
Si **eta** est bien trop petit **(0.000001**), le modèle va converger de manière extrêmement lente. Il faudrait un grand nombre d'itérations pour parvenir à un résultat intéressant. Avec 10 itérations, l'évolution est négligeable.

Si l'on considère un **eta** de **0.0001** (légèrement petit), nous pouvons observer une convergence ni rapide ni lente. Les valeurs peuvent varier de manière plus ou moins importante d'un epoch à l'autre.

Si l'on considère un **eta** de **0.001**, la convergence est rapide et les résultats, au bout de 10 epochs sont concluants.

Si l'on considère un **eta** de **0.1** (trop grand), les poids vont varier de manière très importante et le modèle ne dépassera pas les 11% de précision, en stagnant entre 9 et 11.

Si l'on considère un **eta** de **1**, le réseau va apprendre, à chaque itération la totalité de ses entrés et va stagner à 9.7% à cause de la descente du gradient.



1.2.1 Poids initiaux

Les poids minimum et maximum sont initialement fixés à [−0,001, 0,001]. C'est-à-dire qu'ils sont tirés aléatoirement dans cette plage de valeurs. Si l'on considère une plage moins étroite et que l'on fixe ces extrêmes à [−0,01, 0,01] par exemple, les poids ne seront logiquement plus aussi proches les uns des autres que dans l'ancienne plage. Cela signifie que certaines valeurs extrêmes pourront faire varier de manière importante les sorties qui sont des sommes de chaque entrée pondérée par un poids. Ce paramètre permet donc de donner ou non de l'importance aux valeurs s'éloignant des poids moyens des entrées. Nous avons choisi de laisser les poids initiaux aux valeurs données dans le fichier de base.

Partie 2 : Shallow Network

Cette partie a été réservée à l'implémentation d'un perceptron multi couches avec une seul couche caché et une sortie linéaire avec les outils de Pytorch.

