Projet d'informatique M2 PSA
Neural Network from scratch in Python
brouillon de l'optimisation

Leïla Godinaud

février 2021

Table des matières

1 Erreur significative

Pour les résultats graphiques on utilise quoi comme type d'erreur ?

1. Un truc de genre Mean Square Error comme la fonction loss de la librairie
2. un pourcentage de réponse arrondi fausse modifié pour éviter le minimum local en 0.5. J'ai fait un truc du genre si la réponse est entre 0.4 et 0.6 elle est considérer fausse, sinon si son arrondi est 1 au lieu de 0 c'est aussi faux et réciproquement. On appelle ça comment ?

C'est la différence entre Optimization learning curve et Performance learning curve, selon si on met l'erreur MSE ou notre round error en fonction des epochs.

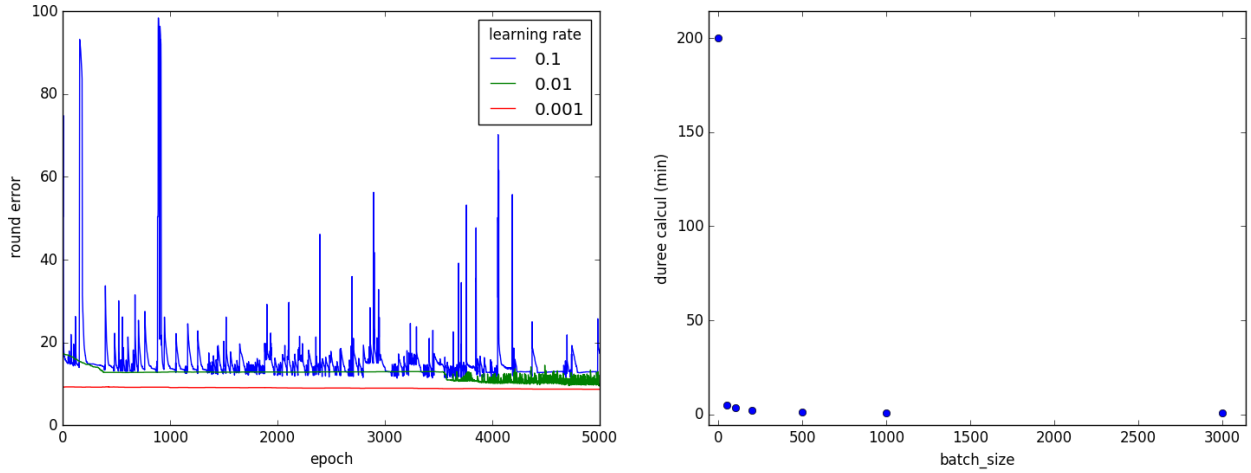
2 Optimisation hyperparamètres

2.1 Learning rate

Un learning rate plus faible diminue les fluctations et est donc favoriser. Par contre cela rallonge le temps de calcul et en particulier pour atteindre la meilleur efficacité.

2.2 Batchsize

C'est très approximatif c'est temps de calcul, mon ordi est très lent aussi, Pour le batch 1 c'est qu'une estimation, c'est plus la comparaison qui compte. Il faut mieux prendre du mini-batch vers quelques centaine donc pour optimiser le temps de calcul.

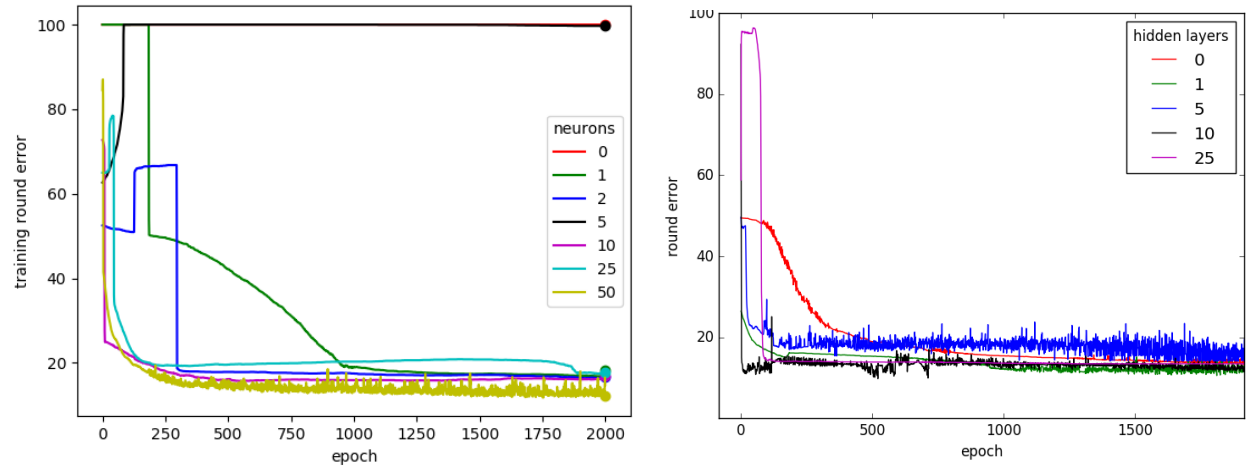


(a) (b)

