Fouille de Données Massives
TD - Réseaux de Neurones et H_20

M2 Informatique - SISE (2022-2023)

Guillaume Metzler

Institut de Communication (ICOM)
Université de Lyon, Université Lumière Lyon 2
Laboratoire ERIC UR 3083, Lyon, France

guillaume.metzler@univ-lyon2.fr

Abstract

Ce présent TP a pour but de vous faire implémenter vos propres algorithmes de Machine Learning et plus spécifiquement, des algorithmes de Deep Learning avec l'apprentissage de modèles basés sur l'utilisation de réseaux de neurones.

Il se décompose en trois parties, une première partie commune à l'ensemble des utilisateurs et qui vous proposent de découvrir la librairie *h2o* sous R (en fait il n'y que deux lignes sous R à faire et le reste se fera à l'aide d'une interface).
La deuxième partie vous propose de vous affranchir de cette interface et d'implémenter directement les méthodes sous R ou Python.
La troisième partie vous propose de découvrir une autre librairie dédiée spécifiquement à l'implémentation des réseaux de neurones : *Keras*.

1 Introduction

Lorsque les jeux de données sont trop volumineux pour être analysés avec une seule machine, nous avons qu'il pouvait être intéressant d'utiliser des architectures complexes afin de distribuer les calculs (Hadoop, Spark, ...). Mettre en place ce type d'outils dans un cadre limité est très contraignant, car nécessite du temps et des efforts matériels.

Nous allons voir comment contourner cette barrière technique en utilisant des alternatives mises à notre disposition : *h2o* et *Keras*, qui sont disponibles sous R et Python, pour pouvoir optimiser nos calculs. La première solution est très générale et nous permettra d'implémenter n'importe quel algorithme que cela soit du Deep learning ou du gradient boosting par exemple (avec ses limites). La deuxième solution sera dédiée spécifiquement au Deep Learning.

Nous travaillerons avec deux jeux de données très utilisés dans la communauté Machine Learning : *MNIST* et *FASHION MNIST*

2 Utilisation de l'interface *h2o*

h2o est une palteforme de Machine Learning vous permettant d'exécuter des algorithmes d'apprentissage dans plusieurs sortes d'environnements : disitrubution des calculs sur un ensemble de coeurs - processeurs ou encore sur différentes machines (clusters- serveurs). Elle a la particularité de rendre accessible et implémentable la plupart des algorithmes de Machine Learning comme le *Gradient Boosting, les méthodes Deep Learning* ou encore l'algorithme *K-means*. Pour l'utiliser, nous pouvons faire appel à l'interface R à travers le package *h2o*, ou avec une interface web, c'est cette deuxième solution qui va nous intéresser pour le moment.

2.1 Installation

Il vous faudra simplement suivre les questions présentées ci-dessous :

1. Commencez par installer le package *h2o* sous R . Faites bien attention à avoir Java sur votre ordinateur.
2. Lancez ensuite les commandes suivantes

```
# Chargement de la librairie
library(h2o)
h2oServer = h2o.init(nthreads = -1, ip = 'localhost', port = 54321)
```

Selon vous, à quoi servent les arguments dans la fonction ?

3. Utilisez les éléments fournis dans la console afin d'ouvrir une interface web, en local. Cette interface sera votre espace de travail et ne nécessitera plus d'utiliser pour le moment.
   Si vous ne trouvez pas les éléments nécessaires en sortie de votre console, essayez l'adresse suivante

   http://localhost:54321

4. Parcourez cette interface et regardez les différents menus. Répondez ensuite aux questions suivantes à l'aide de la documentation sur *h2o* :

   - Citez trois exemples de modèles implémentés par *h2o*
   - Nommez votre flow : "Deep Learning MNIST"
   - Quels sont les formats de données acceptés par la plateforme ?

   Passons maintenant au chargement des données

2.2 Chargement des données

A nouveau, on vous propose de suivre les étapes suivantes pas à pas.

1. Commencez par charger les données MNIST disponible dans les ressources de la séance

   Chargez le fichier *MNIST.csv*

   Ce jeu de données est historiquement le jeu de données en Deep Learning, les résultats obtenus sur ce jeu de données sont proches de 100% par les algorithmes actuels. Regardez les données :

   - Que représentent-elles ?
   - Que représente une ligne ?
   - Que représente une colonne ?
   - Quelle est la taille d'une image ?
   - Que représente la colonne *label* ?
   - Que représente un élément d'un(e) tableau/matrice ?

2. Importez les données et arsez les (vous aurez une petite icône sur la partie inférieure de la page qui vous permettra de faire cela).