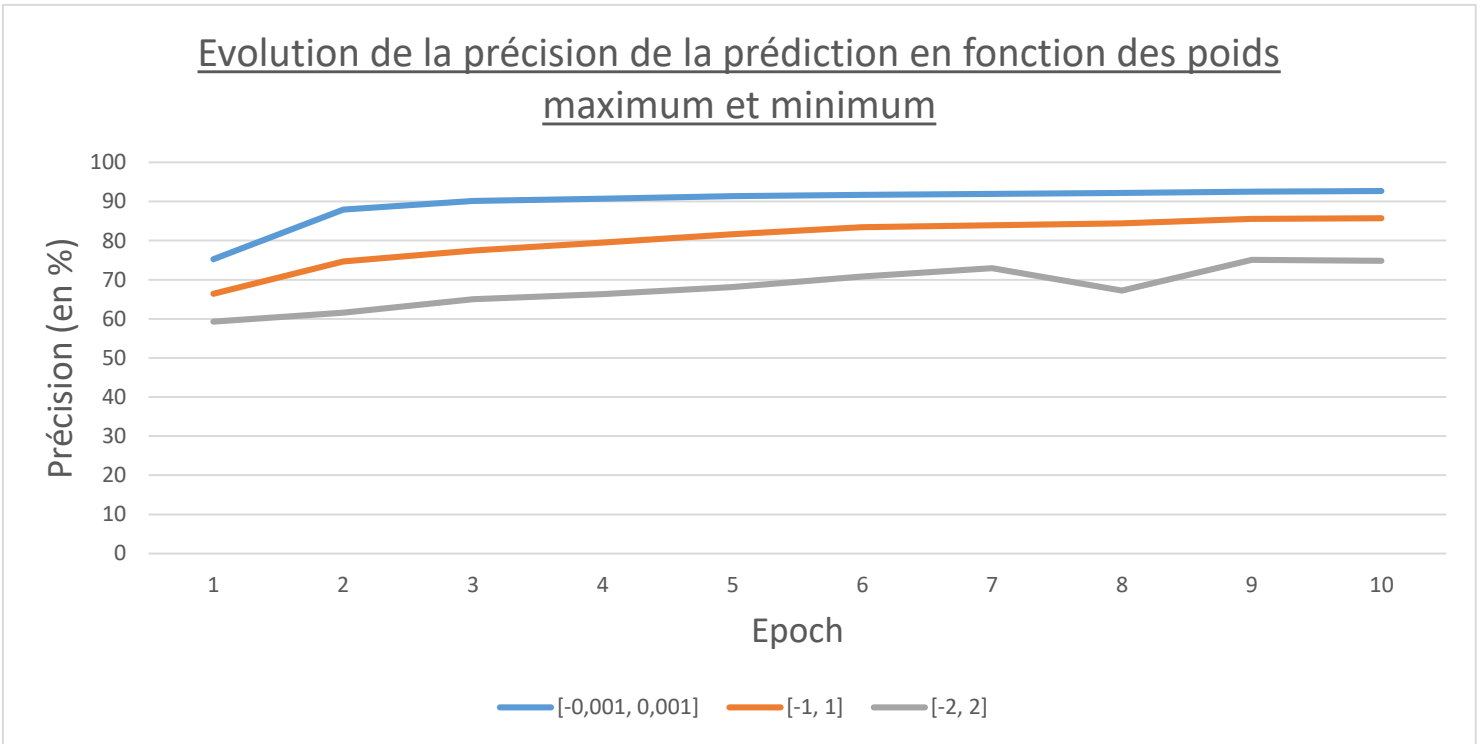
2.2 Paramètres

2.2.1 Taux d'apprentissage η (eta)

Le taux d'apprentissage reste selon nous meilleur lorsqu'il est de 0.001. Le modèle donne des résultats satisfaisants avec cette valeur. Il nous permet d'atteindre 92% de précision en 10 epochs.

2.2.2 Poids initiaux

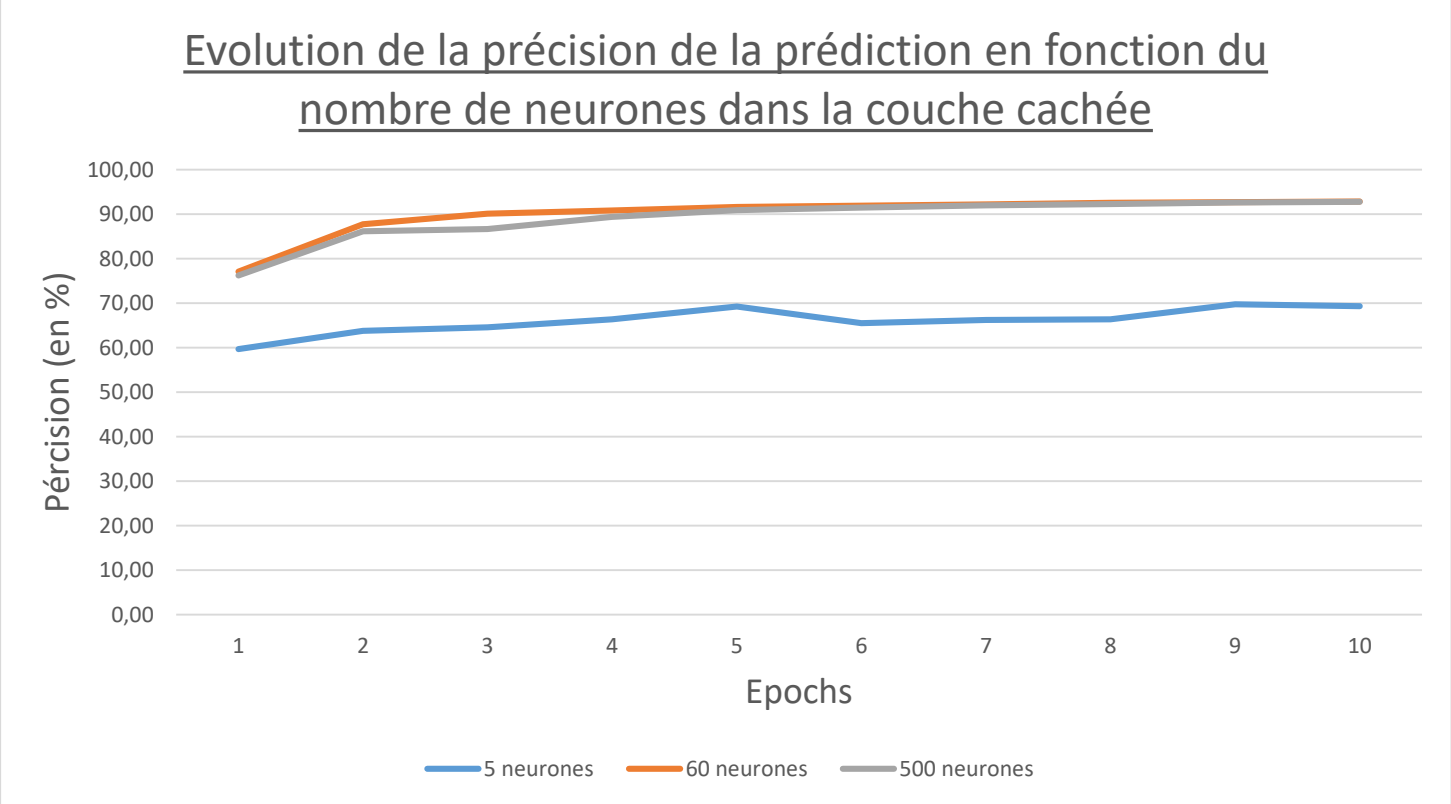
Les résultats sont toujours meilleurs avec une paire de poids maximum et minimum de [-0.001, 0.001]. Cependant, avec une paire [-1, 1] ou encore [-2, 2], nous pouvons observer une convergence plus lente qu'avec nos valeurs initiales.



