Rapport AMAL TME 01 02 03

TOUALBI Ali

Sorbonne université

18 octobre 2021

Table des matières

1 TME1

1.1 Question 6

en utilisant les résultats obtenus aux questions précédentes j'ai réussi à implémenter les classes nécessaires pour la régression linéaire : la fonction linéaire et la fonction de coût MSE. Et j'ai pu tester mes implementation en utilisant le code fourni dans tp1_gradcheck.py où la fonction gradcheck me renvoi à chaque fois TRUE pour les deux classes MSE et Linear

1.2 Question 7

pour la descente de gradient, j'ai implementé l'algo en utilisant les classes que j'avais défini à la question précédente. pour la valeur de epsilon, j'ai effectuer plusieurs tests, pour trouver que la valeur optimal est 0,0005, car avec cette valeur la loss converge vers le minimum, et dans les autres cas y'a pas de convergeance. j'ai pu afficher l'evolution de la loss grâce à TensorBord où j'ai récupérer à chaque epoch le loss calculé et je l'ai passé à la fonction add_scalar pour obtenir la figure 1

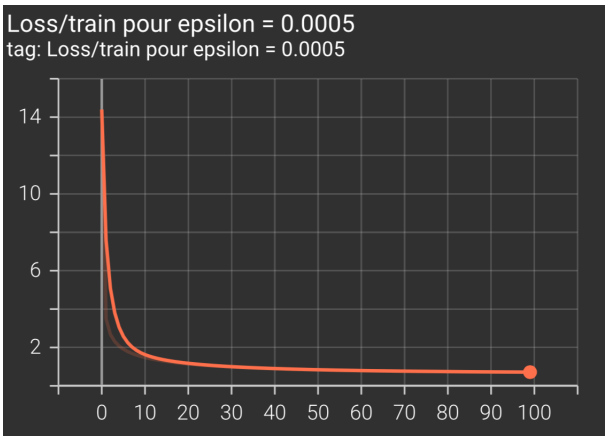

FIGURE 1 – l'evolution de loss par epoch

2 TME2

2.1 Question 1

pour l'implementation de la descente de gradient, il faut juste faire attention que l'opertation w=w-eps*grad ne rentre pas dans le graph de calcul, pour cela il faut juste acceder directement au renseur de données sans passer par la surcouche pytorch et donc l'opération n'est pas enregistrée dans le graph.autrement dit l'opertation devient w.data=w-ep*grad .

en implement les differentes descente de gradient, j'ai remarque que chaque type, il lui faut un epsilon different, et cela me parait logique car chaque descente prends en compte un nombre differents d'exemple et donc le pas de gradient joue un role important pour bien converger.Les résultats obtenus sont présentés dans les figure 2 figure 3 figure 4

