GLO-4030/GLO-7030
Apprentissage par réseaux de neurones profonds
Date de remise : 10 février 2020 à 23h55

Travail pratique 1, en équipe de 1 à 2
Version 1.01

5 février 2020

Version 1.1 : La question 1 a été modifiée. Il y a aussi clarification pour 1.2.1, car il y a dédoublement de question avec 1.1a)

Version 1.0 : La question 3 a été ajoutée.

Attention ! N'oubliez pas d'attacher le code de toutes les questions dans la remise .zip du travail. Si le code est manquant, nous pourrons retirer jusqu'à 40% du total.

Pour tous les étudiants, vous devez fournir un rapport en un seul document (format pdf) et les fichiers de code zippé. Pour toutes les questions, n'oubliez pas de mettre le détail des calculs.

Instructions spéciales pour les étudiants en GLO-7030 : il y a plus de questions à répondre. Aussi, veuillez noter qu'une présentation déficiente dans le rapport (manque de clarté, orthographe et grammaire, police de caractère illisible sur figure, etc) pourra entraîner une pénalité allant jusqu'à 10 % de la note. Le rapport doit aussi obligatoirement être formaté avec LaTex. Plusieurs site web offrent des outils gratuits pour faire l'édition de manière collaborative, notamment sharelatex et overleaf. Assurez-vous que vos documents ne soient pas publics.

Question 1.1 (13 pts)

a) Dans `q1.py`, entraînez un réseau linéaire d'une seule couche sur MNIST pour une sortie de scores, et avec une fonction de perte `nn.MultiMarginLoss` ~~de type SVM multiclass. Notez que la fonction `nn.CrossEntropyLoss` de PyTorch inclut ce softmax directement, pour des raisons de stabilité numérique.~~ Au besoin, consultez la documentation de cette fontion.

b) Dans `q1.py`, entraînez un réseau linéaire d'une seule couche sur MNIST pour une sortie softmax, et avec une fonction de perte de type cross-entropy. Notez que la fonction `nn.CrossEntropyLoss` de PyTorch inclut ce softmax directement, pour des raisons de stabilité numérique. Au besoin, consultez la documentation de cette fontion.

c) Écrivez une fonction permettant de visualiser les poids pour chaque classe, pour produire des images comme dans l'acétate intitulée "Poids W appris : patrons 28 x 28". Ajouter ces images produites dans le rapport.

d) Qu'observez-vous ?

e) Assurez-vous que la sortie softmax soit présente sur votre réseau, et cette fois utilisez plutôt une perte de type Mean Square Error (MSE), qui est moins bien adaptée au softmax. Commentez sur la vitesse d"entraînement et le résultat final.

Question 1.2 (5 pts GLO-7030 seulement)

Ici vous poursuivez les expérimentations sur le même réseau et les mêmes données qu'à la question précédente.

1. **(GLO-7030 seulement) Ceci est un dédoublement de la question 1.1a) par erreur, donc reprenez votre code de 1a pour répondre à cette question.** Au lieu d'utiliser une sortie softmax, entraînez la sortie du réseau linéaire directement, mais avec la perte `nn.MultiMarginLoss`. Encore une fois, commentez sur la vitesse d'entraînement et le résultat final sur le jeu de données de test. Comparez avec les deux autres approches précédentes (softmax+cross-entropy, softmax+MSE).
2. (GLO-7030 seulement) La perte `nn.MultiMarginLoss` peut en théorie devenir nulle. En utilisant des batch de taille 8 images, faites un graphique qui montre le pourcentage de batch ayant des pertes nulles, en fonction de l'epoch.
3. (GLO-7030 seulement) Qu'advient-il de l'apprentissage, pour ces batch aux pertes nulles ?

Question 2 (12 pts)

a) Dans `q2.py`, développez une architecture de réseau de neurones utilisant uniquement des couches pleinement connectées (fully connected) avec fonction d'activation ReLU, et obtenez un taux d'erreur de 0% sur le jeu d'entraînement de MNIST. Rapportez le taux d'erreur sur le jeu de test et insérez les graphes d'entraînement dans le rapport. Notez que vous devez trouver vous-même le nombre de couches.

b) Est-ce que le réseau overfit ? Justifiez.

Question 3 : Propagation du gradient (20 pts)

Dans cette question, vous allez explorer l'impact de la fonction d'activation et de l'initialisation sur la propagation du gradient, au travers des couches d'un réseau pleinement connecté. Comme vous le savez, la bonne propagation du gradient est essentielle à la réussite d'un entraînement. Vous allez reprendre l'architecture pleinement connectée développée à la question 2, mais en utilisant exactement 4 couches cachées. Vous aller aussi utiliser le jeu de données MNIST pour faire les expériences, avec des batch de 32 images.

