Deep Learning

Assignment 1

Implementation of Neural Network

Submission:

Submit all of your codes and results in a single zip file with name FirstName_RollNumber_01.zip

- Submit single zip file containing
  (a) codes (b) report (c) Saved Models (d) Readme.txt
- There should be **Report.pdf** detailing your experience and highlighting any interesting results. Kindly don't explain your code in the report, just explain the results. Your report should include your comments on the results of all the steps, with images, for example what happened when you changed the learning rate etc.
- Readme.txt should explain how to run your code, preferably it should accept the command line arguments e.g dataset path used for training the model.
- The assignment is only acceptable in .py files. No Jupyter notebooks.
- In the root directory, there should be **2** python files, a report and a folder containing saved models.
- Root directory should be named as **FirstName_RollNumber_01**
- Your code notebooks should be named as **'rollNumber_01_task1.ipynb'**
- Follow all the naming conventions.
- For each convention, there is a 3% penalty if you don't follow it.
- Email instructor or TA if there are any questions. You cannot look at others code or use others code, however you can discuss with each other. Plagiarism will lead to a straight zero with additional consequences as well.
- 2% (of obtained marks) deduction per day for late submission.
- The submissions will only be accepted till 22nd April midnight.
- DON'T RESUBMIT THE DATASETS PROVIDED IN YOUR SUBMISSION.

Due Date: 11:59 PM on Sunday, 24th April 2022

Note: For this assignment (and for others in general) you are not allowed to search online for any kind of implementation. Do not share code or look at the other's code. You should not be in possession of any implementation related to the assignment, other than your own. In case of any confusion please reach out to the TA's (email them or visit them during their office hours).

Objectives: In this assignment you will write the code for training the Neural Networks. The goals of this assignment are as follows.

- Mathematical derivation of backpropagation equations.
- Understand how to design and implement an efficient layered architecture for Neural Networks.
- For each layer:

- Implement feedforward (preferably in vectorized fashion).
- Keep track of the local gradients and activations.
- Understand the mechanism of back-propagation (preferably in vectorized fashion).

- Comprehend how different activation functions behave i.e. Sigmoid, ReLU, Tanh, Swish, ELU
- Implement Binary cross-entropy loss function.

NOTE: You can only use Numpy for code implementations. It's recommended that you use VS Code for debugging.

Report: You have to write a report explaining, what is your **implementation logic**? How and why you **distributed the data** in three parts i.e. Train, Test and Validation set? How you came up with the **optimized weights** (how you find learning rate?). Which **activation function** performs best on the given dataset? Finally, **your comment**?

Task 1: Backpropagation (20 Points)

Consider a neural network with N input units, N output units and K hidden units. The activations are computed as follows:

$$\boldsymbol{z} = \boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}$$

$$\boldsymbol{h} = \sigma(\boldsymbol{z})$$

$$\boldsymbol{y} = \boldsymbol{x} + \boldsymbol{W}^{(2)}\boldsymbol{h} + \boldsymbol{b}^{(2)}$$

where σ denotes the logistic function applied elementwise. The cost will involve both $\boldsymbol{h}$ and $\boldsymbol{y}$:

$$\mathcal{E} = \mathcal{R} + \mathcal{S}$$

$$\mathcal{R} = \boldsymbol{r}^T\boldsymbol{h}$$

$$\mathcal{S} = \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{s}\|_2$$

for given $\boldsymbol{r}$ and $\boldsymbol{s}$ vectors.

- Draw the computation graph relating $\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{h}, \boldsymbol{y}, \mathcal{R}, \mathcal{S}, \mathcal{E}$
- Derive backprop equations for computing $\bar{x} = \frac{\partial \mathcal{E}}{\partial \boldsymbol{x}}$. You may use σ' to denote the derivative of logistic function (so you don't need to write it explicitly.)

Task 2: Implement a Neural Network with a single hidden layer, multiple activation function and cross entropy loss function (10 Points)

Steps for the implementation are below:

1. You are given a class named Neural_Network which will take sizes of input, output and hidden layers as a parameter.

a. Input size: Size of the input feature vector to our Neural Network. In our case it is 2, pass it as a parameter.
b. Hidden layer: number of neurons in the hidden layer of the architecture.
c. Output size: is the size output layer, number of neurons that will be in the output layer.
d. Randomly initialize W1 and W2 i.e. Weight matrices connecting input layer to hidden layer and hidden layer to output layer respectively.
e. Pass activation function as an input in the class e.g. 'Sigmoid', 'Tanh' or 'Relu'.

