Due date: Nov 15th. Please submit an .ipynb file.

Instructions:

1) The .ipynb file shall include not only the **source code**, but also necessary **plots/figures** and **discussions** which include your *observations*, *thoughts* and *insights*.
2) Please avoid using a single big block of code for everything then plotting all figures altogether. Instead, use a small bock of code for each sub-task which is followed by its plots and discussions. This will make your homework more readable.
3) Please follow common software engineering practices, e.g., by including sufficient **comments** to functions, important statements, etc.

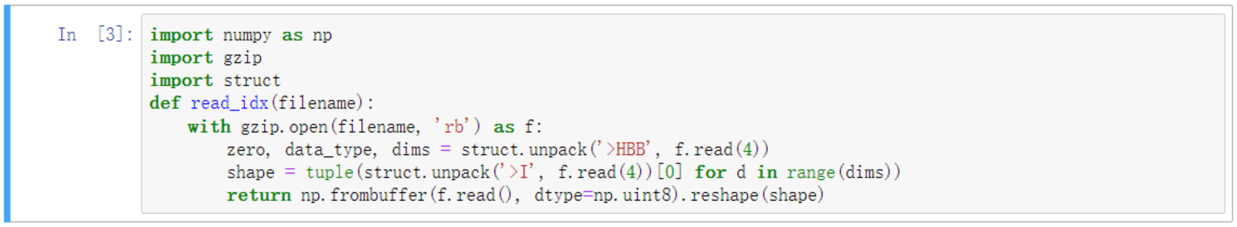
Question 1 – Programming (80 points):

In this programming problem, you will get familiar with building a neural network using backpropagation. You will write a program that learns how to recognize the handwritten digits using stochastic gradient descent and the MNIST training data.

The MNIST database (Modified National Institute of Standards and Technology database is a large database of handwritten digits that is commonly used for training various image processing systems

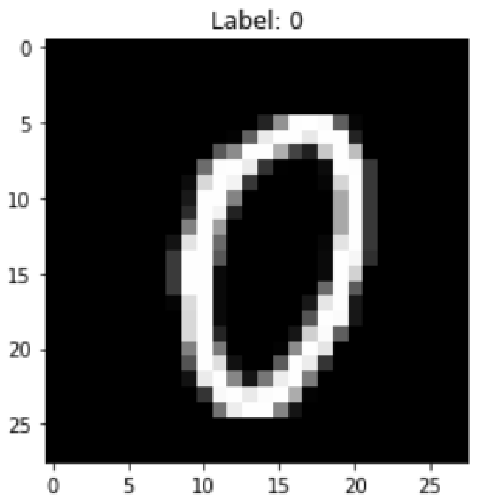
Step 1 Data Acquisition and Visualization (10 pts): In this step, you need to:

(a) Download the "MNIST" dataset and extract the files. You will get four files with **extension .gz** (e.g., train-images-idx3-ubyte.gz). You can use the provided function **read_idx below** to read in the dataset. As its official description, the dataset is split into 60000 training images and 10000 images. The four file corresponds to the training images, training labels, testing images and testing labels. You need to print out their shape to finish this step. **(5 pts)**



```
In [3]: import numpy as np
        import gzip
        import struct
        def read_idx(filename):
            with gzip.open(filename, 'rb') as f:
                zero, data_type, dims = struct.unpack('>HBB', f.read(4))
                shape = tuple(struct.unpack('>I', f.read(4))[0] for d in range(dims))
                return np.frombuffer(f.read(), dtype=np.uint8).reshape(shape)
```

(b) To further understand what the dataset is, you need to use the 'matplotlib' library to print out a random data with code **plt.imshow** together with **its label.(5 pts)** You will see something like this:



Step 2 Data Preprocessing (10 pts): In this step, you need to:

(a) Normalize the pixel values of images to be between 0 and 1. **(5 pts)**
(b) Convert the labels from categorical data into numerical values using one-hot encoding. **(5 pts)** hint: you can explore the eye function in Numpy.

Step 3 Network Initialization (10 pts): We will work with a neuron network with two hidden layers, using Sigmoid function as the activation functions for hidden layers and softmax activation function for the output layer. To finish this, you need to:

(a) Identify the auxiliary input including the **Sigmoid** function and its **derivative** and **Softmax** function **(5 pts)**

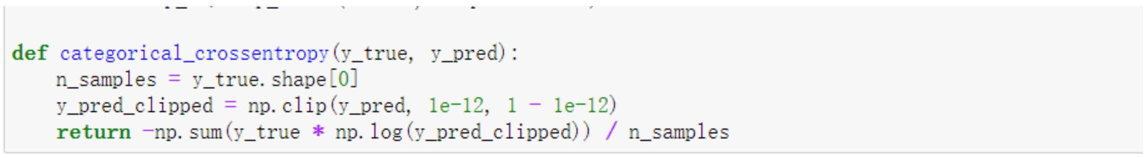
(b) Initialize all the parameters in neural network uniformly. In this network, the input size is **784** dimensions (each input is a 28x28 image, so you have to flatten the data from 2D to 1D). For the two linear hidden layers, we have **128** and **64** neurons respectively. For the output layer, its size will be **10** since there are 10 classes (0-9) in MNIST. To finish this step, you need to **initialize the weights and bias** in random with a pre-set **random seed** using Numpy. Please set the **seed value = 695**. **(5 pts)**

Step 4 Feed Forward (10 pts): In this step, you need to:

(a) Define a function named feed_forward. Given an input x, it should output the **sigmoid of wx+b** where w and b indicates the weights and bias defined in step 2. **(5 pts)**

Step 5 Back Propagation (15 pts): In this step, you need to implement the back propagation:

(a) You need to compute the loss for the output layer first. Here, we use **categorical cross entropy loss function given below** for multi-class classification problem. (5 pts) Note, to achieve this, you need to first encode the categorical labels as numerical values using **one-hot** encoding finished in step 2. **(5 pts)**



```
def categorical_crossentropy(y_true, y_pred):
    n_samples = y_true.shape[0]
    y_pred_clipped = np.clip(y_pred, 1e-12, 1 - 1e-12)
    return -np.sum(y_true * np.log(y_pred_clipped)) / n_samples
```

(b) Calculate the gradients for the weights and bias for each layer. Use the chain rule to compute gradients for previous layers. **(10 pts)**

Step 6 Model Training (15 pts): In this step, you need to:

(a) Use **mini-batch gradient descent** to update the parameters including weights and bias. Notice that a complete training round consists of a feed forward process, back propagation and parameter update. Define the **batch size = 128 and epoch = 100.**

Step 7 Model Evaluation (10 pts): In this step, you need to:

(a) Use your trained neural network to predict the labels of the test dataset and compute the accuracy on the test dataset. **(5 pts)**
Remark: if you correctly execute every step above, you will probably get a result around **90%.**

(b) Plot some of the misclassified images with their predicted and true labels. **(5 pts)** This probably can give you some insights into why these images are misclassified.