IFT 780 : Travail pratique 1

Réseaux de neurones pleinement connectés

Dans ce devoir, vous devez implanter des réseaux de neurones pleinement connectés. Vous travaillerez en premier avec un réseau linéaire (sans couche cachée) puis un réseau à deux couches cachées. Les différentes parties de ce devoir figurent dans des `ipython notebook` dont certains contiennent des questions théoriques auxquelles vous devez répondre. Les réponses aux questions doivent être dans les fichier ipynb et en **format markdown** (voir https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet pour plus de détails sur le standard markdown).

Pour ce travail, vous devez vous créer un environnement virtuel python suivant les recommandations disponibles sur la page web du cours. Vous devez ensuite exécuter dans un terminal les lignes de commande suivantes :

```
$ cd datasets/

$ ./get_datasets.sh
```

afin de télécharger la base de données **CIFAR10**. Par la suite, les fichiers ipynb vous indiqueront les tâches à effectuer. Vous trouverez le code à rédiger dans les fichiers python (.py) aux endroits indiqués par des **TODO**.

Les objectifs de ce devoir sont

1) Comprendre le fonctionnement d'un réseau de neurones simple (entraînement et prédiction)
2) Comprendre l'utilisation de données de validation pour ajuster des hyperparamètres.
3) Vectoriser du code afin de manipuler des « mini-batchs » de données sans boucle for.
4) Implanter un classificateur multi-classes de type SVM
5) Implanter un classificateur multi-classes utilisant un Softmax
6) Implanter un réseau de neurones multi-classes à N couches cachées

À faire :

1. **[3 points]** Implanter et tester un classifier linéaire de type SVM avec la fonction de perte « *Hinge Loss* » de type **one-vs-one (voir** `notebook tp1_hinge.ipynb`)

2. **[3 points]** Implanter et tester un réseau logistique linéaire ayant un *Softmax* et une entropie croisée en sortie (voir `notebook tp1_softmax.ipynb`).

3. **[0 points]** Voyez comment créer un simple réseau de neurones multicouches, comment effectuer une *propagation avant*, une *propagation arrière*, effectuer une *descente de gradient* et l'utilisation d'une « *cache* ». Bien qu'aucun point ne soit attribué à ce `notebook`, il est primordial d'en comprendre le contenu avant de vous attaquer au 4^{e} `notebook`. (voir `notebook tp1_simple_neural_net.ipynb`).

4. **[4 points]** Le but de ce `notebook` est d'enchâsser dans des **classes** les différents éléments vus au point 3.

   NOTE : le code de la propagation avant et de la rétropropagation de la couche dense et du calcul de la perte doit être **vectorisé**. Donc… attention aux boucles for! (voir `notebook tp1_neural_net.ipynb`).