Exercise 2 Report

March 12, 2017

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

```
In [1]: import matplotlib.pyplot as plt
        from keras.datasets import mnist
        # some magic functions for jupyter notebook
        %matplotlib inline
        %load_ext autoreload
        %autoreload 2
```

```
Using Theano backend.
```

1 Prepare the data.

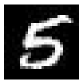
In this section, I do the following steps:

- Load the MNIST data into numpy arrays
- Create and display the original images of some samples
- Normalize the input pixel values to 0-1 range
- Transform integer class vectors into binary class matrix

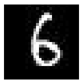
```
In [2]: # load the mnist data
        (X_train, y_train), (X_test, y_test) = mnist.load_data()

        # flatten 28x28 images to a (1,784) vector for each image
        num_pixels = X_train.shape[1] * X_train.shape[2]
        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')
```

```
In [3]: from IPython.display import Image, display
        import numpy as np
        import scipy.misc
        def create_image(array):
            '''Helper function to create the original image from a sample'''
            if array.shape == (784,):
                img_arr = np.reshape(array, (28, 28))
            else:
                raise ValueError('the array have to be size (784,)!')
            scipy.misc.imsave('figures/original character.jpg', img_arr)
            image_name = 'figures/original character.jpg'
            return image_name


        def display_image(image_name):
            '''Display the image'''
            display(Image(filename=image_name, width=100, height=100))
```

```
In [4]: from helpers.plotting_helper import create_image, display_image


        # display the sample number 20001 in train set
        image = create_image(X_train[20000, :])
        display_image(image)
```



```
In [5]: # display the sample number 3001 in test set
        image = create_image(X_test[3000, :])
        display_image(image)
```



```
In [6]: # normalize inputs from 0-255 to 0-1
        X_train = X_train / 255
        X_test = X_test / 255
```

```
In [7]: from keras.utils import np_utils

        # one hot encode outputs
        y_train = np_utils.to_categorical(y_train)
        y_test = np_utils.to_categorical(y_test)
        num_classes = y_test.shape[1]
```

2 Deep Learning!

In this section, I follow the tasks provided to build a simple deep learning model using keras.

- The model is a neural network with one hidden layer.
- It has the same number of neurons as the number of input pixels(784).
- The hidden layer uses a rectifier activation function on the neurons.
- The final(output) layer uses a softmax activation function to turn the outputs into probability-like values(that add up to 1).
- For compiling the model, I uses logarithmic loss as the loss function, stochastic gradient descent as optimizer, and classification accuracy as the metric.

```
In [8]: from keras.models import Sequential
        from keras.layers import Dense
        import matplotlib.pyplot as plt


        def baseline_model(num_pixels, num_classes):
            '''Define a baseline model'''
            # create model
            model = Sequential()
            model.add(Dense(num_pixels, input_dim=num_pixels,
                            init='normal', activation='relu'))
            model.add(Dense(num_classes, init='normal',
                            activation='softmax'))
            # Compile model
            model.compile(loss='categorical_crossentropy',
                          optimizer='sgd', metrics=['accuracy'])
            return model


        def plot_acc_history(history):
            '''Plot the accuracy history'''
            plt.plot(history.history['acc'])
            plt.plot(history.history['val_acc'])
            plt.title('model accuracy')
            plt.ylabel('accuracy')
            plt.xlabel('epoch')
            plt.legend(['train', 'validation'], loc='upper left')
            plt.show()


        def plot_loss_history(history):
            '''Plot the loss history'''
            plt.plot(history.history['loss'])
            plt.plot(history.history['val_loss'])
            plt.title('model loss')
            plt.ylabel('loss')
            plt.xlabel('epoch')
            plt.legend(['train', 'validation'], loc='upper left')
            plt.show()
```

