

Intelligence Bio Inspirée

Réseaux de neurones

Mathieu Lefort

15/19 & 24/26 octobre 2018

Ce TP est à faire en binôme. Vous rendrez un compte-rendu incluant votre code et vos réponses aux questions à la fin de chacune des 2 séances de TPs et la version finale pour le 04 novembre. Pensez à indiquer vos noms dans le(s) fichier(s) déposé(s) sur Claroline.

1 Objectif

L'objectif de ce projet est d'implémenter l'algorithme de perceptron multi-couches vu en cours et de comprendre son fonctionnement.

2 Logiciel

Vous utiliserez le framework PyTorch. L'installation se fait de la manière suivante :

```
pip install --user http://download.pytorch.org/whl/cpu/torch-0.4.1-cp27-cp27mu-linux_x86_64.whl
pip install --user torchvision
```


Pour vérifier l'installation, lancez une console python et tapez l'instruction `import torch`

Si vous souhaitez faire l'installation sur votre ordinateur personnel, l'option `--user` n'est pas nécessaire. Il faut également installer python, pip, numpy et matplotlib. Si vous souhaitez utiliser python 3.x cela nécessite quelques petits ajustements du code fourni (le TP est prévu pour python 2.7).

Vous trouverez une liste des fonctions disponibles ici : <https://pytorch.org/docs/stable/index.html> (ce sont surtout les packages `torch` et `torch.Tensor` qui vous intéressent). Quelques exemples introductifs sont disponibles ici : http://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html (le premier chapitre est à lire, le second chapitre vous servira pour la partie 3, les chapitres 3 et 4 vous seront utiles pour les questions optionnelles). Vous pouvez aussi regarder ce micro tutoriel (pour les parties 3 et 4) : http://pytorch.org/tutorials/beginner/pytorch_with_examples.html

3 Données

Vous avez à votre disposition le jeu de données MNIST qui contient des images (de taille 28×28) de chiffres

manuscrits (par exemple une des images de 5 de la base : ). La base de données est structurée comme suit : ((tableau_image_apprentissage, tableau_label_apprentissage), (tableau_image_test, tableau_label_test)). Les images sont stockées sous la forme d'un vecteur de 28×28 valeurs¹ et les labels sont sous la forme d'un codage tabulaire (par exemple si le chiffre représenté sur l'image est un 5, son label sera : $[0, 0, 0, 0, 0, 1, 0, 0, 0, 0]$).

L'entrée de chaque réseau aura donc 784 entrées (28×28 pixels) et 10 sorties (la $i^{\text{ème}}$ sortie représentant si le réseau reconnaît le chiffre i dans l'image).

4 Partie 1 : Perceptron

Dans cette partie vous implémenterez l'algorithme du perceptron (**vous devez implémenter les équations explicitement et n'avez donc pas le droit d'utiliser le package `pytorch.autograd` ni `pytorch.nn`**). Pour rappel l'algorithme est le suivant (**penser à bien rajouter une entrée supplémentaire à 1 pour chaque neurone comme biais!**) :

Pendant un certain nombre de pas de temps :

1. La fonction `flatten()` vous permet d'«écraser» ce vecteur pour en avoir un de 784 (28×28) valeurs. Vous pouvez aussi regarder la fonction `reshape` qui vous permet de redimensionner à la taille de votre choix.

1. Choisir une entrée $x = \{x_j\}_j$ dans la base d'apprentissage (en pratique il faut préalablement permuter la base pour présenter chaque exemple une fois avant de recommencer la présentation des exemples, voir le fichier `lecture_data.py` où cela est fait)
2. Calculer l'activité $y_i = \sum_j w_{ij} x_j$ de chaque neurone i
3. Modifier chaque poids $\Delta w_{ij} = \eta x_j (t_i - y_i)$ avec t_i la $i^{\text{ème}}$ sortie souhaitée

Testez votre algorithme sur la base MNIST et notez la performance sur la base de test. Testez différentes valeurs des hyper-paramètres (nombre de pas de temps, η , poids initiaux) et discutez des éventuels effets.

