Feuille 2. : Tensorflow, Keras, premières manipulations

1 Installation

Suivre les étapes décrites dans le page `https://www.tensorflow.org/install/` pour installer `tensorflow` sur votre machine.

Installer également la bibliothèque keras : `pip install keras`.

2 Exemple 1 : Le `Hello the world` de `tensorflow`

Comme premier exemple, nous allons entrainer une régression linéaire. Il s'agit donc d'apprendre des paramètres W et b de manière à obtenir un modèle $Y = W * X + b$.

Pour cela, il faut écrire le code suivant (à discuter au tableau) :

```
import tensorflow as tf
# Model parameters
W = tf.Variable([.3], dtype=tf.float32)
b = tf.Variable([-.3], dtype=tf.float32)
# Model input and output
x = tf.placeholder(tf.float32)
linear_model = W*x + b
y = tf.placeholder(tf.float32)
# loss
loss = tf.reduce_sum(tf.square(linear_model - y)) # sum of the squares
# optimizer
optimizer = tf.train.GradientDescentOptimizer(0.01)
train = optimizer.minimize(loss)
# training data
x_train = [1, 2, 3, 4]
y_train = [0, -1, -2, -3]
# training loop
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init) # reset values to wrong
for i in range(1000):
  sess.run(train, {x: x_train, y: y_train})
# evaluate training accuracy
curr_W, curr_b, curr_loss = sess.run([W, b, loss], {x: x_train, y: y_train})
print("W: %s b: %s loss: %s"%(curr_W, curr_b, curr_loss))
```

3 Exercice 2 : Deep Learning avec keras

Nous allons à présent utiliser le dataset `mnist` pour apprendre à reconnaître tous les nombres de 0 à 9. Pour cela, il faut suivre les étapes suivantes. Les explications du code sont données au tableau :

1. On commence par importer les données du dataset :
```
from keras.datasets import mnist
```
2. On importe les fonctions et les classes dont on aura besoin :
```
import numpy
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import np_utils
```
3. On initialise le générateur de nombres aléatoires avec la même graine :
```
seed = 7
numpy.random.seed(seed)
```
4. A présent, on charge les données :
```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
num_pixels = X_train.shape[1] * X_train.shape[2]
```
5. On "aplatît" les images pour obtenir un vecteur colonne à $28 \times 28 = 784$ valeurs de pixels :
```
X_train =  X_train.reshape(X_train.shape[0],num_pixels).astype('float32')
X_test =  X_test.reshape(X_test.shape[0],num_pixels).astype('float32')
```
6. On normalise les valeurs :
```
X_train = X_train /255
X_test = X_test /255
```
7. On transforme les chiffres de sortie en données catégorielles :
```
y_train =  np_utils.to_categorical(y_train)
y_test =  np_utils.to_categorical(y_test)
num_classes = y_test.shape[1]
```
8. On est à présent en mesure de définir le modèle, i.e., un réseau de neurones :
```
def baseline_model():
model = Sequential()
model.add(Dense(num_pixels, input_dim=num_pixels, kernel_initializer='normal',
activation='relu'))
model.add(Dense(num_classes, kernel_initializer='normal', activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
return model
```
9. Puis de le construire et de l'évaluer :
```
model = baseline_model()
model.summary()
model.fit(X_train, y_train, validation_data=(X_test,y_test), epochs=10,
batch_size=200)
scores = model.evaluate(X_test, y_test)
print("Baseline Error: \%.2f\%\%" \% (100-scores[1]*100))
```