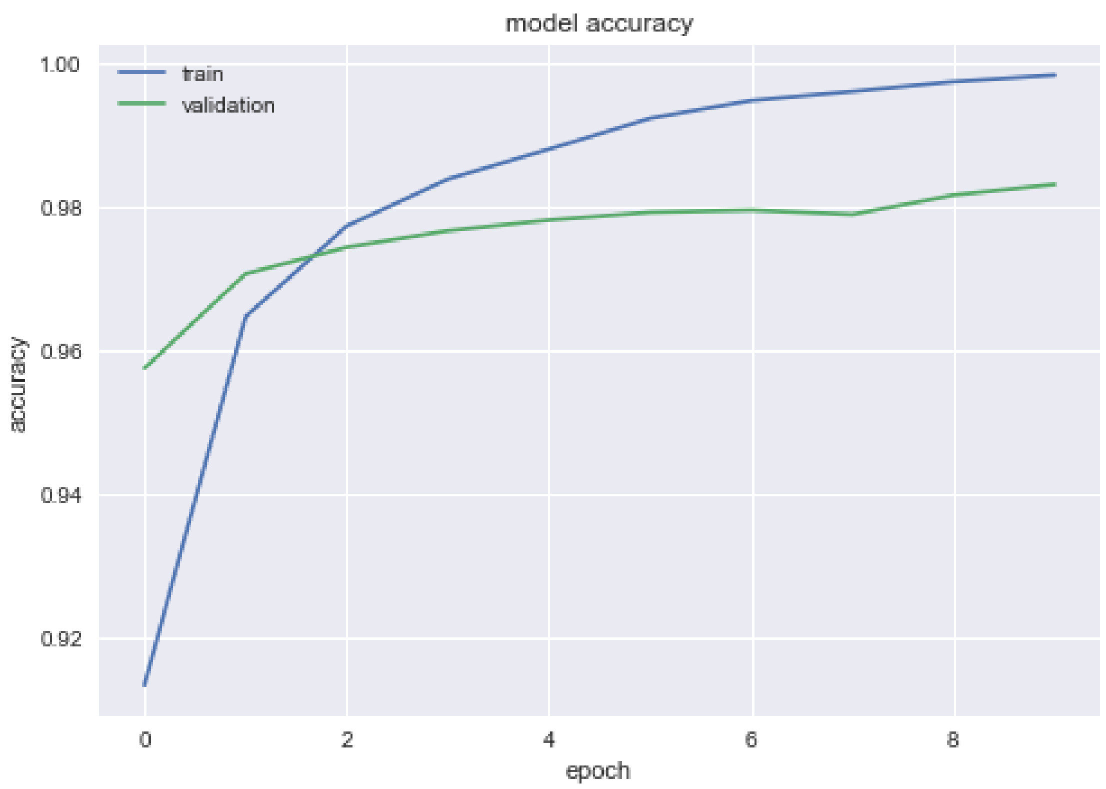
```
In [16]: from helpers.base_model_helper import baseline_model

         # build the model
         model = baseline_model(num_pixels, num_classes)
         # fit the model on the first 50000 training samples
         history = model.fit(X_train, y_train, validation_split=1/6,
                             nb_epoch=10, batch_size=200, verbose=2)
```

```
Train on 50000 samples, validate on 10000 samples
Epoch 1/10
4s - loss: 0.3027 - acc: 0.9122 - val_loss: 0.1439 - val_acc: 0.9623
Epoch 2/10
4s - loss: 0.1244 - acc: 0.9633 - val_loss: 0.1097 - val_acc: 0.9669
Epoch 3/10
5s - loss: 0.0799 - acc: 0.9771 - val_loss: 0.0849 - val_acc: 0.9755
Epoch 4/10
7s - loss: 0.0562 - acc: 0.9838 - val_loss: 0.0821 - val_acc: 0.9743
Epoch 5/10
6s - loss: 0.0415 - acc: 0.9882 - val_loss: 0.0713 - val_acc: 0.9778
Epoch 6/10
5s - loss: 0.0294 - acc: 0.9924 - val_loss: 0.0687 - val_acc: 0.9805
Epoch 7/10
6s - loss: 0.0213 - acc: 0.9951 - val_loss: 0.0672 - val_acc: 0.9799
Epoch 8/10
7s - loss: 0.0154 - acc: 0.9969 - val_loss: 0.0717 - val_acc: 0.9794
Epoch 9/10
6s - loss: 0.0128 - acc: 0.9973 - val_loss: 0.0659 - val_acc: 0.9809
Epoch 10/10
6s - loss: 0.0089 - acc: 0.9987 - val_loss: 0.0727 - val_acc: 0.9801
```

