A rendre le: 16 février 2019
Instructions

- *Montrez les traces de votre travail pour toutes les questions!*
- *Il faut soumettre votre code et votre rapport (pdf) via la page Gradescope du cours. Si vous utilisez le Jupyter Notebook, vous devez l'exporter en pdf, et le soumettre via Gradescope.*
- *Votre code doit être dans un dépôt Github (repository). Vous devez inclure le lien du dépôt dans votre rapport!*

Problème 1

Dans ce problème, nous allons construire un perceptron multicouche (*multilayer perceptron*, MLP) et l'entraîner sur l'ensemble de données de chiffres manuscrits MNIST [1].

Construction du modele [35] Considérons un MLP à deux couches cachées, avec un nombre de neurones h^1 et h^2 respectivement. Pour l'ensemble de données MNIST, le nombre de *features* des données d'entrée est $h^0 = 784$. La sortie du reseau de neurones est parametrisée par la fonction softmax avec $h^3 = 10$ classes

1. Construisez un MLP et choisissez les valeurs de h^1 et h^2 telles que le nombre total de parametres du reseau (en prenant en compte les biais) soit dans l'intervalle [0.5M, 1.0M].
2. Implémentez la propagation avant et arrière (*forward and backward propagation*) du MLP, avec numpy, sans utiliser aucune librairie d'apprentissage profond qui permet de faire la différentiation automatique. Utilisez la structure de classe donnée ici [3].
3. Entraînez le MLP en utilisant l'entropie croisée (*cross entropy*) comme fonction de coût. Minimisez cette fonction de coût pour optimiser les paramètres du model en utilisant *la descente de gradient stochastique* (SGD).

Pour les questions suivantes, précisez l'*architecture du modèle* (le nombre de neurones dans chaque couche, et le nombre total de parametres), la *fonction d'activation* utilisée, le *taux d'apprentissage* (*learning rate*), et la *taille des mini-batch*.

[1] `http://yann.lecun.com/exdb/mnist/` - Utiliser la séparation standard entrainement/validation/test similaire à celle donnée ici[2].

[3] `https://github.com/CW-Huang/IFT6135H19_assignment/blob/master/assignment1.ipynb`

Initialisation [10] Dans cette partie du problème, on considère différentes valeurs initiales pour les poids du reseau. Les biais sont tous égaux à zéro. On considère les initialisations suivantes pour les poids:

- **Zéro**: Tous les poids sont initialisés a zéro.
- **Normale**: Les valeurs des poids sont tirées d'une loi normale standard: $w_{i,j} \sim \mathcal{N}(w_{i,j}; 0, 1)$.
- **Glorot**: Les valeurs des poids sont tirées d'une distribution uniforme: $w^l_{i,j} \sim \mathcal{U}(w^l_{i,j}; -d^l, d^l)$ où $d^l = \sqrt{\frac{6}{h^{l-1}+h^l}}$.

1. Entraînez le modèle pendant 10 époques (*epochs*) [4] en utilisant les méthodes d'initialisation précédentes, et sauvegardez le coût moyen mesuré sur l'ensemble d'entraînement à la fin de chaque époque (vous devriez donc avoir 10 valeurs pour chaque schéma initialisation).
2. Comparez les trois schémas d'initialisation en traçant la courbe des coûts en fonction de la durée d'entraînement (époque). Commentez vos résultats obtenus.

Optimisation des hyper-paramètres [10] Dorénavant, utilisez l'initialisation de Glorot.

1. Trouvez une combinaison d'hyper-paramètres (architecture du réseau, taux d'apprentissage, fonction d'activation, etc.) telle que la précision moyenne (*accuracy*) de classifications sur l'ensemble de validation ($r^{(valid)}$) soit d'au moins 97%.
2. Reportez les hyper-paramètres que vous avez testés, et les $r^{(valid)}$ correspondants.

Validation des gradients en utilisant les différences finies [15] L'approximation par différences finies de la dérivée d'une fonction scalaire $x \in \mathbb{R} \mapsto f(x) \in \mathbb{R}$, avec une précision ϵ, est définie par $\frac{f(x+\epsilon)-f(x-\epsilon)}{2\epsilon}$. Considérez les poids de la première couche du MLP que vous avez construit a la première question, comme un vector $\theta = (\theta_1, \ldots, \theta_m)$. On s'intéresse à l'approximation du gradient de la fonction coût L, évaluée en utilisant **une seule** image de l'ensemble d'entraînement, par rapport à $\theta_{1:p}$, les $p = \min(10, m)$ premiers éléments de θ, en utilisant les différences finies.

1. Évaluez l'approximation des gradients par différences finies $\nabla^N \in \mathbb{R}^p$ en utilisant $\epsilon = \frac{1}{N}$ pour différentes valeurs de N

$$\nabla^N_i = \frac{L(\theta_1, \ldots, \theta_{i-1}, \theta_i + \epsilon, \theta_{i+1}, \ldots, \theta_p) - L(\theta_1, \ldots, \theta_{i-1}, \theta_i - \epsilon, \theta_{i+1}, \ldots, \theta_p)}{2\epsilon}$$

   Utilisez au moins 5 valeurs de N parmi $\{k10^i : \ i \in \{0, \ldots, 5\}, \ k \in \{1, 5\}\}$. Tracez la courbe de la différence maximale entre le vrai gradient et l'approximation par différences finies en fonction de N.
2. Tracez la courbe de la différence maximale entre le vrai gradient et l'approximation par différences finies ($\max_{1 \leq i \leq p} |\nabla^N_i - \frac{\partial L}{\partial \theta_i}|$) en fonction de N. Commentez le résultat.

