Copy_of_fall2022_hw3

November 12, 2022

1 CS171-EE142 - Fall 2022 - Homework 3

2 Due: Tuesday, November 15, 2022 @ 11:59pm

2.0.1 Maximum points: 80 pts

2.1 Submit your solution to Gradescope:

1. Submit a single PDF to **HW3**
2. Submit your jupyter notebook to **HW3-code**

See the additional submission instructions at the end of this notebook

2.2 Enter your information below:

2.2.1 Your Name (submitter): Yash Aggarwal

2.2.2 Your student ID (submitter): 862333037

By submitting this notebook, I assert that the work below is my own work, completed for this course. Except where explicitly cited, none of the portions of this notebook are duplicated from anyone else's work or my own previous work.

2.3 Academic Integrity

Each assignment should be done individually. You may discuss general approaches with other students in the class, and ask questions to the TAs, but you must only submit work that is yours . If you receive help by any external sources (other than the TA and the instructor), you must properly credit those sources, and if the help is significant, the appropriate grade reduction will be applied. If you fail to do so, the instructor and the TAs are obligated to take the appropriate actions outlined at http://conduct.ucr.edu/policies/academicintegrity.html . Please read carefully the UCR academic integrity policies included in the link.

3 Overview

In this assignment you will implement a two-layer neural network. You will implement the loss functions, gradients, optimizers to train the network and test its performance on MNIST dataset.

For this assignment we will use the functionality of Pandas (https://pandas.pydata.org/), Matplotlib (https://matplotlib.org/), and Numpy (http://www.numpy.org/).

If you are asked to **implement** a particular functionality, you should **not** use an existing implementation from the libraries above (or some other library that you may find). When in doubt, please ask.

Before you start, make sure you have installed all those packages in your local Jupyter instance

3.1 Read *all* cells carefully and answer all parts (both text and missing code)

You will complete all the code marked `TODO` and answer descriptive/derivation questions

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import math
from sklearn.utils import shuffle

# make sure you import here everything else you may need
```

3.1.1 Load MNIST Dataset

For this assignment, we will use MNIST handwritten digits data set. The dataset consists 10 handwritten digits (0,1,...,9). It is a widely used dataset to demonstrate simple image classification problem.

MNIST dataset is publicly available from different sources. We will be using MNIST from Keras package. If you do not have Keras installed, you can find the installation guide here.

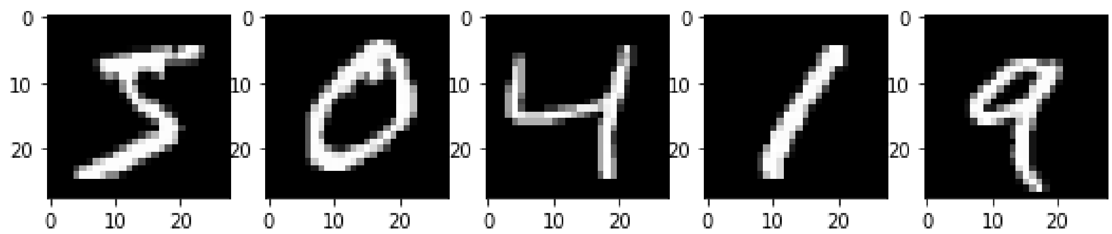
In short, you need to run `conda install -c anaconda keras` or `pip install keras`

The training data consists of 60000 images of size 28×28 pixels; the test data consists of 10000 images.

[2]:
```
from keras.datasets import mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()

print('Training data shape:',x_train.shape)
print('Test data shape:',x_test.shape)

n_img=5
plt.figure(figsize=(n_img*2,2))
plt.gray()
for i in range(n_img):
```

```
    plt.subplot(1,n_img,i+1)
    plt.imshow(x_train[i])
plt.show()
```

```
Training data shape: (60000, 28, 28)
Test data shape: (10000, 28, 28)
```



We will be vectorizing the training and test images. So, the size of each vector will be 784.

[3]:
```
x_train=x_train.reshape(x_train.shape[0],-1)
x_test=x_test.reshape(x_test.shape[0],-1)

print('Training data shape after reshaping:',x_train.shape)
print('Test data shape after reshaping::',x_test.shape)
```

```
Training data shape after reshaping: (60000, 784)
Test data shape after reshaping:: (10000, 784)
```

3.2 Question 1: Binary classification using neural network [45 pts]

We will start with classification of images for two different digits using a two-layer network with a cross entropy loss.

In the next question, we will extend the same architecture to multi-class classification.

Pick any two digits out of ten for our classification (say 5 and 8), which we will assign label ”0” or ”1”.

Pick same number of images from each class for training and create arrays for input and output (say 1000).