2.2.3 Nombre de neurones de la couche cachée

Lorsque nous mettons moins de neurones dans la couche caché que dans la couche de sorti, nous notons une baisse importante des performances. Le réseau effectue une compression dans la couche cachée ce qui provoque une perte d'information et donc un moins bon score de précision.

Il est plus intéressant d'avoir plus de neurones dans la couche caché que dans la couche de sortie.



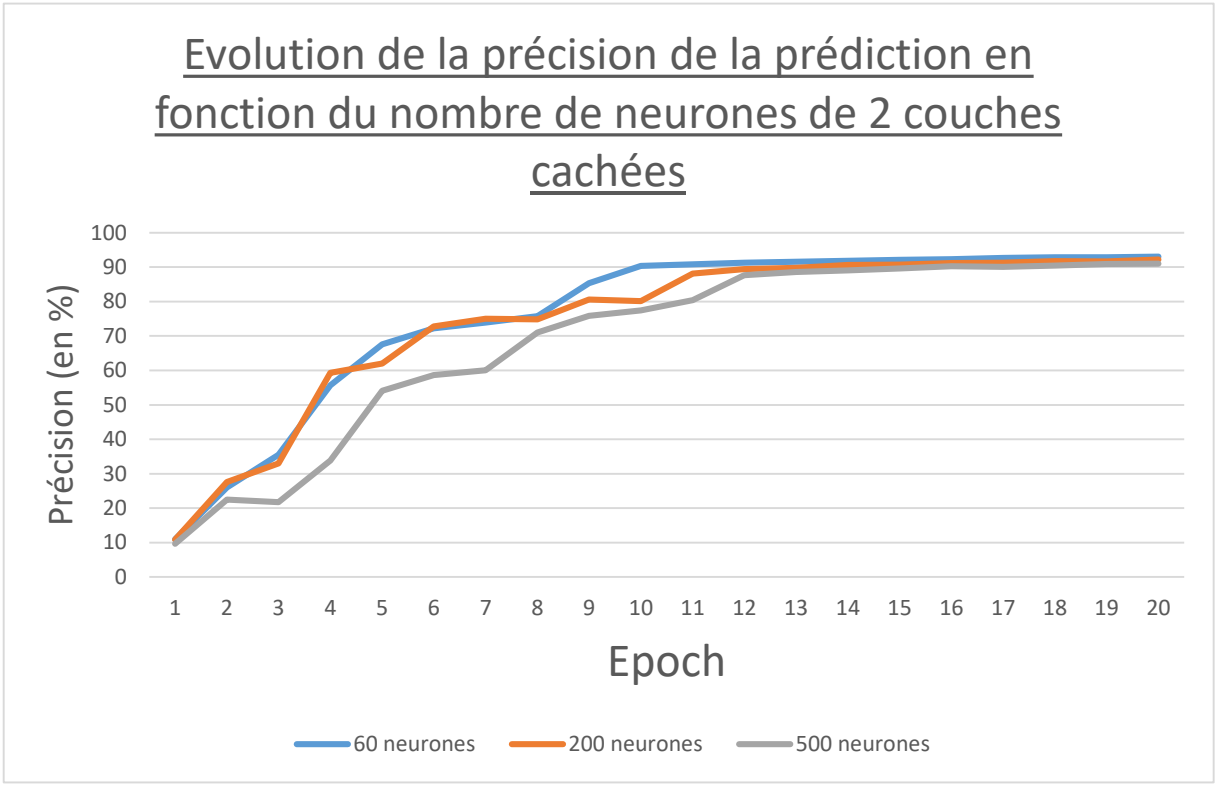
Au-delà de d'une soixantaine de neurones dans la couche cachée, les résultats sont similaires mais le temps de calcul est bien plus long. Nous avons donc décidé de laisser 60 neurones dans la couche cachée.