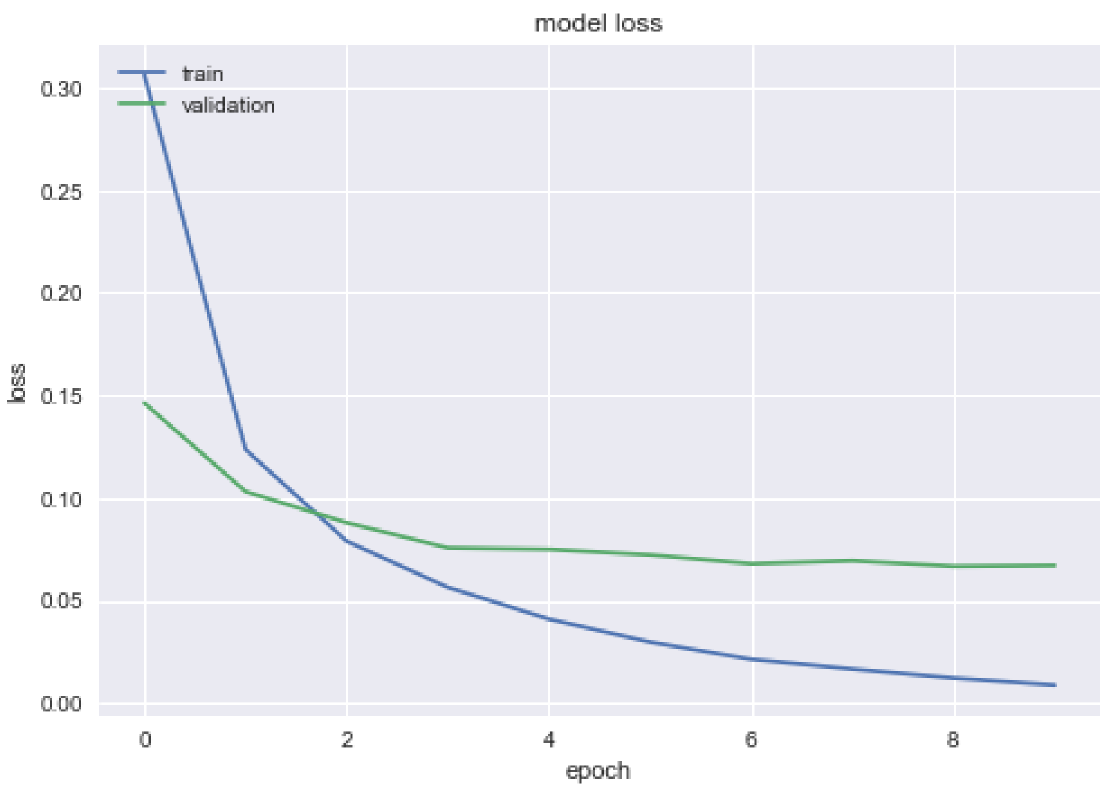
I fit the model over 10 epochs with gradient updates every 200 images(the batch size). Only 50000 samples are trained, the other 10000 samples are for validation. I use a verbose value of 2 to reduce the output to one line for each training epoch.

Next, I plot the accuracy and loss history

```
In [10]: from helpers.base_model_helper import plot_acc_history


         # plot the accuracy history
         plot_acc_history(history)
```



```
In [11]: from helpers.base_model_helper import plot_loss_history

         # plot the loss history
         plot_loss_history(history)
```



From these 2 plots, we can see that at first, the model's performance on the train set is worse than on the validation set. Then on the third epoch, they overlap. After the third epoch, the accuracy of the train set begin to increase faster, maybe indicate that the model are overfitting the train data.

```
In [12]: # predict on the test set
         y_test_predicted = model.predict(X_test)

In [13]: # the classification accuracy
         scores = model.evaluate(X_test, y_test, verbose=0)
         print("Classification Error: %.2f%%" % (100-scores[1]*100))
```

```
Classification Error: 1.98%
```

After training, the model can be used for prediction on the test set, and it yeild an impressive classification error: only 1.98%!

Conclusion: Using the library keras, we can build a simple 1-hidden layer neural network and achieve great result on the MNIST dataset. This is just a simple and very high-level application of deep learning though. I will need to study more math to really understand all of the technical details under the hood!