IFT 3395 Fondements de l'apprentissage machine
Prof. Ioannis Mitliagkas

Devoir 3 - Partie pratique

- Ce devoir doit être déposé sur Gradescope et doit être fait en équipe de 3. Vous pouvez discuter avec des étudiants d'autres groupes mais les réponses soumises par le groupe doivent être originales. À noter que nous utiliserons l'outil de détection de plagiat de Gradescope. Tous les cas suspectés de plagiat seront enregistrés et transmis à l'Université pour vérification.
- La partie pratique doit être codée en python (avec les librairies numpy, matplotlib et PyTorch), et envoyée sur Gradescope sous la forme d'un fichier python. Pour permettre l'évaluation automatique, vous devez travailler directement sur le modèle donné dans le répertoire de ce devoir. Ne modifiez pas le nom du fichier ou aucune des fonctions signatures, sinon l'évaluation automatique ne fonctionnera pas. Vous pouvez bien sûr ajouter de nouvelles fonctions et importations python
- Tous les graphiques, tableaux, dérivations ou autres explications doivent être soumis à Gradescope sous forme de rapport pratique et **séparément de la partie théorique du devoir**. Vous êtes bien sûr encouragés à vous inspirer de ce qui a été fait en TP.

1 Entrainement de réseaux de neurones avec la différentiation automatique [25 points]

Cette partie consiste en la mise en place d'une classe `Trainer` pour entraîner des réseau de neurones pour la régression univariée. Elle sera basée sur PyTorch et inclura un ensemble de fonctionnalités. Vous devrez implémenter la plupart des fonctions requises dans la classe `Trainer` fournie dans le modèle de solution.

Vous explorerez divers perceptrons multicouches (MLP) et réseaux de neurones convolutifs (CNN). Vous entraînerez votre réseau de neurones avec une descente de gradient stochastique en mini-batch, en utilisant l'erreur quadratique moyenne comme fonction de coût et l'optimiseur Adam.

La classe `Trainer` est initialisée avec les paramètres suivants :

- Une clé de l'ensemble {'mlp', 'cnn'} désignant le type de réseau.
- `net_config` : une structure de données définissant l'architecture du réseau de neurones. Il doit s'agir d'un NamedTuple `NetworkConfiguration`, tel que défini dans le fichier de solution. Les différents champs du tuple sont des tuples d'entiers, précisant les hyperparamètres pour les différentes couches cachées du réseau, à savoir : `n_channels`, `kernel_sizes`, `strides` et `dense_hiddens`.
  - Si `network_type = 'cnn'`, un CNN doit être créé. Les 3 premiers champs spécifient la topologie des couches convolutives, tandis que `dense_hiddens` spécifie le nombre de neurones pour les couches cachées entièrement connectées pour la dernière partie du réseau. Par exemple, en appelant `NetworkConfiguration(n_channels=(16,32),`

`kernel_sizes=(4,5), strides=(1,2), dense_hiddens=(256, 256)`) on créera un CNN avec deux couches convolutives : la première couche aura 16 canaux, un 4×4 kernel et une "stride" de 1×1 ; la deuxième couche aura 32 canaux, un noyau de 5×5 et une "stride" de 2×2. Le dernier attribut signifie qu'il y a deux couches cachées entièrement connectées avec 256 neurones chacune dans la partie finale du réseau.

  - Si `network_type = 'mlp'`, seul l'attribut `dense_hiddens` est utilisé, et un réseau avec une couche d'entrée, deux couches cachées avec 256 neurones et une couche de sortie doit être construit.

- `lr`: le taux d'apprentissage utilisé pour entraîner le réseau de neurones avec Adam.
- `batch_size`: la taille des batchs pour Adam.
- `activation_name`: une chaîne de caractères décrivant la fonction d'activation à utiliser. Elle peut être `"relu"`, `"sigmoid"`, ou `"tanh"`. On vous rappelle les 3 fonctions d'activation:
  - $\text{RELU}(x) = \max(0, x)$
  - $\sigma(x) = \frac{1}{1+e^{-x}}$
  - $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

La fonction `__init__` vous est donnée. En plus du chargement du jeu de données et l'initialisation des variables de classe, cette fonction initialise un dictionnaire de logs d'entraînement, qui contient des informations sur les coûts et les métriques sur les ensembles d'entraînement et de test pendant l'entraînement.