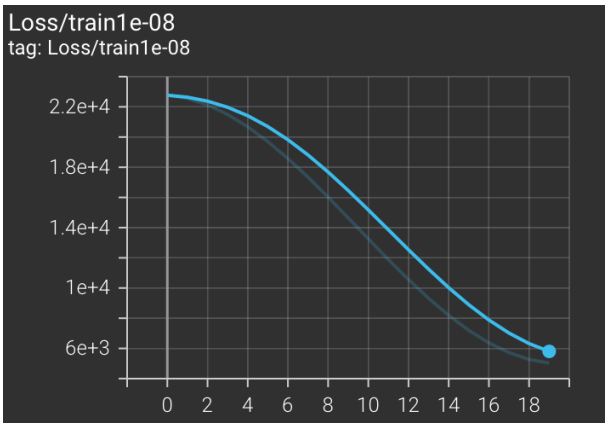

FIGURE 2 – descente batch

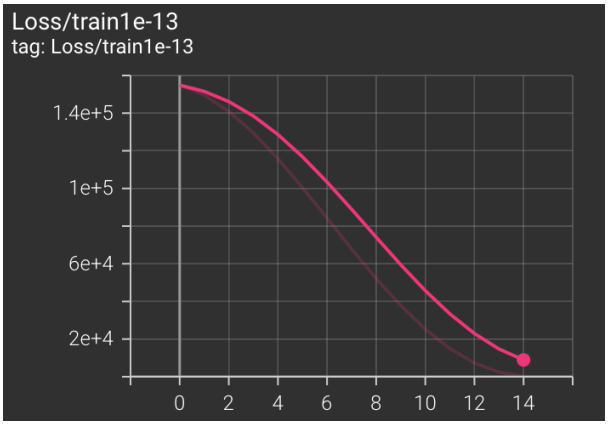

FIGURE 3 – descente stochastique

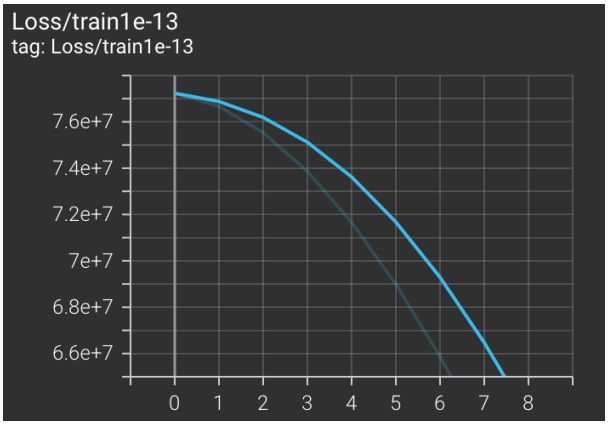

FIGURE 4 – descente mini batch

concernant la vitesse d'execution on remarque que la plus est celle de la descente en mini-batch, car on on effectue moin d'iteration en prenant batch par batch, contrairement à la descente stochastique où on prends exemple par exemple.Une autre remarque, la descente stochastique était possible dans cet exemple, car la quantitée des données est petite est donc on a pas besoin de beaucoup de mémoir, mais dans la réalité et pour des données plus large, la descente stochastique est quasiment impossible.

2.2 Question 2

pour cette partie, j'ai implementer un réseau à deux couches, linear -t̂anh -l̂inear -M̂SE. en utilisant un optimiseur fourni avec la librairie pytorch.l'evolution de loss est representée à la figure 5

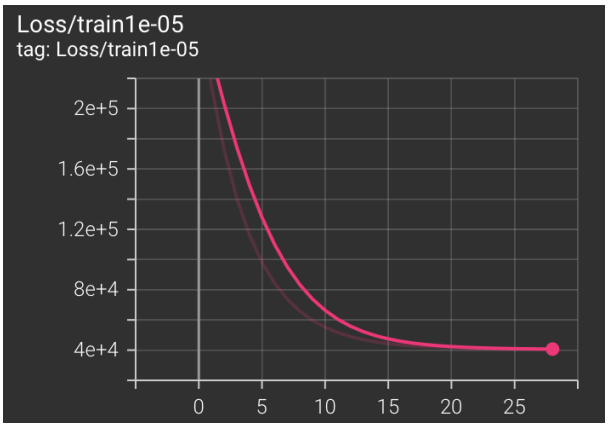

FIGURE 5 – l'évolution de loss pour le reseau à deux couches

3 TME3

3.1 Question 1

Pour cette partie j'ai implementer un DataSet en faisant attention à normaliser les valuer entre 0 et 1, et transformer les données en Float32.J'ai pu tester mon dataset en creant un dataloader qui renvoie des batchs de taille size_batch et qui sont des tuplets (inputs,labels).

