MNIST ou le *Hello Word* de la classification d'images

Réseaux de neurones profonds

Le 12/12/2019

M2-SDS *Université Paris-Sud*

1.Le jeu de données MNIST

Le jeu de données MNIST est hébergé sur le page web de Yann LeCun. Ce jeu de données sert de référence pour les compétitions en apprentissage où la donnée explicative est sous forme d'image. Commençons par une visualisation du jeu de données et plus précisément les 36 premières images. Pour cela nous allons installer le package keras qui installe à son tour tensorflow. Le bloc de code suivant permet cette visualisation

```
#install.packages("keras", dep=TRUE)
require(keras)
mnist   <- dataset_mnist()
x_train <- mnist$train$x
y_train <- mnist$train$y
x_test  <- mnist$test$x
y_test  <- mnist$test$y

# visualize the digits
par(mfcol=c(6,6))
par(mar=c(0, 0, 3, 0), xaxs='i', yaxs='i')
for (idx in 1:36) {
  im <- x_train[idx,,]
  im <- t(apply(im, 2, rev))
  image(1:28, 1:28, im, col=gray((0:255)/255),
        xaxt='n', main=paste(y_train[idx]))
}
```

2. Implémentation d'un RNA avec une seule couche cachée avec keras

Reprenons le jeu de données après aplatissement mais sans réduction de dimension. Appliquons une normalisation en divisant les intensités des pixels par 255 (intensité maximale d'un pixel).

```
require(keras)
mnist <- dataset_mnist()
x_train <- mnist$train$x
```

```
y_train <- mnist$train$y
x_test <- mnist$test$x
y_test <- mnist$test$y
y_train <- to_categorical(y_train, 10)
y_test <- to_categorical(y_test, 10)
# reshape
x_train <- array_reshape(x_train, c(nrow(x_train), 784))
x_test <- array_reshape(x_test, c(nrow(x_test), 784))
# rescale
x_train <- x_train / 255
x_test <- x_test / 255
```

À l'aide du package keras, entraîner un RNA avec une couche cachée comme suit :

1. Le nombre de neurones (LTU) de la couche cachée est de 784 avec la fonction d'activation relu[1]. De combien de LTU a-t-on besoin à la dernière couche ? préciser sa fonction d'activation.
2. Utiliser la fonction de perte ainsi que la métrique appropriés pour le contexte de la classification. Faire appel à l'algorithme d'optimisation *adam*. Utiliser des mini-lots de données de taille 200 et 10 cycles d'apprentissage. Inclure dans la procédure le jeu de données test comme jeu de données de validation[2]. Préciser la meilleure performance obtenue en terme de taux de bon classement sur les 10 cycles d'apprentissage.

3. Implémentation d'un réseau de neurones convolutifs

Reprendre le jeu de données précédent sans les transformations *array_reshape*[3].

À l'aide du package keras, entraîner un CNN composé des couches suivantes :

1. Une couche de convolution 2d avec 30 filtres de convolution ainsi qu'un noyaux d'un pas de c(5,5) et une fonction d'activation *relu*.
2. Une couche de max-pooling 2d avec un noyau de pooling d'un pas de c(2,2).
3. Une couche de convolution 2d avec 15 filtres de convolution ainsi qu'un noyaux d'un pas de c(3,3) et une fonction d'activation relu.
4. Une couche de max-pooling 2d avec un noyau de pooling d'un pas de c(2,2).
5. Une couche *dropout* avec un taux d'extinction de 30 %.
6. Une couche d'aplatissement flatten pour aplatir les sorties de la couche précédente.
7. Une couche cachée composée de 128 LTU et une fonction d'activation relu.
8. Une couche cachée composée de 50 LTU et une fonction d'activation relu.
9. La dernière couche de sortie composée de 10 LTU et une fonction d'activation softmax.

[1] Il faut absolument consulter l'aide du package keras pour comprendre le principe de fonctionnement et de construction des réseaux de neurones avec keras.

[2] La lecture de ce document: *Optimization algorithms* permet de se familiariser avec le vocabulaire.

[3] Conserver la normalisation des intensités.

Reprendre les mêmes paramètres d'entraînement précisés dans la section précédente.