2. Below is the list of functions which you need to implement within the class Neural_Network, description of each function is provided in the source code.
   a. Feedforward (self, X)
      i. X is input feature(s) ii.
      Return predicted vector(s)
   b. Backpropogation (self,X, Y, y_pred, lr):
      i. X is input feature(s)
      ii. Y is actual label(s)
      iii. Y_pred is predicted value(s)
      iv. lr is learning rate
   c. Sigmoid(self, s)
      i. Return sigmoid applied on s value(s)
   d. Sigmoid_derivative(self, s)
      i. Return derivative of sigmoid, on s
   e. tanh(self, s)
      i. Return tanh applied on s value(s)
   f. tanh_derivative(self, s)
      i. Return derivative of tanh, on s
   g. relu(self, s)
      i. Return relu applied on s value(s)
   h. relu_derivative(self, s)
   i. Return derivative of relu, on s
   i. Crossentropy(self, Y, Y_pred)
   i. Y is the actual label(s) ii. Y_pred is a predicted label(s) iii. Return error based on cross entropy
   j. Train(self, trainX, trainY,epochs = 100, learningRate = 0.001, plot_err = True ,validationX = Null, validationY = Null)
      i. trainX is the training feature dataset in row format
      ii. trainY is the label of the dataset iii. epochs is the number of times the entire dataset will be passed to the network for training, default value is 100.

iv. learningRate is the constant used for weight update.

v. plot_err bool variable if you want to plot error on a graph or not

vi. validationX is the validation feature dataset in row format, show validation error in each epoch

vii. validationY is the validation label of the dataset.

k. Save_model(self,name)

i. Save the model under the name of 'name'

l. Load_model(self,name)

i. Load the model using 'name'.

m. Accuracy(self, testX, testY)

i. testX is dataset for testing

ii. testY is the labels of test data

iii. plot accuracy on an image

iv. return accuracy

n. predict(self, testX)

i. testX is the test row feature

ii. return predicted value on testX

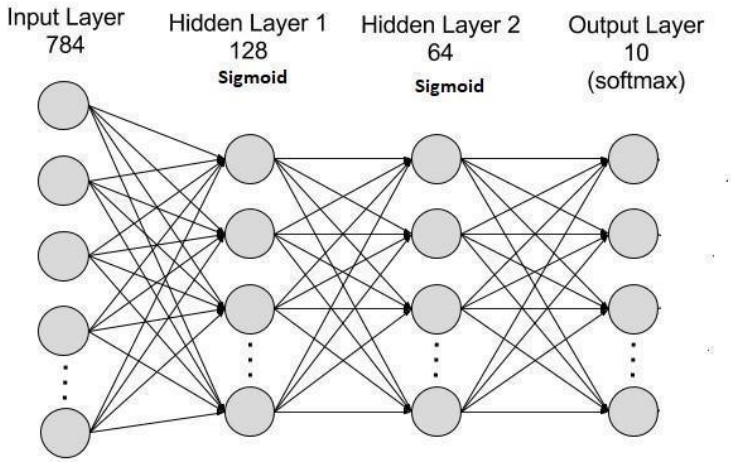
o. main()

i. Call all the functions to train and test the network in this function.

ii. Find the accuracy and loss curves of training and validation data and print the accuracy of the test set and return all these values.

Task 3: Implement a multi-layer Neural Network for multi-class classification.

In this part, you will create a complete neural network architecture consisting of multiple layers. You are required to report results with 2 hidden layers on the MNIST dataset. ***Follow the architecture of your network as shown in the below diagram.***



MNIST Dataset:

The dataset is attached with the assignment in the Task3_Data folder. This dataset contains *60000* training and *10000* test samples. Each sample is a grayscale image of size 28x28. There are 10 classes in which each sample is an image of one of the digits (0 to 9). Please note that in the MNIST dataset there are 10 categories. If we randomly guess one category, there is a 1/10 probability that it would be correct. Therefore, you cannot (theoretically) make a classifier that performs worse than that. If you get less than 10% accuracy in any of your experiments, you can safely assume that you are doing something fundamentally wrong. If your final results are less than 20% in terms of accuracy, your solution(s) will not be graded. To load the dataset, we are providing you a function *load_dataset* that will return you the followings:

```
train_set_x, train_set_y, test_set_x, test_set_y =

load_dataset (path_to_dataset)
```