```
# train_x -- N x 784 array of training input
# train_y -- N x 1 array of binary labels
```

If you use 1000 images from each class N = 2000. You can increase the number of training samples if you like. It is just a suggestion.

We also need to transpose the dimension of the data so that their size becomes $784 \times N$. It will be helpful to feed it to our model based on our notations.

[4]:
```python
def extract_binary_classification_dataset(x, y, label1, label2, num_samples):
    """Make a subset dataset from MNIST, containing only 2 classes for binary␣
↪classification task
    Args:
        x (numpy.ndarray): data, can be x_train or x_test
        y (numpy.ndarray): labels of data, can be y_train or y_test
        label1 (int): the first class you pick, e.g. 5
        label2 (int): the second class you pick, e.g. 8
        num_samples (int): the number of images you select for each class, e.g.␣
↪1000
    Returns:
        x_ (numpy.ndarray): the data for 2 picked classes
        y_ (numpy.ndarray): the corresponding labels for 2 picked classes
    """
    # for class 1
    x1 = x[y == label1]
    x1 = x1[:num_samples]
    y1 = np.zeros(len(x1))

    # for class 2
    x2 = x[y == label2]
    x2 = x2[:num_samples]
    y2 = np.ones(len(x2))

    # combine 2 classes
    x_ = np.concatenate((x1,x2),axis=0)
    y_ = np.concatenate((y1,y2),axis=0)
    return x_, y_


# Pick your own digits
label1 = 5
label2 = 8
num_samples = 1000

# Train & test data
train_x, train_y = extract_binary_classification_dataset(x_train, y_train,␣
↪label1, label2, num_samples)
test_x, test_y = extract_binary_classification_dataset(x_test, y_test, label1,␣
↪label2, num_samples)

# reshape data
# Images are stored row wise, store it column wise
train_x = train_x.T
test_x = test_x.T
print("Training data shape:", train_x.shape)
print("Training labels shape:", train_y.shape)
```

```
print("Test data shape:", test_x.shape)
print("Test labels shape:", test_y.shape)
```

```
Training data shape: (784, 2000)
Training labels shape: (2000,)
Test data shape: (784, 1866)
Test labels shape: (1866,)
```

3.2.1 Network Architecture

We will be using a two layer neural network in our experiment. The input layer will have 784 nodes, the hidden layer will have 256 nodes and the output layer will have 1 node. Each node will have *sigmoid* activation function.

The equations for feedforward operation will be the following:

$$\mathbf{z}^{(1)} = W^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\mathbf{y}^{(1)} = \varphi(\mathbf{z}^{(1)})\mathbf{z}^{(2)} = W^{(2)}\mathbf{y}^{(1)} + \mathbf{b}^{(2)}\mathbf{y}^{(2)} = \varphi(\mathbf{z}^{(2)})$$

where $\mathbf{x} \in \mathbb{R}^{784}$ is the input layer, $\mathbf{y}^{(1)} \in \mathbb{R}^{256}$ is the hidden layer, $\mathbf{y}^{(2)} \in \mathbb{R}$ is the output layer, $W^{(1)} \in \mathbb{R}^{256\times784}$ is the first layer weights, $W^{(2)} \in \mathbb{R}^{1\times256}$ is the second layer weights, $\mathbf{b}^{(1)} \in \mathbb{R}^{256}$ is the first layer bias, $\mathbf{b}^{(2)} \in \mathbb{R}$ is the second layer bias, $\varphi(\cdot)$ is the activation function.

3.2.2 Network initialization [5 pts]

We initialize the weights for $W^{(1)}$ and $W^{(2)}$ with random values drawn from normal distribution with zero mean and 0.01 standard deviation. We will initialize bias vectors $\mathbf{b}^{(1)}$ and $\mathbf{b^{(2)}}$ with zero values.

We can fix the seed for random initialization for reproducibility.

[5]:
```
def TwoLayerNetwork(layer_dims=[784,256,1]):