Figure 1: *(à gauche :)* Evolution de l'erreur pour différent learning rate *(à droite :)* Evolution du temps de calcul pour différent batch size

remarques :
- si pas de batch (batch size =1) ou faible : extrement long, un peu plus de fluctuation sur l'erreur
- si batch size = training size : le pourcentage d'erreur augmente que le minimal local à 0.5 est favorisé

3 Optimisation du réseau



(a) (b)

Figure 2: *(à gauche :)* Evolution de l'erreur pour différent nombre de neurones dans une seule couche cachée *(à droite :)* Evolution de l'erreur pour différents nombre de couche cachées. Les points à la fin de chaque courbe correspond à l'erreur pour le testing test pour vérifier qu'il n'y a pas d'overfitting

3.1 Nombre de neurone dans une couche

test réalisé avec les 6 paramètres et réseau type Linear(6,i)-tanh()-Linear(i,1)-Sigmoid(), learning rate = 0.001, batch size = 100

- pour un neuron piège du 0.5 plus fréquent
- même efficacité finale et même vitesse pour l'atteindre pour nbr de neurone supérieur à 3
- un peu plus de fluctuation quand nombre de neurones augmente mais même pas systématique

3.2 Nombre de couches cachées

La figure 2a semble indiquer que si il y a plus de couche, la convergence de l'algorithme sera plus efficace. Néanmoins un plus grand nombre de couche introduit aussi plus de bruit et une plus grande probabilité de tomber dans un minimal local.

4 Sur-apprentissage

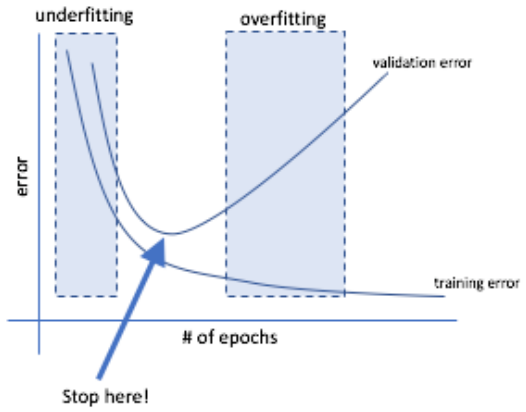
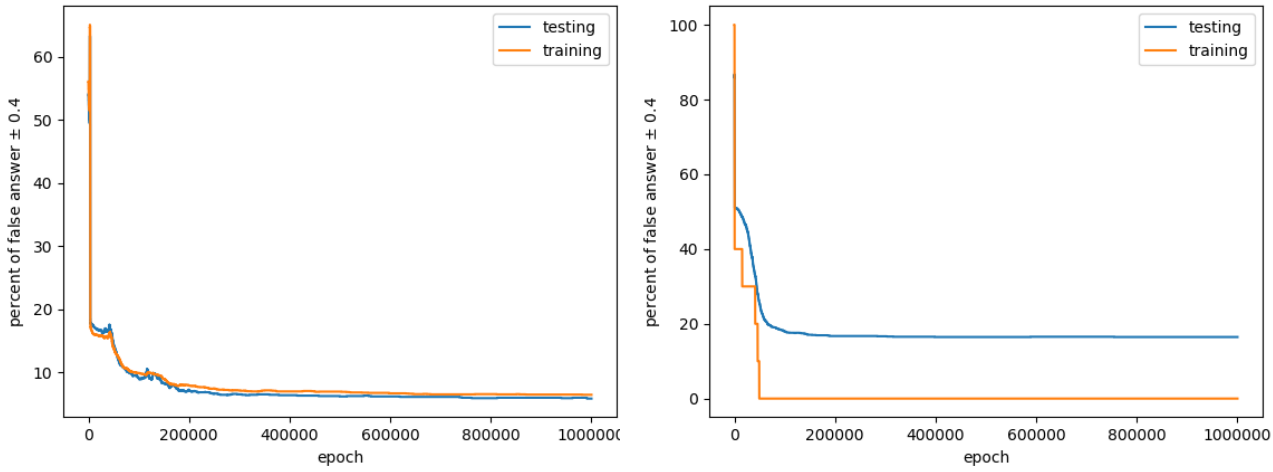


Figure 3

Cette figure revient toujours dans les articles de machine learning, ce serait bien que nous aussi on arrive à voir la limite de sur-apprentissage avec notre programme. Cela permet ensuite d'optimiser le nombre d'epoch.
Le problème c'est que je n'arrive jamais à voir cette divergence malgré des essaies varier comme ci dessous. En un sens c'est bien, ça signifie que notre réseau est bien réglé.