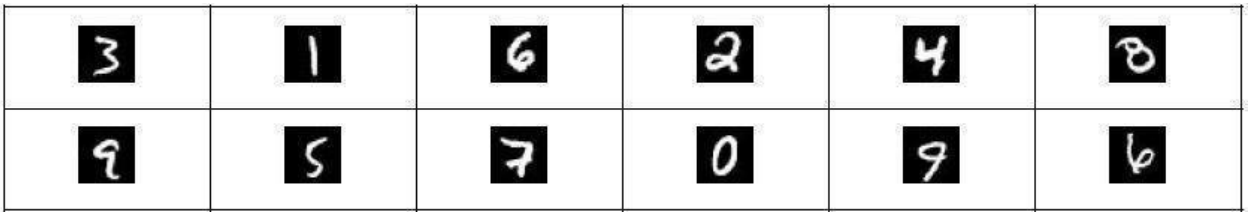


Figure: Sample Images from the Dataset

```
# Function to load dataset
from matplotlib import image as img
def loadDataset(path):
   print('Loading Dataset...')
        train_x, train_y, test_x, test_y = [], [], [], []
        for i in range(10):
                for filename in glob.glob(path + '\\train\\' + str(i)+'\\*.png'):
im=img.imread(filename)
                train_x.append(im)
                train_y.append(i)
        for i in range(10):
                for filename in glob.glob(path + '\\test\\' + str(i)+'\\*.png'):
                        im=img.imread(filename)
                        test_x.append(im)
                        test_y.append(i)
        print('Dataset loaded...')
        return np.array(train_x), np.array(train_y),
np.array(test_x),np.array(test_y)
```

Note: You can also use any other custom function you want to load the data.

Points to keep in mind:

- *train_set_x* and *test_set_x* are numpy arrays of shape `(m_train, num_px, num_px, 1)` and `(m_test, )` respectively, where `m_train` is total number of train or test images, and `num_px` is the width and height of images that is 28 in case of MNIST.
  - `*_set_x` : is being used for the sample
  - `*_set_y` : is being for the label of the sample.
- The dimension of `train_set_y` and `test_set_y` should be (`m_train, 1`) and `(m_test , )` respectively
- Each row of your `train_set_x` and `test_set_x` should be an array representing an image. Write the following code in your main notebook to verify that the `train_set_x` is correctly loaded by visualizing any image using the following code (Feel free to change the index value and re-run to see other images).

```
index = 25
plt.imshow(train_set_x[index])
```

To verify that all the data is loaded print the dimension of each variable and you should get the following outputs:

```
train_set_x shape: (60000, 28, 28, 1)
test_set_x shape: (10000, 28, 28, 1)
```

```
train_set_y shape: (60000, 1)
test_set_y shape: (10000, 1)
```

Vectorization of Samples:

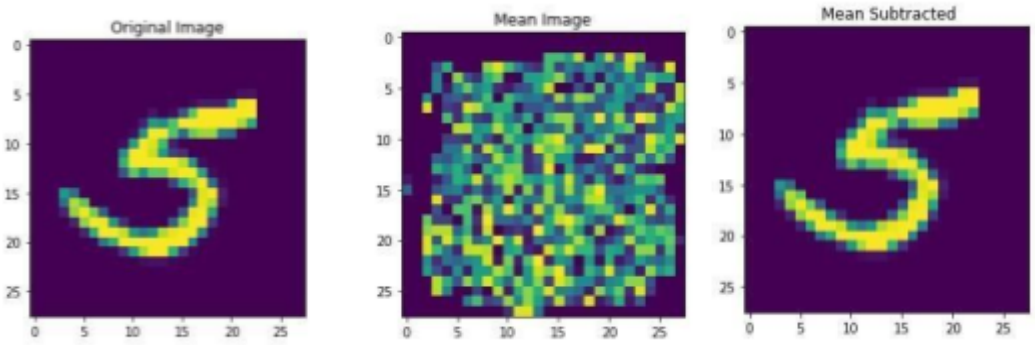
To input samples in the neural network, we need to convert a 2D matrix into a one dimensional vector. Reshape your images of shape (num_px, num_px, 1) to a flattened numpy array of shape `(num_px*num_px, 1)`. After reshaping, your training and test dataset should be a numpy-array where each row represents a flattened image. The dimension of the data should be:

```
Train_set_x_flatten shape: (60000, 784)
Test_set_x_flatten shape: (10000, 784)
```