    # Fix the seed
    np.random.seed(3)

    mean = 0
    std = 0.01

    # TODO
    # Your code goes here
    params = {}
    params['w1'] = np.random.normal(mean, std,␣
 ↪size=(layer_dims[1],layer_dims[0]))
    params['b1'] = np.zeros(layer_dims[1])
    params['w2'] = np.random.normal(mean, std,␣
 ↪size=(layer_dims[2],layer_dims[1]))
    params['b2'] = np.zeros(layer_dims[2])
```

```
    return params
```

3.2.3 Sigmoid activation function

Now we will write the sigmoid activation function as

$$\varphi(z) = \frac{1}{1+e^{-z}}$$

Note that derivative of **sigmoid** is $\varphi'(z) = \varphi(z)(1-\varphi(z))$.

[6]:
```
def sigmoid(Z):
    # Input: Z -- numpy.ndarray
    # TODO
    Y = (1 / (1 + np.exp(-Z)))
    return Y
```

3.2.4 Cross entropy loss function [5 pts]

We will minimize the binary cross entropy loss function. You will use the true labels and predicted labels of a batch of N samples.

Binary crossentropy loss for i^{th} sample can be written as

$$Loss_i = -y_i \log y_i^{(2)} - (1-y_i)\log(1-y_i^{(2)})$$

where y_i is the true label. We can find the average loss for a batch of N samples as $Loss = \frac{1}{N}\sum_{i=1}^{N} Loss_i$.

Note that the gradient of the cross entropy loss w.r.t. the output is

$$\nabla_{y^{(2)}} Loss_i = -\frac{y_i}{y_i^{(2)}} + \frac{1-y_i}{1-y_i^{(2)}} = \frac{y_i^{(2)} - y_i}{y_i^{(2)}(1-y_i^{(2)})}.$$

We can also show that

$$\delta^{(2)} = \nabla_{\mathbf{z}^{(2)}} Loss_i = \nabla_{y^{(2)}} Loss_i \odot \varphi'(\mathbf{z}) = y_i^{(2)} - y_i,$$

where $\odot$ denotes element-wise multiplication of the arrays.

[7]:
```
def CrossEntropyLoss(Y_true, Y2):
    # TODO
    # Write your code here

    loss = 0
    for y_true,y_pred in zip(Y_true, Y2.T):
```

```
        l = (y_true*np.log(y_pred)) + ((1-y_true)*np.log(1-y_pred))
        l = -l
        loss += l

    return loss[0] / len(Y2)
```

3.2.5 Forward propagation [5 pts]

Next, we will write the code for the forward pass for two layer network. Each layer consists of an affine function (fully-connected layer) followed by an activation function. You wil also return the intermediate results ($\mathbf{x}, \mathbf{z}^{(1)}, \mathbf{y}^{(1)}, \mathbf{z}^{(2)}$) in addition to final output ($\mathbf{y}^{(2)}$). You will need the intermediate outputs for the backpropagation step.

[8]:
```
def forward(X, params):

    # TODO
    # Write your codes here
    # X -- 784 x N array
    # params --
    # w1 -- 256 x 784 matrix
    # b1 -- 256 x 1 vector
    # W2 -- 1 x 256 matrix
    # b2 -- 1 x 1 scalar
    # Y2 -- 1 x N output

    intermediate = {}

    intermediate['X'] = X
    intermediate['z1'] = params['w1'].dot(intermediate['X'])+params['b1'].
 ↪reshape(params['b1'].shape[0], 1)
    intermediate['Y1'] = sigmoid(intermediate['z1'])
    intermediate['z2'] = params['w2'].dot(intermediate['Y1'])+params['b2'].
 ↪reshape(params['b2'].shape[0], 1)
    Y2 = sigmoid(intermediate['z2'])