1. [1 points] Complétez la méthode `Trainer.create_activation_function`, qui prend comme argument une chaîne dans l'ensemble {`"relu"`, `"tanh"`, `"sigmoid"`}, et qui retourne un objet de type `torch.nn.Module` implémentant cette fonction d'activation spécifique.

2. [5 points] Cette question concerne l'implémentation d'une fonction constructeur pour un MLP. Vous devrez compléter la fonction `Trainer.create_mlp`, qui a comme arguments la dimension d'entrée de la première couche, un objet `NetworkConfiguration` définissant la structure du réseau et une fonction d'activation sous la forme d'un objet `torch.nn.Module`. La fonction d'activation fournie doit être appliquée après chaque couche, à l'exception de la dernière couche, qui ne devrait pas avoir une activation. La fonction doit retourner un objet `torch.nn.Module` avec l'architecture souhaitée ; veuillez noter que le réseau doit prendre en entrée l'image sous format tensoriel et doit l'aplatir à l'interne. Le modèle doit avoir un neurone de sortie, qui est la prédiction de la régression. **Nous nous attendons à ce que les couches entièrement connectées soient des instances de `torch.nn.Linear`. Nous vous suggérons également d'utiliser un conteneur `torch.nn.Sequential`. Par exemple, un `hidden_sizes` de longueur 3 devrait impliquer un MLP de 4 couches `Linear` au total ; trois avec la fonction d'activation prescrite et une en sortie sans activation.**

3. [6 points] Cette question concerne l'écriture d'une fonction constructeur pour un CNN. Vous devrez compléter la fonction `Trainer.create_cnn`, qui a comme arguments le nombre de canaux de l'image d'entrée, un objet `NetworkConfiguration` et une fonction d'activation sous forme de `torch.nn.Module`. La fonction d'activation doit être appliquée après chaque

couche convolutive ou entièrement connectée, à l'exception de la dernière, qui ne devrait pas avoir une activation. Après la fonction d'activation de chaque couche convolutive, une couche `torch.nn.MaxPool2d(kernel_size=2)` doit être appliquée ; la dernière couche convolutive ne devrait pas avoir de pool maximum et être plutôt suivie d'un `torch.nn.AdaptiveMaxPool2d((4, 4))` et d'un `torch.nn.Flatten()`, juste avant la couche complètement connectée, pour s'assurer que le réseau est agnostique (jusqu'à un certain degré) à la taille d'entrée. La fonction doit retourner un objet `torch.nn.Module` avec l'architecture souhaitée. Le modèle doit avoir un neurone de sortie, qui est la prédiction de la régression. **Encore une fois, nous nous attendons à ce que les couches convolutives et entièrement connectées soient des instances de `torch.nn.Conv2d` et `torch.nn.Linear`. Nous vous suggérons également d'utiliser un conteneur `torch.nn.Sequential`.**

4. [4 points] Complétez la fonction `Trainer.compute_loss_and_mae` qui prend en entrée une tenseur d'images et un un tenseur d'étiquettes. La fonction doit renvoyer l'erreur quadratique moyenne ainsi que l'erreur absolue moyenne sur la batch donnée en entrée. Attention, les valeurs de loss et d'erreur absolue moyenne renvoyées doivent être des `torch.Tensor` gardant une trace des calculs qui y ont conduit, afin que la différenciation automatique puisse calculer les gradients avec l'algorithme de rétropropagation.

5. [6 points] Pour cette question, vous devez utiliser les fonctions précédentes que vous avez implémentées.

   Complétez la fonction `Trainer.training_step`. Cette fonction implémente une seule étape d'apprentissage, en prenant un batch des données et d'étiquettes comme entrées. Il sera ensuite utilisé à l'intérieur de la fonction `train_loop` pour avoir une procédure d'entraînement complète. Votre fonction doit calculer le gradient de la fonction de perte de votre réseau actuel et le mettre à jour à l'aide de l'optimiseur, puis retourner la loss et la MAE pour la batch donnée en entrée.

6. [3 points] Complétez la fonction `Trainer.evaluate`. Cette fonction doit retourner le coût moyen et l'erreur absolue moyenne sur un ensemble. Vous pouvez utiliser les fonctions qui vous sont déjà données.

2 Expérimentation sur le jeu de données Street View House Numbers (SVHN) [(bonus) 20 points]

Dans cette partie du devoir, vous allez faire quelques expérimentations, en utilisant votre classe `Trainer`, sur le jeu de données Street View House Numbers (SVHN). Ce jeu de données peut être chargé en utilisant la méthode `Trainer.load_dataset`.