3. Observez le résumé des premières colonnes et expliquez pourquoi certains *pixels* ont une valeur toujours égale à 0.

4. Dans la cellule *Job*, cliquez sur *View*, puis *Split* afin de choisir un ensemble d'entraînement représentant 75% du jeu de données et les 25% restants constitueront votre ensemble de validation. Fixez également une graine (*seed*) égales à 10.

   Nous avons maintenant tous les éléments et nous pouvons préparer notre modèle.

2.3 Création d'un réseau de neurones

Il n'y a qu'à suivre les démarches ci-dessous pour créer votre premier modèle :

1. Avec l'onglet *Model*, demandez à construire un réseau de neurone (*i.e.* un modèle du type *Deep Learning*).

2. Dans la section paramètres (pas avancés), donner le jeu de données à l'entraînement du modèle, puis celui de la validation.

3. Configurez un réseau de neurones comprenant 4 couches avec respectivement 4, 3 , 3 et 5 neurones

4. Observez l'erreur et expliquez pourquoi l'erreur vous semble si élevée.

5. Testez d'autres paramètres du modèles pour améliorer ce score? N'hésitez pas à ajouter d'autres hyper-paramètres, à modifier les fonctions d'activations, changer le nombre d'epochs ou encore à modifier la structure de votre réseau. C'est une phase exploratoire, testez les différents paramètres et observez.
   Attention : si vous voulez observer l'effet d'un paramètre, ne les changez que un par un !

6. Comparez vos résultats avec d'autres modèles disponibles dans *h2o*.

3 Utilisation du package *h2o*

On va maintenant regarder comment s'affranchir de cette interface web. L'objectif étant que vous vous passiez par la suite de cette interface et vous puissez utiliser l'outil dans n'importe quel contexte, *i.e.* le rendre plus flexible en terme d'utilisation.

Nous séparons maintenant les sections en deux, une première section dédiée aux utilisateurs de  et l'autre section pour les utilisateurs de Python.

3.1 Pour les utilisateurs de R

Vous verrez que l'exercice est relativement simple car il s'agit essentiellement de traduire ce qui figure sur l'interface web pour cela on commencera par exécuter les commandes suivantes :

1. Importez les données MNIST sous R .
2. Splittez votre jeu de données en deux ensembles selon la même règle que dans la section précédente.
3. Exécutez la commande suivante (création de votre espace de travail)

```
# Chargement de la librairie
library(h2o)
h2oServer = h2o.init(nthreads = -1)
```

4. Convertissez vos jeux de données au bon format

```
train.h2o<-as.h2o(train)
valid.h2o<-as.h2o(valid)
# On pensera aussi a convertir la variable réponse ou variable à predire
# sous forme de facteur dans les deux ensembles
...
...
```

5. Etudiez ensuite le fonctionnement de la fonction *h2o.deeplearning* pour implémenter un modèle de réseau de neurones que vous appellerez *dlmodel* avec les caractéristiques suivantes :
   - un réseau avec trois couches de respectivement 40, 20 puis 10 neurones
   - qu'il fasse 50 epochs
   - utilise du "dropout" et la fonction "Tanh" comme fonction d'activation
   - que les batch soient équilibrés (en terme d'exemples de chaque classe) avec une taille minimale égale à 50
   - le pas d'apprentissage sera égale à 0.01
   - vous introduirez également un terme de régularisation $L2$
6. Mesurez les *performances* de votre modèle sur votre ensemble de validation et donnez la *matrice de confusion* et également l'AUC

7. Exécutez la commande suivante et observez les résultats

```
plot(dlmodel)
```

8. Essayez de faire cela avec un autre jeu de données disponibles sous R (comme Iris) et trouvez le meilleur modèle.

3.2 Pour les utilisateurs de Python

En cours de construction ...

4 Utilisation de la librairie *Keras*

Keras est une bibliothèque open source écrite en Python, donc les utilisateurs de R devront penser à installer Python sur leur ordinateur. *Keras* permet d'implémenter très facilement des algorithmes de *Deep Learning* mais aussi d'autres algorithmes de *Machine Learning*, mais on se contentera ici de l'implémentation de réseaux de neurones.
Keras a l'avantage d'être un peu plus flexible que la librairie *h2o*.

