Exercises – Deep Learning

For this set of exercises we will be using Keras [http://keras.io/], a high-level neural networks library, written in Python and capable of running on top of either TensorFlow or Theano. To install Keras, use PyPI with: `sudo pip install keras`

The default backend library in Keras is TensorFlow but it requires an additional installation. For these exercises, change the backend to Theano [http://keras.io/backend/].

1. Neural Network

Open the nn.py and edit only the main() function.

a) Load the MNIST dataset that is included in the Keras library using the loadMNISTdataset() function

```
 x_train, y_train, x_test, y_test = loadMNISTdataset()
```

b) Create a multilayer percetron model and interpret its architecture using the Keras documentation [http://keras.io/layers/core/]:

```
model = Sequential()
model.add(Dense(64, input_dim=28*28, init='uniform', activation='sigmoid'))
model.add(Dense(10, init='uniform', activation='sigmoid'))
model.compile(loss='mse', optimizer='adam', metrics=['accuracy'])
```

c) Train the model

```
history = model.fit(x_train, y_train, batch_size=64, nb_epoch=3, verbose=1,
validation_split=0.1)
```

Plot the accuracy of the training process using the function plotTrainingHistory(). Try different number of epochs (3, 10, 20) and interpret the results.

d) Test the model

```
score = model.evaluate(x_test, y_test)
print("\nTest accuracy: %0.05f" % score[1])
```

Show the images that are erroneously classified using the function showErrors()

2. Convolutional Neural Network

Open the cnn.py and edit only the main() function.

a) Load the MNIST dataset that is included in the Keras library using the loadMNISTdataset() function

```
x_train, y_train, x_test, y_test, input_shape = loadMNISTdataset()
```

b) Create a multilayer percetron model and interpret its architecture architecture using the Keras documentation [http://keras.io/layers/core/]:

```
nb_filters = 32
pool_size = (2, 2)
kernel_size = (3, 3)
model = Sequential()
model.add(Convolution2D(nb_filters, kernel_size[0], kernel_size[1],
                        border_mode='valid',
                        input_shape=input_shape))
```

```
model.add(Activation('relu'))
model.add(Convolution2D(nb_filters, kernel_size[0], kernel_size[1]))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=pool_size))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(10))
model.add(Activation('softmax'))
model.compile(loss='categorical_crossentropy',
              optimizer='adadelta',
              metrics=['accuracy'])
```

c) Train the model (it may take a while…)

```
history = model.fit(x_train, y_train, batch_size=64, nb_epoch=3, verbose=1,
validation_split=0.1)
```

d) Test the model

```
score = model.evaluate(x_test, y_test)
print("\nTest accuracy: %0.05f" % score[1])
```

e) Since training a complex model can take a very long time to train, they can be saved and loaded later. Explore how this can be done [https://keras.io/models/about-keras-models/]

f) It is also possible to use very complex models with pre-trained weights, such as VGG and ResNet. Explore how this can be done [http://keras.io/applications/]