Le jeu de données Street View House Numbers (SVHN) est composé d'images en couleur de 32x32 pixels de chiffres (10 classes différentes). Il y a 73 257 images d'entraînement et 26 032 images de test. Chaque image est représentée par un tenseur de dimension $3 \times 32 \times 32$, où la première dimension représente les trois canaux RGB composant l'image. Les variables `self.train` et `self.test` initialisées dans le constructeur de la classe `Trainer` sont des objets de type `DataLoader` de PyTorch. Ces objets sont des itérables qui produisent un tuple `(images, labels)` contenant un lot d'images avec leurs étiquettes correspondantes.

Les réponses aux questions de cette section, ainsi que les chiffres requis, doivent être écrits dans un rapport que vous soumettrez à Gradescope comme partie pratique du devoir (séparément de la partie théorique).

1. [10 points] Entraînez un MLP avec 2 couches cachées, de taille 128 et 128 respectivement sur le jeu de données SVHN, pour 50 époques, et une taille de batch 128. Utilisez la fonction d'activation ReLU. Pour des raisons de reproductibilité, **veuillez ne pas modifier la graine qui est déjà définie dans le fichier de solution**.

   Lorsqu'il n'est pas spécifié, laissez la valeur par défaut pour les arguments du `Trainer`. Incluez dans votre rapport la figure et la réponse à la question suivante :

   - Générez une figure avec l'erreur absolue moyenne du test en fonction d'époques croissantes pour des valeurs de taux d'apprentissage dans l'ensemble $\{0.01, 1\times10^{-4}, 1\times10^{-8}\}$. Quels sont les effets des taux d'apprentissage très petits ou très grands ?

2. [5 points] Entraînez un CNN sur le jeu de données SVHN pour 50 époques avec une taille de batch 128. Utilisez la fonction d'activation ReLU. Utilisez trois couches convolutionnelles cachées avec un nombre de filtres de $(16, 32, 45)$, des noyaux de taille 3, et un "stride" de 1. Utilisez une dernière couche cachée entièrement connectée avec 128 neurones. Pour des raisons de reproductibilité, veuillez ne pas modifier la graine qui est déjà définie dans le fichier de solution.

   Lorsqu'il n'est pas spécifié, laissez la valeur par défaut pour les arguments du `Trainer`. Incluez dans votre rapport la figure et la réponse à la question suivante :

   - Générer une figure avec l'erreur absolue moyenne de test en fonction de l'augmentation des époques pour le CNN ci-dessus.

3. [5 points] Expérimentons maintenant avec la signification de "profondeur" et de "largeur" dans les réseaux de neurones convolutifs. Comme vu précédemment, les interactions éparses ("sparse") sont l'une des caractéristiques de la convolution appliquée aux réseaux de neurones : cela signifie que, grâce à la localité des noyaux, seule une partie des features de l'échantillon interagissent. Par contre, dans les MLPs, la multiplication matricielle permet une interaction dense entre les entrées et les paramètres. Nous allons maintenant casser cette propriété de deux manières opposées.

   (a) Imaginez un **CNN profond** avec à peu près le même nombre de paramètres que l'original, 4 couches avec un nombre de filtres de $(8, 16, 32, 64)$, des "strides" de 1 et 0 de rembourrage. Cette fois, vous utiliserez un `kernel_size` de 1 dans toutes les couches convolutives, et un MLP final à deux couches cachées de 256 neurones. Quelle est la relation entre le type de couches convolutives utilisées par ce réseau et les couches entièrement connectées ?

   (b) Imaginez un **CNN pas profond** avec une seule couche convolutive, avec un `kernel_size` égal à la taille de l'image d'entrée (28×28) et un nombre de canaux dans cette couche tels que le nombre de paramètres est approximativement le même que celui des réseaux précédents. Considérez les autres hyperparamètres comme ceux du CNN précédent ("stride" de 1, zéro rembourrage, deux couches cachées finales de 256 neurones). Quelle est la relation entre le type de couches convolutives utilisées par ce réseau et les couches entièrement connectées ?

(c) Maintenant, entraînez le **CNN profond** et le **CNN pas profond** tels que définis ci-dessus pour 50 époques avec une taille de batch de 128 et mettez dans votre rapport un chiffre comparant les précisions de validation et d'entraînement de ces trois types de CNN (y compris le CNN original de la question précédente). Comment se comparent leurs performances générales et leurs capacités de généralisation ? Et quelle est votre hypothèse sur leurs différents comportements ?