5 Partie 2 : *Shallow network*

Dans cette partie vous implémenterez l'algorithme du perceptron multi-couches avec une couche cachée et une sortie linéaire (**vous devez implémenter les équations explicitement et n'avez donc pas le droit d'utiliser le package `pytorch.autograd` ni `pytorch.nn`**). Pour rappel l'algorithme est le suivant (**penser à bien rajouter une entrée supplémentaire à 1 pour chaque neurone comme biais**) :

Pendant un certain nombre de pas de temps :

1. Choisir une entrée $x = \{x_j\}_j$ dans la base d'apprentissage (en pratique il faut toujours préalablement permuter la base pour présenter chaque exemple une fois avant de recommencer)
2. Propagation de l'activité :
 - Calculer l'activité $y_i^{(1)} = \frac{1}{1 + e^{-\sum_j w_{ij}^{(1)} x_j}}$ de chaque neurone i de la couche cachée
 - Calculer l'activité $y_i^{(2)} = \sum_j w_{ij}^{(2)} y_j^{(1)}$ de chaque neurone i de la couche de sortie
3. Rétro-propagation du gradient :
 - Pour chaque neurone i de la couche de sortie, calculer l'erreur : $\delta_i^{(2)} = t_i - y_i^{(2)}$ avec t_i la $i^{\text{ème}}$ sortie souhaitée
 - Rétro-propager l'erreur à chaque neurone i de la couche cachée $\delta_i^{(1)} = y_i^{(1)}(1 - y_i^{(1)}) \sum_j \delta_j^{(2)} w_{ji}^{(2)}$
4. Modifier chaque poids $\Delta w_{ij}^{(l)} = \eta \delta_i^{(l)} y_j^{(l-1)}$ avec $y_j^{(0)} = x_j$

Testez votre algorithme sur la base MNIST et notez la performance sur la base de test. Quelles sont les influences des différents hyper-paramètres (nombre de pas de temps, η , poids initiaux, nombre de neurones de la couche cachée) ?

6 Partie 3 : *Deep network*

Dans cette partie vous implémenterez l'algorithme du perceptron multi-couches avec plusieurs couches cachées (nombre à faire varier comme un hyper-paramètre) et une sortie linéaire. PyTorch fournit un outil de différenciation automatique (package `torch.autograd`) permettant de calculer le gradient pour chaque **Variable** ayant déclaré l'option `requires_grad=True` (ce qui vous dispensera d'implémenter la rétropropagation du gradient). Il suffit pour cela de simplement appeler la fonction `backward()` sur la variable de coût (ici l'erreur quadratique) qui va calculer les gradients qui seront alors disponibles dans la variable `grad` de chaque variable concernée (et qu'il faudra remettre à 0 une fois utilisée pour modifier les poids). NB : vous pouvez vérifier que vos implémentations sont cohérentes en refaisant la partie 2 avec la différenciation automatique.

Testez un réseau profond sur la base MNIST et notez la performance sur la base de test. Quelles sont les éventuelles influences des différents hyper-paramètres (nombre de pas de temps, η , poids initiaux, nombre de couches cachées, nombre de neurones par couche cachée) ?

7 Partie 4 : Pour aller plus loin (optionnel)

Si vous avez le temps et l'envie vous pouvez traiter certaines des questions suivantes :

Question : Testez d'autres fonctions d'activation pour les neurones : `tanh` et `ReLU` en particulier

Question : Utilisez une méthode de gradient plus efficace que celle vue en cours comme Adam ou Adagrad (utilisez le package `torch.optim`) et testez des minibatches.

Question : Pour le traitement d'image, on utilise généralement des réseaux à convolutions. Implémentez en un (utilisez le package `torch.nn`) en essayant de trouver des hyperparamètres qui donnent une bonne performance.