

# Introduction à l'apprentissage profond

Mathieu Lefort & Alexandre Devillers

2 et 9 décembre 2021

Ce TP peut être fait en binôme (vous indiquerez alors la répartition des tâches que vous avez adopté). Vous rendrez un compte-rendu incluant votre code commenté et vos réponses aux questions à la fin de chacune des 2 séances de TPs et la version finale pour le 9 janvier. Pensez à indiquer votre(s) nom(s) dans le fichier déposé sur Claroline.

## 1 Objectif

L'objectif de ce projet est d'implémenter l'algorithme du perceptron multi-couches vu en cours, de comprendre son fonctionnement et de prendre en main le framework PyTorch.

## 2 Installation


Vous devez installer python, pytorch et numpy (et matplotlib si vous voulez faire des affichages). Pour l'installation de PyTorch regardez <https://pytorch.org/get-started/locally/>. Pour vérifier l'installation, lancez une console python et tapez l'instruction `import torch`.

## 3 PyTorch

Vous trouverez une liste des fonctions disponibles ici : <https://pytorch.org/docs/stable/index.html>. Regardez ce micro tutoriel : [http://pytorch.org/tutorials/beginner/pytorch\\_with\\_examples.html](http://pytorch.org/tutorials/beginner/pytorch_with_examples.html). Quelques exemples introductifs sont également disponibles ici : [http://pytorch.org/tutorials/beginner/deep\\_learning\\_60min\\_blitz.html](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).

## 4 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 valeurs et les labels sont sous la forme d'un codage *one-hot* (i.e. 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<sup>ème</sup> sortie représentant à quel point le réseau reconnaît le chiffre i dans l'image).

## 5 Partie 1 : Perceptron

Vous avez à votre disposition 2 fichiers d'implémentation du (même) perceptron :

- PERCEPTRON\_PYTORCH.PY qui utilise uniquement les tenseurs
- PERCEPTRON\_PYTORCH\_DATA\_AUTO\_LAYER\_OPTIM.PY qui utilise la plupart des outils de Pytorch (les *dataloaders*, la différenciation automatique, les couches et les optimiseurs)

Regardez les fichiers fournis.

- Indiquer la taille de chaque tenseur dans le premier fichier. Expliquer.
- Quantifier l'impact des différents hyperparamètres (en particulier  $\eta$  et les poids initiaux) sur les performances.

## 6 Partie 2 : *Shallow network*

Dans cette partie vous implémenterez l'algorithme du perceptron multi-couches avec une seule couche cachée et une sortie linéaire. 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  $y_j^{(0)} = x_j$

De manière similaire au perceptron (et en vous appuyant sur les fichiers fournis),

- implémenter une version en n'utilisant que les tenseurs et une utilisant les outils de Pytorch.
- Quantifier l'influence des différents hyper-paramètres (en particulier  $\eta$ , les poids initiaux et le nombre de neurones de la couche cachée).

## 7 Partie 3 : *Deep network*

Maintenant que vous maîtrisez Pytorch, vous allez pouvoir l'utiliser pour tester des modèles plus complexes comme un réseau profond.

- Implémenter un réseau de neurones profond (i.e. avec au moins 2 couches cachées) en utilisant les outils fournis par Pytorch.
- Quantifier l'influence des différents hyper-paramètres (en particulier  $\eta$ , les poids initiaux, le nombre de couches cachées et de neurones par couche cachée et la taille des mini batches).

## 8 Partie 4 : Pour aller plus loin (optionnel)

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

- Tester d'autres fonctions d'activation pour les neurones (tanh ou ReLU en particulier).
- Étudier d'autres méthodes d'initialisation des poids (poids gaussien, initialisation Xavier par exemple)
- Utiliser une méthode de gradient plus efficace que celle vue en cours comme Adam ou Adagrad.
- Tester d'autres fonctions de coût (cross entropie par exemple).
- Pour le traitement d'image, on utilise généralement des réseaux à convolutions (vus dans la deuxième partie d'IBI). Implémenter en un en essayant de trouver des hyperparamètres qui donnent une bonne performance.