IFT 3395 Fondements de l'apprentissage machine

Prof. Guillaume Rabusseau

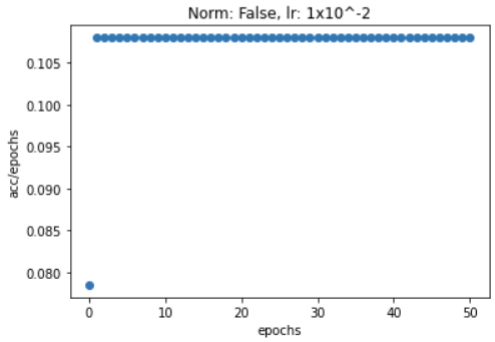
Devoir 3 - Rapport

Gabriel Ménard, Kevin Lessard

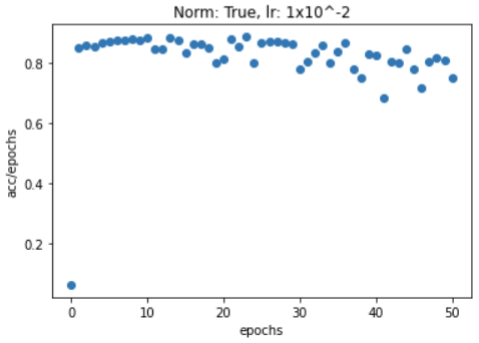
December 2021

Question 2

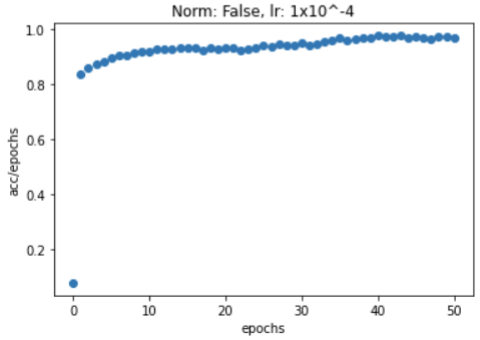
```
# normalisation: False lr: 2
y = dict_norm_false_2['train_accuracy']
plt.scatter(x, y)
plt.xlabel("epochs")
plt.ylabel("acc/epochs")
plt.title("Norm: False, lr: 1x10^-2")
plt.show()
```



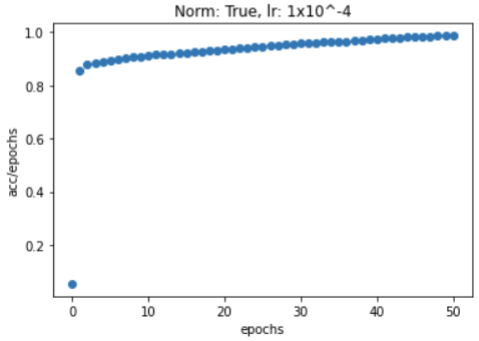

```
# normalisation: True lr: 2
y = dict_norm_true_2['train_accuracy']
plt.scatter(x, y)
plt.xlabel("epochs")
plt.ylabel("acc/epochs")
plt.title("Norm: True, lr: 1x10^-2")
plt.show()
```




```
# normalisation: False lr:4
y = dict_norm_false_4['train_accuracy']
plt.xlabel("epochs")
plt.ylabel("acc/epochs")
plt.title("Norm: False, lr: 1x10^-4")
plt.scatter(x, y)
plt.show()
```

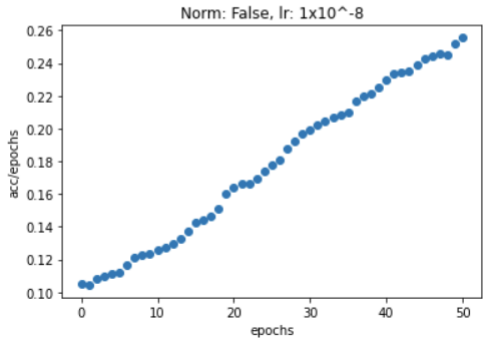


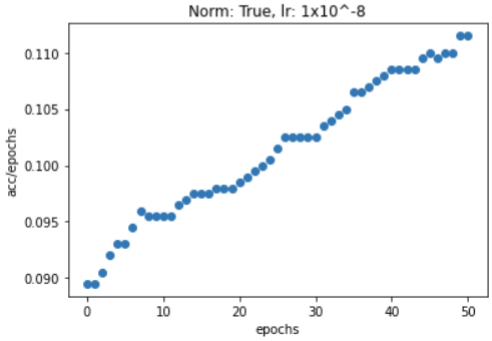

```
# normalisation: True lr: 4
y = dict_norm_true_4['train_accuracy']
plt.scatter(x, y)
plt.xlabel("epochs")
plt.ylabel("acc/epochs")
plt.title("Norm: True, lr: 1x10^-4")
plt.show()
```



```python
# normalisation: False lr: 8
y = dict_norm_false_8['train_accuracy']
plt.scatter(x, y)
plt.xlabel("epochs")
plt.ylabel("acc/epochs")
plt.title("Norm: False, lr: 1x10^-8")
plt.show()
```

```python
# normalisation: True lr: 8
y = dict_norm_true_8['train_accuracy']
plt.scatter(x, y)
plt.xlabel("epochs")
plt.ylabel("acc/epochs")
plt.title("Norm: True, lr: 1x10^-8")
plt.show()
```





a)

En général, la normalisation permet de réduire la taille des nombres qui sont calculés dans le gradient. Donc, celle-ci permet de réduire le nombre de fois où on obtient un gradient non-nul. Ainsi, le réseau peut apprendre plus rapidement et plus précisément.

b)

On remarque que la normalisation aide à la convergence dans le cas où on a un petit taux d'apprentissage. Dans le cas sans normalisation, le gradient calculé est trop grand et on rebondit toujours à la même valeur à la place de converger. Donc, la précision reste basse. Même avec la normalisation, on remarque qu'on ne converge pas à cause du taux d'apprentissage qui est trop grand. Par contre, la précision est nettement meilleure à cause des plus petits gradients qui permet de rebondir moins haut dans la fonction concave.

Par contre, toujours dans le cas sans normalisation, on remarque que pour un taux d'apprentissage assez petit, on ne fait qu'accélérer l'apprentissage. Il est toujours possible de rebondir à la place de converger dû à la grosseur des gradients. Par contre, dans un cas où il faudrait utilisé une très petite valeur d'apprentissage pour peu d'itérations, on peu se permettre de ne pas normaliser.

Dans le cas où on a un bon taux d'apprentissage, par exemple avec 0.0001, on remarque qu'il est préférable de normaliser pour avoir des augmentations plus constantes et permettre de se rendre plus près du gradient nul. On remarque qu'il y a un certain rebondissement vers la fin des itérations. Ainsi, il est préférable de normaliser si le nombre d'itérations et le taux d'apprentissage sont

bien choisis.

c)

Un trop petit taux d'apprentissage va faire en sorte que l'algorithme de descente de gradient va rebondir dans la fonction de perte qui est une fonction concave. Ainsi, il est possible de ne pas arriver à la convergence et d'être pris avec une plus petite précision qui ne s'améliore plus d'itération en itération.

Un trop grand taux d'apprentissage va demander beaucoup plus d'itérations pour arriver à convergence et il se peut simplement que ça demande un temps trop élevé pour la tâche. Il est possible d'avoir des précision nettement efficace, mais il est possible aussi de créer de l'overfitting.

C'est pour cela qu'il faut toujours choisir un juste milieu pour notre taux d'apprentissage et faire des tests. Ici, $lr = 10^{-4}$ est la situation préférable avec normalisation.