My first steps with Deep Learning: the MNIST case

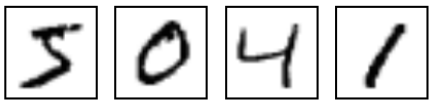
Stéphane Canu
scanu@insa-rouen.fr, asi.insa-rouen.fr\~scanu

October the 10th 2019

The objectives of the lab

The purpose of this notebook is to take its first step in deep learning by reproducing the results given on the MNIST site. In less than 3 minutes, you will build and train a fully connected neural network (NN) performing less than 1.5% error on the MNIST database, and then, in less than 15 minutes, a convolutional neural network performing less than 1% error. We propose to use Keras: The Python Deep Learning library. Why Keras, because its recomeded to beginners by the Deep Learning Framework Power Scores 2018. See also `https://deepsense.ai/keras-or-pytorch/` and `https://github.com/ilkarman/DeepLearningFrameworks/` for a comparizon of different frameworks.

A companion notebook is available here `https://github.com/StephaneCanu/Deep_learning_lecture/blob/master/jupyter_notebooks/TP1_MNIST.ipynb`



Ex. 1 — the MNIST database

1. What is the MNIST database?
   a) What is the nature of the problem addressed by the MNIST data?
   b) What is the nature of the data: how many examples and features?
   c) What is the evaluation criteria used to compare the proposed solutions?
2. What are the best performances reported?
   a) By a fully connected network?
   b) By a convolutional neural network?
   c) Over all (the so-called state of the art)?
3. Check the data
   a) Visualize the first digit of the MNIST training and test data.

```
import keras
import matplotlib.pyplot as plt

from keras.datasets import mnist
from keras.datasets import fashion_mnist
(X_train, y_train), (X_test, y_test) = mnist.load_data()

plt.subplot(1,2,1)
plt.imshow(X_train[1], cmap='gray', interpolation='none')
plt.title("Class {}".format(y_train[1]))
plt.subplot(1,2,2)
plt.imshow(X_test[1], cmap='gray', interpolation='none')
plt.title("Class {}".format(y_test[1]))
plt.show()

x_train.view()
```

   b) Check the labels

```
Y_train.view()
```

Ex. 2 — Neural Networks

1. Install the keras package `https://keras.io/#installation`

```
pip3 install keras
```

a) Normalize the data: convert the gray scale values from integers ranging between 0 to 255 into floating point values ranging between 0 and 1

```
X_train = X_train/255
X_test = X_test/255
x_train = X_train.reshape(60000, 784) # reshape input from (28,28) to 784
x_test = X_test.reshape(10000, 784)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
```

b) Create a feed-forward neural network fully connected with a hidden layer containing 256 hidden neurons with ReLU activation function.

```
from keras.models import Sequential
from keras.layers import Dense

NN = Sequential()
NN.add(Dense(units=300, activation='tanh', input_dim=784))
NN.add(Dense(units=10, activation='softmax'))
```

c) What use the softmax output (why is it relevant to our problem)?

d) Check your neural network architecture

```
NN.summary()
```

e) Compile the model with cross-entropy loss function, stochastic gradient descent optimizer and the accuracy metrics:

```
NN.compile(loss = 'categorical_crossentropy',
optimizer = keras.optimizers.SGD(),
metrics = ['accuracy'])
```

f) Propose alternative choices for the loss function, the optimizer, and the metrics. Why the previous choices are relevant[1].

g) Transform the labels into a 10 dimensional binary vector

```
num_classes = 10;
Y_train = keras.utils.to_categorical(y_train, num_classes)
Y_test = keras.utils.to_categorical(y_test, num_classes)
```

h) Train the neural network during 10 epochs with batches of 128 images and using 20% of the data as a validation set.

```
NN.fit(x_train, Y_train, epochs=10, batch_size =128, validation_split=0.2)
```

i) What is the influence of the batch size on the training?

j) Evaluate the model's performance on the test data

```
score = NN.evaluate(x_test, Y_test)
err = 100*(1-np.array(score))
print('Test loss: %4.3f '%score[0])
print('Test err: %4.2f %%'% err[1])
```

Ex. 3 — Your turn

1. By modifying the fields of the `model` and `fit` objects, propose a fully connected (no convolution) network without *drop out* that obtains less than 1% error on the MNIST data base (that improves upon the neural network of exercice 2). Implement it. *Tips: you can use the MNIST web site to identify a relevant neural networks architecture.*

[1] `https://keras.io/losses/,https://keras.io/optimizers/,https://keras.io/metrics/`