[4] une époque correspond à un passage sur la totalitée de l'ensemble d'entraînement

Problème 2

Réseaux neuronaux à convolution: Plusieurs techniques correspondent à l'incorporation de connaissances à priori sur la structure des données dans la paramétrisation du modèle. Par exemple, la convolution est adéquate pour le traitement d'images.

Instructions: [25] Dans cette partie du devoir, nous allons entraîner un réseau neuronal à convolution (CNN) sur l'ensemble de donnée MNIST pendant 10 époques. Vous pouvez utiliser une librairie d'apprentissage profond comme Pytorch ou Tensorflow. Tracez les erreurs sur les ensembles d'entraînement et de validation à la fin de chaque époque.

1. Trouvez une architecture CNN avec environ le même nombre de paramètres que le MLP entraîné au Problème 1. Décrivez l'architecture.
2. Comparez les performances du CNN à celles obtenues par le MLP. Commentez vos observations.

Vous pouvez vous inspirer de l'architecture utilisée ici[5].

Problème 3

Dogs vs. Cats est une competition InclassKaggle pour la classification d'images. En utilisant ce que vous avez appris en cours et dans les problèmes précédents, essayez d'entraîner le meilleur classifieur CNN (réseau de neurones à convolution) pour cette tâche. Utilisez le lien suivant[6] pour télécharger les données et accéder à la compétition. Assurez-vous de bien lire la description et les règles.

- Les données sont déjà traitées et prêtes à être utilisées, mais vous pouvez pré-traiter plus vos données (*preprocessing*) si vous le souhaitez. Vous pouvez par exemple faire votre propre recadrage (*cropping*). Toute étape de traitement supplémentaire doit être mentionnée dans votre rapport. Vous ne pouvez pas utiliser de sources de données externes, mais vous pouvez utiliser les techniques d'augmentation de données usuelles (assurez-vous de les mentionner dans votre rapport). Vous êtes responsables de la division de l'ensemble d'entraînement en un ensemble d'entrainement et de validation.
- En plus de vos soumissions Kaggle, on vous demande de soumettre votre code sur Gradescope, avec votre rapport. Nous allons faire tourner votre code et nous assurer qu'on peut recréer votre soumission.

[5] `https://github.com/MaximumEntropy/welcome_tutorials/tree/pytorch/pytorch`
[6] `https://www.kaggle.com/c/ift6135h19`

- Vous ne pouvez pas utiliser de techniques pas encore introduites en cours, directement des librairies d'apprentissage profond, comme les couches BatchNorm/WeightNorm/LayerNorm, les techniques de régularisation (comme Dropout), ou les optimiseurs comme ADAM ou SGD avec moment. **sauf si vous les implémentez vous-mêmes**. Voici la liste de ce que vous pouvez utiliser de votre librarie d'apprentissage profond favorite (Pytorch par exemple):
  - Couches conv/pool/linéaires Conv/Pool/Linear layers
  - N'importe quelle fonction d'activation
  - Descent de gradient stochastique basique (sans moment et pas de Adam)
  - Entropie croisée comme fonction de coût (en particulier, pas de régularisation comme la méthode de dégradation des pondérations (*weight decay*)
  - Techniques d'augmentation des donnees

  Si vous utilisez autre chose, vous devez l'implémenter vous-mêmes en utilisant numpy par exemple.
- Ce problème à pour but de vous donner une chance de jouer avec les méthodes vues en classe et les appliquer a un problème de classification réel. Montrez toutes les traces de ce que vous avez essayé (même ce qui n'a pas donné des scores élevés).
- Votre note sera une combinaison du score de précision (*accuracy*) obtenu sur la partie privée du classement (plus de détails sur Kaggle), et la qualité de votre rapport. Votre rapport doit inclure les réponses aux questions suivantes:

Important: La date limite pour la soumission Kaggle est 1 jour avant celle du devoir.

1. [20 score/5 description] Décrivez l'architecture (nombre de couches, tailles des filtres de convolutions, couches de *pooling*, etc.). Reportez le nombre total de paramètres. Vous pouvez vous inspirer de certaines architectures récentes, comme le réseau VGG, pour améliorer les performances.
2. [15] Tracez les courbes de l'erreur sur l'ensemble d'entraînement et l'ensemble de validation, en plus des courbes de la fonction coût sur les deux ensembles. Commentez. Quelles techniques (que vous n'avez pas implémentées) pourraient être utiles pour améliorer les performances du modèle. Commentez les résultats obtenus sur l'ensemble de validation. Est-ce qu'ils se comparent à ceux obtenus sur l'ensemble de test (que vous pouvez obtenir sur Kaggle seulement).
3. [15] Comparez différentes configurations d'hyperparamètres. Reportez les performances finales sur l'ensemble de validation. En plus des resultats quantitatifs, inclure des analyses visuelles, comme les *feature maps* ou les filtres de convolution. Vous pouvez aussi montrer des exemples d'images qui (a) sont mal classifiées avec grande confidence et (b) résultent en des probabilités d'à peu près 0.5 d'appartenir aux deux classes. Expliquez vos observations, et/ou suggérez des possibles pistes d'amélioration.