3.2 Question 2

pour la partie autoencodeur, j'ai implementer un reseau qui compresse les données du format 28x28 en 500, puis reconstitus ces images.j'ai pu le tester un faisant une descente de gradient par mini-batch en utlisant le trainloader construit avec la base MNIST.les résultats obtenues sont présentés à la figure 6

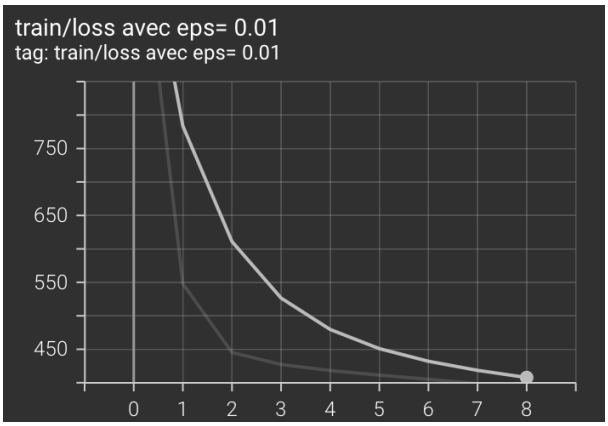

FIGURE 6 – l'évolution de loss pour l'autoencodeur

3.3 Question 3

Pour cette question j'ai implementer un reseau de neuronne avec 2 couches cachés avec l'architecture suivante : linear – relu – linear – relu – crossEntropy , j'ai fait attention à utiliser le GPU pour les calculs et le Checkpointing pour sauvegarder au fur et à mesure de l'apprentissage le modèle. les résultats obtenus sont présenté aux figure 7 figure 8 figure 9

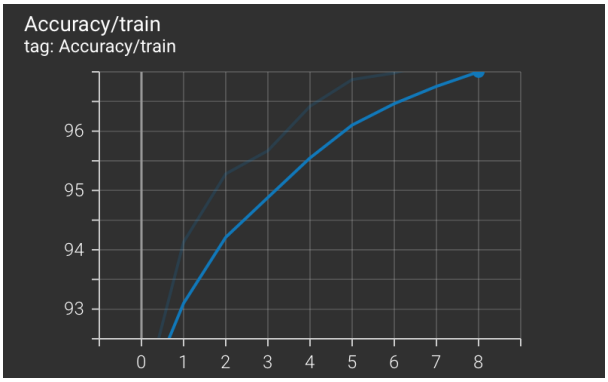

FIGURE 7 – l'évolution de l'accuracy

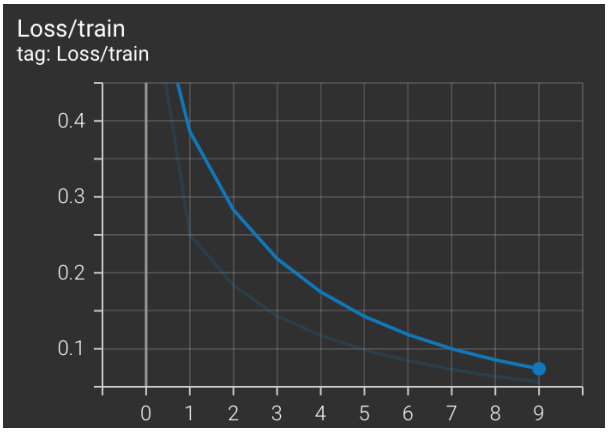

FIGURE 8 – l'évolution de train loss

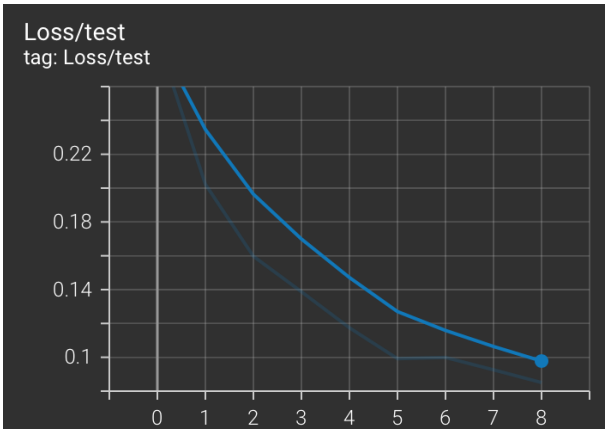

FIGURE 9 – l'évolution de test loss