4.1 Pour les utilisateurs de R

Vous pouvez retrouver toutes les étapes pour installer *Keras* et l'utiliser à l'aide du logiciel Rau lien suivant :

Installation Keras

Vous pouvez également consulter les liens suivants pour de plus amples explications :

Explications Keras

Dans cette partie on va considérer le jeu de données *CIFAR10*, *i.e.* votre code devra commencer de la façon suivante :

```
# Vous aurez peut-etre d'installer le package devtools, sinon essayer
# ce qui figure dans le lien mentionne ci-dessous
devtools::install_github("rstudio/keras")
library(keras)
install_tensorflow()
```

```
# Jeu de donnees, choisir CIFAR10, vous pourrez tester les autres chez vous
mnist<-dataset_mnist()
cifar10<-dataset_cifar10()
imdb<-dataset_imdb()
```

Regardez ensuite comment implémenter un réseau de neurones à l'aide de *Keras* et implémentez des modèles de *Deep Learning* vous permettant d'obtenir de bonnes performances sur le jeu de données *CIFAR10*

Remarque : le jeu de données CIFAR10 est un jeu de données de $60,000$ exemples (images) de taille 32×32 avec 10 classes différentes (avion, auto, oiseau, grenouille, chien, char cheval, bateau, ...)

Vous pourrez ensuite tester des réseaux de neurones convolutionnels pour effectuer de l'extraction de features sur le même jeu de données en vous inspirant du code ci-dessous :

```
model <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(3,3), activation = "relu",
                input_shape = c(32,32,3)) %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = "relu")
```

La première ligne vous permet d'initialiser votre modèle, ensuite vous allez ajouter des couches à votre réseau de neurones ici de deux types *convolutions* et *max-pooling*.
La première couche doit nécessairement comporter la taille de vos images en entrée (paramètres *input shape*.
• La fonction *conv2d* prend trois paramètres paramètres en entrée : la dimension des données en sorties (*filters*), la taille du masque (*kernel size*) et la fonction d'activation (*activation*).
• La fonction *max pooling* qui prend comme paramètre la taille du masque (*poolsize*).

On utilisera ensuite la commande suivante pour passer du format **matriciel** de nos données au format **vectoriel**

```
layer_flatten()
```

4.2 Pour les utilisateurs de Python

Je vous propose ici deux possibilités, le but étant de vous faire implémenter votre propre réseau de neurones en travaillant sur des données images, cela peut se faire sous forme de deux travaux :

- première possibilité, travailler sur la classification binaire d'image, le but est de chercher à prédire s'il s'agit d'une image de chien ou de chat, pour cela, on pourra suivre le tutoriel *Keras* que l'on trouvera à la page suivante :

  Tutoriel Keras

  conda install tensorflow-gpu==2.3

  donc il faut avoir "pip" d'installer sur son ordinateur, cela ne fonctionne pas avec conda.**Il vous faudra nécessaire la version 2.3 de tensorflow pour pouvoir suivre ce tutoriel, que vous pourrez installer à l'aide de la fonction :**

  conda install tensorflow-gpu==2.3

  donc il faut avoir "pip" d'installer sur son ordinateur, cela ne fonctionne pas avec conda.

- **si vous ne pouvez pas installer la dernière version de tensorflow et donc, ne pouvez pas effectuer le travail précédent (ce qui est mon cas), vous pouvez utiliser le bout de code python ci-dessous afin de procéder à de *l'extraction de feature*. Il s'agit d'extraire de l'information à partir d'images pour obtenir une représentation vectorielle, cela se fait à partir de réseaux de neurones convolutionnel.**

  Utilisez le fichier *CNN.py*

  Cette deuxième option ne nécessite pas de version de Tensorflow particulière. Vous pourrez ensuite utiliser ces nouvelles *features* pour apprendre un modèle de Machine Learning standard, vous pourrez par exemple vous aidez du code disponible à l'adresse suivante pour vous aider à implémenter votre propre réseau de neurones à partir des données précédemment générées.

  Utilisez le fichier *NN.py*

5 Images Satellites

Pour cette dernière partie, on considérera le jeu de données se trouvant à l'adresse suivante :

Jeux de données satellites

Chaque ligne de ce jeu de données représente une image satellite de taille 3×3 pixels, ces images sont décrites par 4 bandes spectrales, ce qui nous donne un ensemble de 36 descripteurs pour chaque exemple. La dernière colonne vous donne la variable à prédire, *i.e.* le type de sol correspondant à l'image. Des informations supplémentaires vous sont donner à l'adresse ci-dessus.

L'objectif est simple : établir un protocole expérimentale et apprendre un modèle en utilisant le jeu de donnée **sat.trn** conduisant aux meilleures performances sur le jeu de données **sat.tst** à l'aide de l'*algorithme* et du *langage* de votre choix.