    return Y2, intermediate
```

3.2.6 Backpropagation step [10 pts]

Now we will implement the backpropagation step for the two layer neural network.

You will need the gradient of the Loss w.r.t. $W^{(l)}, \mathbf{b}^{(l)}$ for $l = 1, 2$ for all the training samples.

We saw that we can write the gradient of Loss with respect to $W^{(l)}, \mathbf{b}^{(l)}$ for a single sample as

$$\nabla_{W^{(l)}} Loss_i = \delta^{(l)} \mathbf{y}^{(l-1)T},$$

$$\nabla_{\mathbf{b}^{(l)}} Loss_i = \delta^{(l)},$$

where

$$\delta^{(l)} = \nabla_{\mathbf{z}^{(l)}} Loss_i = \nabla_{\mathbf{y}^{(l)}} Loss_i \odot \varphi'(\mathbf{z}^{(l)}).$$

For the the last layer, we can compute $\delta^{(L)}$ by plugging the value of $\nabla_{\mathbf{y}^{(L)}} Loss$ as described above.

For the intermediate layers $l < L$, we can write

$$\delta^{(l)} = W^{(l+1)T} \delta^{(l+1)} \odot \varphi'(\mathbf{z}^{(l)}).$$

Once we have the gradients $\nabla_{W^{(l)}} Loss_i, \nabla_{\mathbf{b}^{(l)}} Loss_i$ for all i. We can compute their average to compute the gradient of the total loss function $\frac{1}{N}\sum_{i=1}^{N} Loss_i$ as

$$\nabla_{W^{(l)}} Loss = \frac{1}{N} \sum_i \nabla_{W^{(l)}} Loss_i,$$
$$\nabla_{\mathbf{b}^{(l)}} Loss = \frac{1}{N} \sum_i \nabla_{\mathbf{b}^{(l)}} Loss_i.$$

Please refer to the slides and lectures for more details.

[9]:
```
def backward(Y_true, Y2, intermediate, params):

    # Inputs:
      # Y_true -- 1 x N true labels
      # Y2 -- 1 x N output of the last layer
      # intermediate -- X, Z1, Y1, Z2
      # params -- W1, b1, W2, b2

    # Outputs:
      # grads -- [grad_W1, grad_b1, grad_W2, grad_b2]

    # TODO
    # Write your codes here

    grads = {}

    Y_true_reshaped = train_y.reshape(1,train_y.shape[0])

    delta_2 = Y2 - Y_true_reshaped
    grads['grad_w2'] = delta_2.dot(intermediate['Y1'].T)
    grads['grad_b2'] = np.sum(delta_2,axis=1)

    delta_1 = params['w2'].T.dot(delta_2) * (sigmoid(intermediate['z1'])*(1 -
 ↪sigmoid(intermediate['z1'])))
    grads['grad_w1'] = delta_1.dot(intermediate['X'].T)
```

```
    grads['grad_b1'] = np.sum(delta_1,axis=1)

    return grads
```

3.2.7 Optimizer [5 pts]

We will use a standard gradient descent-based optimizer to minimize the loss function. You have already implemented gradient descent in HW2. You may have to adjust learning rate that provides you best training/validation performance. In this exercise, we are not using validation data; in practice, you should use it to tune your hyperparameters such as learning rate, network architecture etc.

You can use same learning rate for all weights in this assignment.

You should update $W^1, \mathbf{b}^1, W^2, \mathbf{b}^2$ as

$$W^1 \leftarrow W^1 - \alpha \nabla_{W^1} Loss$$
$$\mathbf{b}^1 \leftarrow \mathbf{b}^1 - \alpha \nabla_{\mathbf{b}^1} Loss$$
$$W^2 \leftarrow W^2 - \alpha \nabla_{W^2} Loss$$
$$\mathbf{b}^2 \leftarrow \mathbf{b}^2 - \alpha \nabla_{\mathbf{b}^2} Loss$$

α is the learning rate.

[10]:
```
def GD(params, grads, learning_rate):

