Exercise 2 Report

April 18, 2017

By: Ngo Minh Thang, Student Number: 608820.
Goal: Developing and evaluating a deep learning model.

```
In [1]: import matplotlib.pyplot as plt
        %matplotlib inline
        %load_ext autoreload
        %autoreload 2
```

1 Prepare the data

In this section, I do the following steps:

- Load the MNIST data into numpy arrays
- Flatten the 28x28 images to (1, 784) vector
- Normalize the input pixel values
- One-hot encoding the output classes

```
In [2]: from keras.datasets import mnist

        # download the mnist data
        (X_train, y_train), (X_test, y_test) = mnist.load_data()
```

```
Using TensorFlow backend.
```

```
In [3]: print("The shape of the train data: ", X_train.shape, y_train.shape)
```

```
The shape of the train data:  (60000, 28, 28) (60000,)
```

```
In [4]: print("The shape of the test data: ", X_test.shape, y_test.shape)
```

```
The shape of the test data:  (10000, 28, 28) (10000,)
```

In [5]:
```
# number of pixels in each image
num_pixels = X_train.shape[1]*X_train.shape[2]
num_pixels
```

Out[5]: 784

In [6]:
```
import numpy as np

# reshape 28x28 images to a (1,784) vector
X_train = X_train.reshape(X_train.shape[0], num_pixels)
X_test = X_test.reshape(X_test.shape[0], num_pixels)

print("The new shape of train and test data: ", X_train.shape, X_test.shape)
```

```
The new shape of train and test data:  (60000, 784) (10000, 784)
```

In [7]:
```
from sklearn.preprocessing import StandardScaler

# perform normalization
scaler = StandardScaler().fit(X_train)
scaler.transform(X_train)
scaler.transform(X_test)
```

```
E:\Anaconda\envs\deeplearning\lib\site-packages\sklearn\utils\validation.py:429: DataConversionW
  warnings.warn(msg, _DataConversionWarning)
```

Out[7]:
```
array([[ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       ...,
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.]])
```

In [8]:
```
from keras.utils import np_utils

# one hot encode the outputs
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
```

In [9]:
```
# number of unique labels
num_classes = y_train.shape[1]
num_classes
```

Out[9]: 10

2 Deep learning

I make a sequential model in Keras. There are 3 layers:

- The input layer: It take as inputs array of shape (, 784) and output arrays of the same shape. The activation function is the rectifier function.
- The hidden layer: Its inputs are the outputs of the first layer but it output arrays of shape (, 392) (half the number of pixels). Rectifier function is also used. I also apply a dropout, which help prevent overfitting by randomly turn off 20% of the neurons during each update in the training time.
- The output layer: uses a softmax activation function to turn the outputs into probability-like values. It has 10 neurons corresponding to the 10 classes.

```
In [10]: from keras.models import Sequential
         from keras.layers import Dense, Dropout

         # define the model
         model = Sequential()

         # the input layer
         model.add(Dense(units=num_pixels, input_shape=(num_pixels, ), activation='relu'))

         # the hidden layer
         model.add(Dense(units=num_pixels//2, activation='relu'))
         model.add(Dropout(0.2))

         # the output layer
         model.add(Dense(units=num_classes, activation='softmax'))

In [11]: model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 784)               615440
_________________________________________________________________
dense_2 (Dense)              (None, 392)               307720
_________________________________________________________________
dropout_1 (Dropout)          (None, 392)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                3930
=================================================================
Total params: 927,090.0
Trainable params: 927,090.0
Non-trainable params: 0.0
_________________________________________________________________
```

For compiling the model, I use:

- Optimizer: Stochastic Gradient Descent
- Loss function: Mean Absolute Error
- The metric for evaluate the model: Classification Accuracy