(a) (b)

Dans la figure a, le réseau était du type Linear(6,3)-tanh()-Linear(3,1)-Sigmoid(), learning rate = 0.001, batch size = 100. Le programme a tourné durant 4h30 pour 1 million de tour. Il n'y a pas de divergence, les valeurs finales sont 6.5% d'erreur pour le training set et 5.8% pour le testing set. Les données ont été enregistrées sur un csv.

En général les conseils pour éviter l'overfitting sont : augmenter le nombre de données pour l'entraînement, complexifier le réseau de neurone, augmenter le nombre de paramètres à traiter. Donc j'ai essayer d'entraîner l'algorithme avec moins de données pour obtenir cette figure. J'ai testé avec le même réseau et même hyper-paramètres mais avec seulement 10 évenements dans le set de training. C'est la figure (b). C'est assez choquant, on obtient 20% d'erreur seulement sur le testing set après !

Je vais tester avec un réseau plus complexe et moins de paramètres.

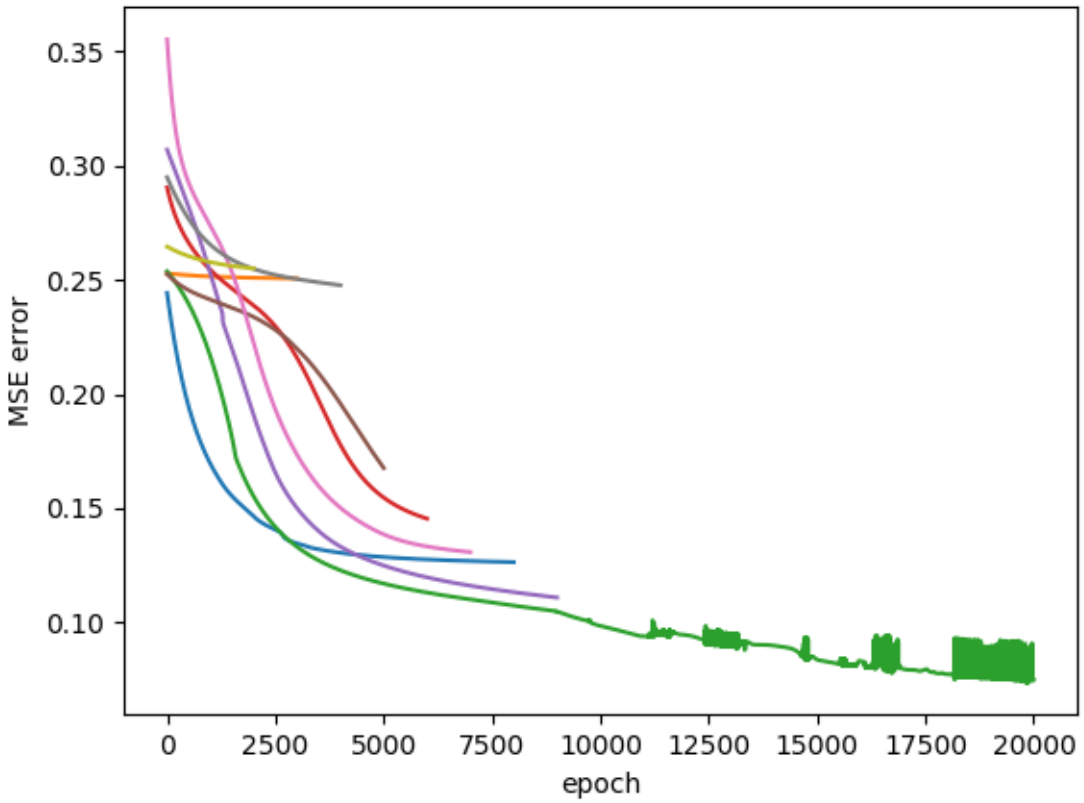


Figure 5

5 Simultaneous Neural Network

Idée d'un article (http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.100.2375rep=rep1type=pdf) sur comment éviter les minima locaux, comme le 0.25 pour notre fonction d'erreur MSE.loss() qui correspond à la prédiction 0.5 aussi bien pour le bruit et le signal.

L'idée est d'entraîner simultanément neuf réseaux de neurones. Arrivés à 10% du nombre maximal d'itération, on supprime le réseau le moins performant et ainsi de suite tout les 5% jusqu'à 50%. Le réseau qui reste fini d'être entraîné et est utilisé ensuite.

Pour la figure 5, il a été testé avec un réseau de type : Linear(6,5)-tanh()-Linear(5,4)-tanh()-Linear(4,3)-tanh()-Linear(3,2)-tanh()-Linear(2,1)-Sigmoid(), learning rate = 0.0001, batch size = 100. On voit que le minimum local à 0.25 est bien évité.

6 A discuter en plus

1. comment éviter piège 0.5 : mettre une pénalisation sur le calcul de l'erreur par la fonction loss quand le chi2 est égale à 0.25
2. où mettre la seed pour que ça marche ?
3. choix des paramètres optimaux parmi les 6 qu'on a
4. Impact du pre-processing de Quentin