    # updated params = old params - learning rate * gradient of Loss computed
 ↪at old params
    # TODO
    # Write your codes here

    params['w2'] = params['w2'] - learning_rate*grads['grad_w2']
    params['b2'] = params['b2'] - learning_rate*grads['grad_b2']
    params['w1'] = params['w1'] - learning_rate*grads['grad_w1']
    params['b1'] = params['b1'] - learning_rate*grads['grad_b1']


    return params
```

[11]:
```
def predict(x, params):
  Y2, _ = forward(x, params)
  Y2 = np.array([1 if y_i > 0.5 else 0 for y_i in Y2.T]).reshape(1,-1)
  return Y2
```

[12]:
```
def accuracy(y, y_pred):
  aa = y_pred.reshape(-1)
  bb = np.array([y == aa])
  acc = np.sum(bb) / bb.shape[1]
```

```
    return acc
```

3.2.8 Train the Model [5 pts]

We will train the model using the functions we wrote above.

First, we specify the number of nodes in the layers, number of epochs and learning rate. Then we initialize the network.

[13]:
```
layer_dims = [train_x.shape[0],256,1]
epochs = 100
lr = 0.00001

params = TwoLayerNetwork(layer_dims)
```

Then we train the network for the number of epochs specified above. In every epoch, we will do the following: 1. Calculate the forward pass to get estimated labels. 2. Use the estimated labels calculate loss. We will be recording loss for every epoch. 3. Use backpropagation to calculate gradients. 4. Use gradient descent to update the weights and biases.

You should store the loss value after every epoch in an array `loss_history` and print the loss value after every few epochs (say 20).

[14]:
```
# TODO
# Write your codes here

loss_history = []
train_acc = []
test_acc = []
for epoch in range(epochs):

    print ('For Epoch : ', epoch)

    Y2, intermediate = forward(train_x, params)

    loss = CrossEntropyLoss(train_y, Y2)
    loss_history.append(loss)

    train_pred = predict(train_x, params)
    train_accuracy = accuracy(train_y, train_pred)
    train_acc.append(train_accuracy)

    test_pred = predict(test_x, params)
    test_accuracy = accuracy(test_y, test_pred)
    test_acc.append(test_accuracy)

    print ('loss', loss)
    print ('train_accuracy', train_accuracy)
```

```python
    print ('test_accuracy', test_accuracy)

    grads = backward(train_y, Y2, intermediate, params)
    params = GD(params, grads, lr)

    print('-------------')
```

```
For Epoch :  0
loss 1382.6803822154936
train_accuracy 0.5005
test_accuracy 0.5219721329046088
-------------
For Epoch :  1
loss 1344.6835766468082
train_accuracy 0.5615
test_accuracy 0.5659163987138264
-------------
For Epoch :  2
loss 1312.5122541437577
train_accuracy 0.719
test_accuracy 0.6854233654876741
-------------
For Epoch :  3
loss 1281.2094111728147
train_accuracy 0.813
test_accuracy 0.7867095391211146
-------------
For Epoch :  4
loss 1249.7418429811107
train_accuracy 0.872
test_accuracy 0.8397642015005359
-------------
For Epoch :  5
loss 1218.2988593657276
train_accuracy 0.8925
test_accuracy 0.87513397642015
-------------
For Epoch :  6
loss 1186.226182628
train_accuracy 0.9125
test_accuracy 0.897642015005359
-------------
For Epoch :  7
loss 1153.0783868022802
train_accuracy 0.9215
test_accuracy 0.912647374062165
-------------
```