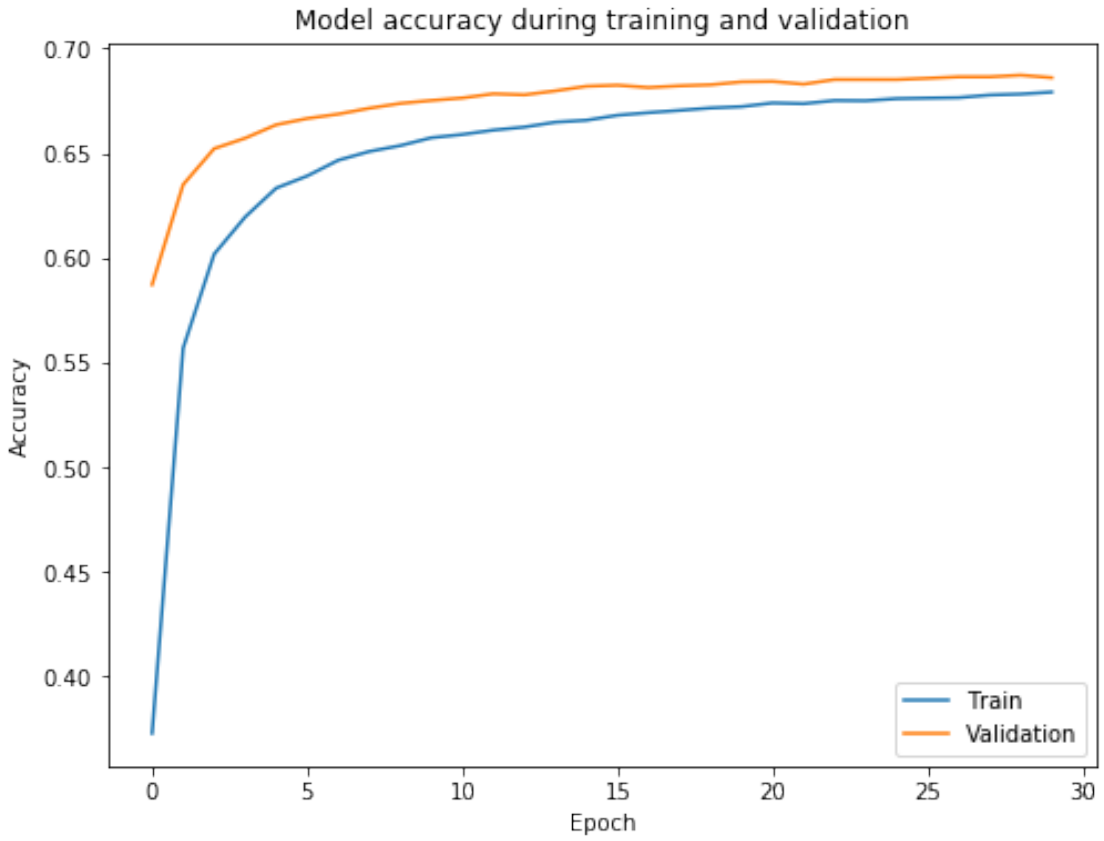
```
In [12]: # compile the model
         model.compile(optimizer='SGD', loss='mean_absolute_error', metrics=['accuracy'])
```

I fit the model over 30 epochs with updates every 300 images(the batch size). Only 50000 samples are trained, the other 10000 samples are for validation.

```
In [13]: # train the model on the training data
         model_training = model.fit(x=X_train, y=y_train, batch_size=300,
                                    epochs=30, verbose=0, validation_split=1/6)
```