Partie 3 : Deep Network

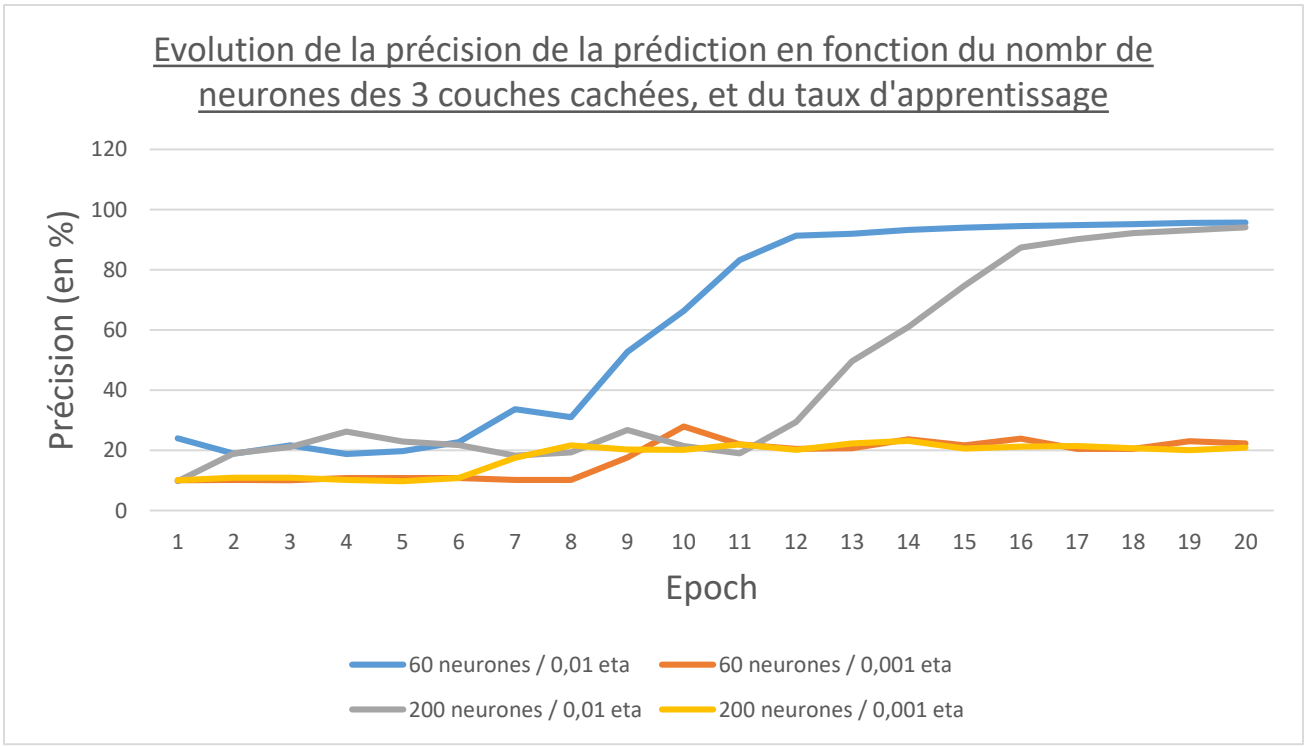
Les conclusions que nous tirons dans cette partie concernant les hyper paramètres sont les mêmes que pour les 2 parties précédentes. Nous considérerons toujours que les meilleurs résultats sont obtenus pour un eta = 0.001 ainsi que pour une paire de poids maximum et minimum de [-0.001, 0.001].

Nous allons nous concentrer ici sur le nombre de couches cachées et le nombre de neurones par couche cachée, pour cela, nous passons nb_epochs à 20 pour par souci de précision.

Lors de la partie précédente, nous avons conclu que 60 neurones dans la couche cachée permettaient d'avoir de bons résultats. Nous vérifions si c'est toujours le cas pour plus de couches cachées :



On voit que les résultats pour 60 ou 200 neurones sont assez proches, mais la convergence est atteinte plus rapidement pour 60 neurones quand il s'agit de 2 couches cachées.



Nous voyons également que, pour 3 couches cachées, les meilleurs résultats sont atteints pour un eta=0.01 avec une convergence atteinte plus vite pour 60 neurones que pour 200. Les résultats sont cependant très légèrement meilleurs pour le modèle à 3 couches cachées avec un pic à 96% pour le modèle à 60 neurones contre 93% pour celui à 2 couches cachées