HINT: there is a reshape function in the numpy allowing you to reshape the whole matrix with a single call. **Do check that you have vectorized the images correctly.**

Mean Image Subtraction:

There are many ways to do mean subtraction but here we are using mean image subtraction. Mean Image Subtraction is a method of normalizing the dataset. We compute the mean of the whole dataset and subtract the mean from each sample in our dataset. In case of MNIST dataset example is given below:



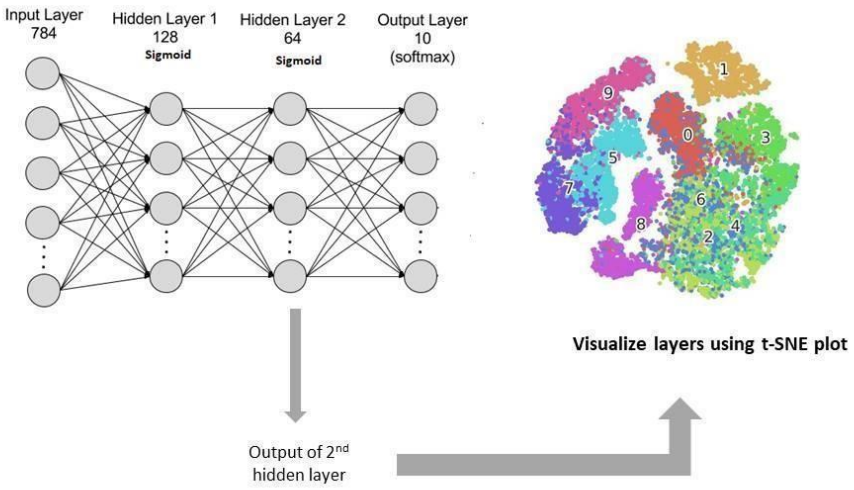
Mean subtraction will translate our dataset to the mean of the whole data.

Validation Set:

Divide the training dataset after flattening into train and validation set. You can try different fractions for this division. You can use the sklearn.model_selection.train_test_split() function for this task. Try different versions where you also shuffle the data to divide them into these two sets.

Note: Do not use the final instances for the validation set. The dataset loading code loads the instances in order. You might not get the instances for the last class (that comes in order which will probably be class 9) in the training set at all this way. Remember to shuffle the training data before train test split or use the library builtin function to do this.

t-SNE: t- Distributed Stochastic Neighbor Embedding (t-SNE) is an unsupervised, non-linear technique primarily used for data exploration and visualizing high-dimensional data. In simpler terms, t-SNE gives you a feel or intuition of how high dimensional data is arranged in space.



You can use sklearn library to plot t-SNE and visualize data points.

Once you have loaded the data you need to implement the several helper functions.

1. Data preprocessing

```
data = meanSubtraction(data)
```

Note that this function must be called before splitting data into train, test and validation set.

2. Initialize Network

```
net = init_network(no_of_layers, input_dim, neurons_per_layer)
```

For example if you pass following parameters to this function:

```
net = init_network(2, 784, [100, 50, 10])
```

- Use `np.random.randn()` to initialize the weights matrices or you can use any other weight initialization method you learned in the class.

It should return you the network architecture with parameters initialized:

```
Size of net(1).w = 784x100
Size of net(1).b = 100
Size of net(2).w = 100x50
```

```
Size of net(2).b = 50
Size of net(3).w = 50x10
Size of net(3).b = 10
```

You should add the empty arrays in each layer to store the activations and local gradients of each layer; this will help you in back-propagation.

3. Training

```
net = train_sgd(net, train_set_x, train_set_y, test_set_x,
test_set_y, learning_rate, drop_out, batch_size, training_epochs)
```

- This function returns a trained Neural Network `net`
- `net` network architecture
- `train_set_x` input training data of MxN size. where M is the feature dimension which is 784 for MNIST, and N is the number of training examples
- `train_set_y` 1xN array containing class labels i.e. [0,1,2,..,9]
- `test_set_x` and `test_set_y` will be used for validation
- `learning_rate` as the name suggest, learning rate of your Neural Network
- `batch_size` tells how many examples to pick for each iteration e.g. 20
- `training_epochs` how many training epochs to run.

Please make sure your code is modular. You can divide your training process into following function

- ➢ **feed-forward** - this function will forward through (the network) your input examples and will also compute local gradients along the way at each layer
- ➢ **back-prop** - this function will traverse network backwards and keep
- ➢ **back propagating the loss** - multiplying gradients from above to the current local gradients
- ➢ **validate/test** - function will test how well your network is doing in terms of loss and accuracy.