```
For Epoch :  8
loss 1118.9972745434084
train_accuracy 0.9275
test_accuracy 0.9185423365487674
-------------
For Epoch :  9
loss 1084.7422031586223
train_accuracy 0.931
test_accuracy 0.9217577706323687
-------------
For Epoch :  10
loss 1049.964695444493
train_accuracy 0.9355
test_accuracy 0.9287245444801715
-------------
For Epoch :  11
loss 1015.0413282089033
train_accuracy 0.9395
test_accuracy 0.9297963558413719
-------------
For Epoch :  12
loss 980.2564032048854
train_accuracy 0.9425
test_accuracy 0.9330117899249732
-------------
For Epoch :  13
loss 945.7846655431864
train_accuracy 0.9445
test_accuracy 0.935155412647374
-------------
For Epoch :  14
loss 911.9151813785345
train_accuracy 0.945
test_accuracy 0.9367631296891747
-------------
For Epoch :  15
loss 878.7881808361788
train_accuracy 0.9455
test_accuracy 0.9367631296891747
-------------
For Epoch :  16
loss 846.4799514354353
train_accuracy 0.9455
test_accuracy 0.9372990353697749
-------------
For Epoch :  17
loss 815.1096598949241
train_accuracy 0.946
```

```
test_accuracy 0.9378349410503751
-------------
For Epoch :  18
loss 784.7641641162679
train_accuracy 0.947
test_accuracy 0.9389067524115756
-------------
For Epoch :  19
loss 755.7800848341215
train_accuracy 0.949
test_accuracy 0.9389067524115756
-------------
For Epoch :  20
loss 728.2806699924226
train_accuracy 0.95
test_accuracy 0.9410503751339764
-------------
For Epoch :  21
loss 700.7745553112132
train_accuracy 0.95
test_accuracy 0.9421221864951769
-------------
For Epoch :  22
loss 675.5968491512332
train_accuracy 0.9505
test_accuracy 0.939978563772776
-------------
For Epoch :  23
loss 651.7580419153363
train_accuracy 0.9505
test_accuracy 0.9437299035369775
-------------
For Epoch :  24
loss 629.4161590733192
train_accuracy 0.9525
test_accuracy 0.9405144694533762
-------------
For Epoch :  25
loss 607.9837002835761
train_accuracy 0.954
test_accuracy 0.9431939978563773
-------------
For Epoch :  26
loss 588.0771111520274
train_accuracy 0.954
test_accuracy 0.9405144694533762
-------------
For Epoch :  27
```

```
loss 568.9542933708474
train_accuracy 0.954
test_accuracy 0.9437299035369775
-------------
For Epoch :  28
loss 550.9514971628929
train_accuracy 0.954
test_accuracy 0.9421221864951769
-------------
For Epoch :  29
loss 534.1031271491797
train_accuracy 0.957
test_accuracy 0.9442658092175777
-------------
For Epoch :  30
loss 517.9299618387331
train_accuracy 0.9555
test_accuracy 0.9437299035369775
-------------
For Epoch :  31
loss 502.8528300972274
train_accuracy 0.9595
test_accuracy 0.9448017148981779
-------------
For Epoch :  32
loss 488.54445990220347
train_accuracy 0.957
test_accuracy 0.9448017148981779
-------------
For Epoch :  33
loss 475.0297021935305
train_accuracy 0.9605
test_accuracy 0.9453376205787781
-------------
For Epoch :  34
loss 462.1930003750459
train_accuracy 0.9575
test_accuracy 0.9453376205787781
-------------
For Epoch :  35
loss 450.09908950239543
train_accuracy 0.962
test_accuracy 0.9464094319399786
-------------
For Epoch :  36
loss 438.6311993035542
train_accuracy 0.9605
test_accuracy 0.9448017148981779
```

```
-------------
For Epoch :  37
loss 427.95115157440546
train_accuracy 0.9625
test_accuracy 0.947481243301179
-------------
For Epoch :  38
loss 417.53504663422353
train_accuracy 0.9605
test_accuracy 0.9448017148981779
-------------
For Epoch :  39
loss 407.81528601071926
train_accuracy 0.9635
test_accuracy 0.9469453376205788
-------------
For Epoch :  40
loss 398.14437162628786
train_accuracy 0.9625
test_accuracy 0.9458735262593784
-------------
For Epoch :  41
loss 389.1551206294519
train_accuracy 0.965
test_accuracy 0.947481243301179
-------------
For Epoch :  42
loss 380.2852605283457
train_accuracy 0.9635
test_accuracy 0.9464094319399786
-------------
For Epoch :  43
loss 372.0185010513821
train_accuracy 0.966
test_accuracy 0.947481243301179
-------------
For Epoch :  44
loss 363.8777274380071
train_accuracy 0.9645
test_accuracy 0.9480171489817792
-------------
For Epoch :  45
loss 356.370319027147
train_accuracy 0.969
test_accuracy 0.9469453376205788
-------------
For Epoch :  46
loss 349.07849445408755
```