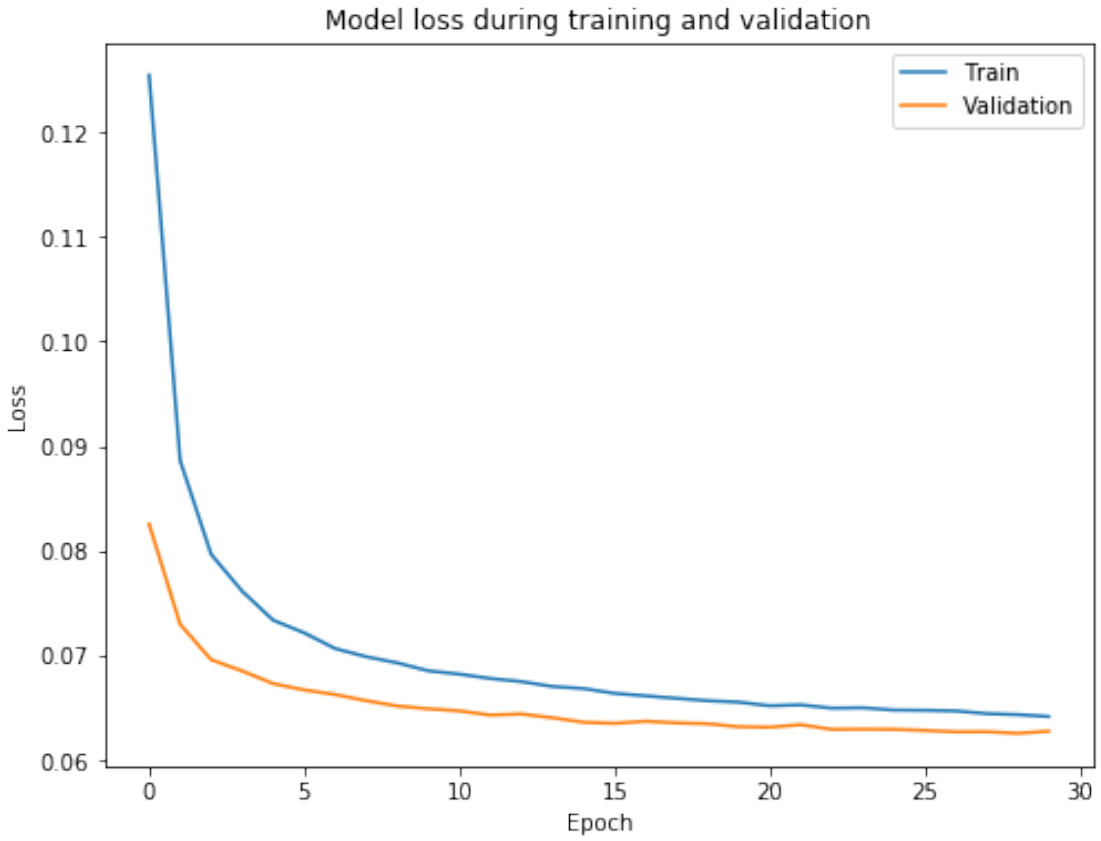
```
In [14]: # plot the accuracy of the model during training and validation
         plt.figure(figsize=(8, 6))
         plt.plot(model_training.history['acc'])
         plt.plot(model_training.history['val_acc'])
         plt.title('Model accuracy during training and validation')
         plt.ylabel('Accuracy')
         plt.xlabel('Epoch')
         plt.legend(['Train', 'Validation'], loc='best')
```

```
Out[14]: <matplotlib.legend.Legend at 0x111265d978>
```




```
In [15]: # plot the loss of the model during training and validation
         plt.figure(figsize=(8, 6))
         plt.plot(model_training.history['loss'])
         plt.plot(model_training.history['val_loss'])
         plt.title('Model loss during training and validation')
         plt.ylabel('Loss')
         plt.xlabel('Epoch')
         plt.legend(['Train', 'Validation'], loc='best')

Out[15]: <matplotlib.legend.Legend at 0x111e944198>
```




```
In [16]: # evaluate the model on the test data
         scores = model.evaluate(X_test, y_test, verbose=0)
         print("Classification Error: {0:02} %".format(100-scores[1]*100))
```

```
Classification Error: 31.450000000000003 %
```

Conclusion: Using the library keras, we can build a simple 1-hidden layer neural network and achieve a pretty good result on the MNIST dataset. The result is still very far from state-of-the-art though, and it can be improved by using more complex methods like convolutional neural network.

The project was very challenging to me. But it was also pretty fun and Keras is definitely a great deep learning framework for beginner(other frameworks, for example Tensorflow is much more complicated in my opinion!).