You need to convert the labels to one hot encoding, because we now have 10 classes and their labels are 0,1,2,3...and 9. For each training sample you need to generate a vector of length 10, whose all indices will be zeros except the index of its original label that will be 1. For example:

Training Sample	True Label	one-hot-encoding									
x1	5	0	0	0	0	0	1	0	0	0	0
x2	7	0	0	0	0	0	0	0	1	0	0
x3	0	1	0	0	0	0	0	0	0	0	0

4. Feed Forward step

```
net = feedForward(net, batch_data, keep_prob)
```

$$z_i = w_T x_i + b \; a^i$$

$$= sigmoid(z^i)$$

Perform these two operations on all the layers and store the value of **a** in net(layer).a

At this step also store the value of derivative of your activation function in net(layer).local_grad
Derivative of sigmoid is $\sigma(x)*(1 - \sigma(x))$.

5. Back Propagation Step

```
[net cost] = backPropagation(net, batch_label, batch_size,
learningRate, keep_prob)
```

The first step here, you will apply the soft-max on the output layer. Now for each input example you have 10 output probabilities and you also have its label vector of same length. You have to implement softmax at the last layer and calculate cross-entropy loss.

Once you have calculated the loss you can compute the gradients `dw` at any layer using the equations we discussed in class. Please note that you will essentially be running in reverse order. Store `dw` and `db` for each layer.

6. Update Step

```
[net] = sgd (net, dw, db, learning_rate)
```

Again here you will be running a for loop by iterating layers from first to last and update the weights using update rule.

```
net(i).w = net(i).w - (learning_rate * dW(i))
net(i).b = net(i).b - (learning_rate * dB(i))
```

7. Testing Step `[net] = test(net, test_set_x, test_set_y)`

Since you have updated your weights for 1 epoch, now perform a feedforward step on your test data and compute the loss. After every epoch plot the training loss and validation loss.

Merging all functions together:

You will now see how the overall model is structured by putting together all the building blocks (functions that you implemented in the previous parts) together, in the right order. Now implement a main function model in which call all the functions in the correct order to train and test your network.

Note: You are not restricted to implement the assignment in a way that is explained above, you can break down or merge the several functions, but you are required to implement in a modular way.

Report

For this assignment, and all other assignments and projects, you must write a report. In the report you will describe the critical decisions you made, important things you learned, or any decisions you made to write your algorithm a particular way. You are required to report the accuracy you achieved. For each experiment, you are required to provide analysis of various hyperparameters.

1. Plot loss and accuracy curves for both training and testing data with mean image subtraction and without mean image subtraction. Report the difference in their accuracy and loss curves.
2. Visualize data points using t-SNE technique
    - Having a shallow network of 2 hidden layers, take an output of 1st and 2nd hidden layer and plot t-SNE and place those images in your report. If you have more layers, you need to plot t-SNE plots after each layer for your best accuracy model.
    - Add more hidden layers in your network and report how it affects the t-SNE plot(You can take a small subset of like 500 images for comparison).
3. Show and analyse confusion matrix of 10- class classification problem.
4. Plot loss and accuracy curves for different configurations of the architecture..
5. Report the accuracy by changing the number of neurons in the hidden layers.

Marks Division:

1. The marks division is as below:
    a. Working code [50 points]
        i. Task 2
            1. Feedforward [10 points]
            2. FeedBackward [10 points]
            3. Activations [10 points]
            4. Loss [05 points]
        ii. Task 3

1. Feedforward [10 points]
2. FeedBackward [10 points]
3. Activations [10 points]
4. Loss [05 points]
5. Generic [10 points]
6. Mean Subtraction [05 points]
7. t-SNE plots [15 points]

iii.

b. Report [40 points]

i. Task 1

1. Computation Graph [10 points]
2. Mathematical Derivation [10 points]

ii. Task 2

1. Loss and accuracy curves [10 points]
2. Test Accuracy [05 points]
3. Analysis(in different experiments) [10 points]

iii. Task 3

1. Loss and accuracy curves with and without mean subtraction [10 points]
2. Test Accuracy [05 points]
3. Test Accuracy [05 points]
4. Confusion Matrix for Training, Validation and Test set [10 points]
5. t_SNE plot [10 points]
6. Analysis(in different experiments) [10 points]

Conclusion [10 points]

make confusion matrix after training model

c. Evaluation and Viva [10 points]