Pour cette question, vous devez comparer la distribution des gradients en fonction de la profondeur, pour :

— les fonctions d'activation suivantes : ReLU et sigmoide ;

— les initialisations `Normal(0, 1)` et `Xavier_Normal(1)`. Pour `Xavier`, initialiser les biais à 0. Le code nécessaire pour faire les initialisations est le suivant :

```
def weight_init_normal(m):
  # puisque nous voulons apporter les modification seulement aux
  # couches fully connected
    if isinstance(m, torch.nn.Linear):
        torch.nn.init.normal_(m.weight)
        torch.nn.init.normal_(m.bias)

def weight_init_xavier_normal(m):
  if isinstance(m, torch.nn.Linear):
    torch.nn.init.xavier_normal_(m.weight)
    #torch.nn.init.xavier_normal_(m.bias) ne fonctionne pas sur les biais
    m.bias.data.zero_()
```

Notez que le learning rate devra être trouvé manuellement pour chacun des quatres cas (voir Table 1). Veuillez utiliser l'approche décrite en classe, c'est-à-dire de prendre le learning rate le plus élevé possible qui permet de converger dans l'entraînement.

TABLE 1 – Learning rate utilisé pour faire les entraînements.

	Normal	Xavier
ReLU		
Sigmoide		

Dans votre rapport, incluez les éléments suivants :

a) Des graphiques qui montrent a) la valeur moyenne absolue et b) l'écart-type des gradients, en fonction de l'itération et de la couche. Faites un graphique par couche, par type de réseaux (ReLU vs sigmoïde) et par activation. Vous aurez donc au total 32 graphiques pour ces gradients (4 couches $\times$ 2 fonctions $\times$ 2 initialisation $\times$ [moyenne/écart-type]). Par convention, la couche 1 sera la couche la plus près des entrées.

b) Discutez des différences de comportement des gradients entre ces quatres cas.

c) Incluez dans votre rapport les courbes d'entraînements qui montrent la précision en train, val, test en fonction de l'époque, pour les quatres cas.

d) Commentez sur les résultats (gradient, temps d'entrainement, précision sur test, etc.)

e) Incluez un tableau (Table 1) qui montre le learning rate utilisé pour faire l'entrainement.

Question 4 (35 pts pour GLO-4030, 50 pts pour GLO-7030)

Pour cette question, vous implémenterez un graphe de calcul en Python. Le squelette du code est fourni dans le dossier `scratch_net`. Dans ce dossier, il y a également un fichier de tests. Ces tests vous indiqueront facilement si votre code fonctionne ou non et serviront à la correction. Pour cette question, seulement le code est corrigé, à l'exception de la question d) réservé à GLO-7030. Si vous avez des précisions sur votre implémentation, indiquez-les dans votre rapport.

a) Allez dans le fichier `operation.py` et remplissez les **TODO** qui se trouvent dans les fonctions `apply()` des classes `Operation`. Une fois cela terminé, le test `test_simple_equation_forward()` devrait fonctionner. NOTE : Le code de `AddOperation` est fourni, vous pouvez vous en inspirer pour les autres opérations.

b) Allez dans le fichier `scratch_grad.py` et complétez la fonction `_build_gradient()`. Cette fonction est copiée sur l'algorithme 6.6 du livre du cours (page 210), à quelques différences près. Par exemple, il n'y a pas de `grad_table` globale, car chaque objet `Variable` sauvegarde son propre gradient dans l'attribut `self.gradient`. De même, il est possible de retrouver les valeurs que retourneraient `get_consumers`, `get_operation` et `get_inputs` à partir des attributs des objets `Variable` et `Operation`. NOTE : Vous n'avez pas à coder l'algorithme 6.5 du livre, un équivalent est déjà fait pour vous dans les fonctions `backprop()` et `_build_gradient_all_parents()`.

c) Allez dans le fichier `operation.py` et remplissez les **TODO** qui se trouvent dans les fonction `derivate_inputs()` des classes `Operation`. NOTE : un **TODO** est réservé aux étudiants de **GLO-7030**. Une fois cela terminé, les tests `test_simple_equation_back()` et `test_neural_net_without_loss()` devraient fonctionner. Pour les étudiants de **GLO-7030**, tous les tests devraient fonctionner.

d) **GLO-7030 seulement** Dans le fichier `train.py`, écrivez du code qui fait la mise-à-jour d'un réseau similaire à celui dans le fichier de tests pour dix itérations. À chaque itération, créez un graphe de calcul à partir d'un vecteur d'entrée aléatoire et des paramètres du réseau, faites la backpropagation puis soustrayez aux paramètres $\alpha = 1 \times 10^{-2}$ fois leur gradient. Cette variable α correspond au learning rate.