```
train_accuracy 0.9655
test_accuracy 0.9480171489817792
-------------
For Epoch :  47
loss 342.04857086558974
train_accuracy 0.9705
test_accuracy 0.9469453376205788
-------------
For Epoch :  48
loss 335.3163594747206
train_accuracy 0.968
test_accuracy 0.9480171489817792
-------------
For Epoch :  49
loss 328.92587601927966
train_accuracy 0.971
test_accuracy 0.947481243301179
-------------
For Epoch :  50
loss 322.72953350925746
train_accuracy 0.968
test_accuracy 0.947481243301179
-------------
For Epoch :  51
loss 316.77669957919323
train_accuracy 0.9715
test_accuracy 0.9490889603429796
-------------
For Epoch :  52
loss 310.9018892917722
train_accuracy 0.9695
test_accuracy 0.9480171489817792
-------------
For Epoch :  53
loss 305.53306522956825
train_accuracy 0.9715
test_accuracy 0.9496248660235799
-------------
For Epoch :  54
loss 300.11038717171647
train_accuracy 0.9715
test_accuracy 0.9490889603429796
-------------
For Epoch :  55
loss 295.1209981975711
train_accuracy 0.9725
test_accuracy 0.9506966773847803
-------------
```

```
For Epoch :  56
loss 289.971813327418
train_accuracy 0.9725
test_accuracy 0.9490889603429796
-------------
For Epoch :  57
loss 285.5585209905177
train_accuracy 0.9735
test_accuracy 0.9506966773847803
-------------
For Epoch :  58
loss 280.860778194801
train_accuracy 0.9735
test_accuracy 0.9496248660235799
-------------
For Epoch :  59
loss 276.7921480922436
train_accuracy 0.975
test_accuracy 0.9512325830653805
-------------
For Epoch :  60
loss 272.38521213530464
train_accuracy 0.974
test_accuracy 0.9496248660235799
-------------
For Epoch :  61
loss 268.724036334955
train_accuracy 0.975
test_accuracy 0.9512325830653805
-------------
For Epoch :  62
loss 264.609903551408
train_accuracy 0.974
test_accuracy 0.9490889603429796
-------------
For Epoch :  63
loss 261.0684204095201
train_accuracy 0.9755
test_accuracy 0.9506966773847803
-------------
For Epoch :  64
loss 257.04486153668853
train_accuracy 0.9755
test_accuracy 0.9490889603429796
-------------
For Epoch :  65
loss 253.70201526598004
train_accuracy 0.976
```

```
test_accuracy 0.9506966773847803
-------------
For Epoch :  66
loss 249.83451093039628
train_accuracy 0.976
test_accuracy 0.9496248660235799
-------------
For Epoch :  67
loss 246.4130977127861
train_accuracy 0.977
test_accuracy 0.9506966773847803
-------------
For Epoch :  68
loss 242.64651414104762
train_accuracy 0.9765
test_accuracy 0.9496248660235799
-------------
For Epoch :  69
loss 239.4074944738534
train_accuracy 0.978
test_accuracy 0.9506966773847803
-------------
For Epoch :  70
loss 236.0271812055234
train_accuracy 0.977
test_accuracy 0.9496248660235799
-------------
For Epoch :  71
loss 232.89682599493784
train_accuracy 0.9785
test_accuracy 0.9512325830653805
-------------
For Epoch :  72
loss 229.79909876179318
train_accuracy 0.9775
test_accuracy 0.9496248660235799
-------------
For Epoch :  73
loss 227.1067108766107
train_accuracy 0.9785
test_accuracy 0.9512325830653805
-------------
For Epoch :  74
loss 224.23680445752015
train_accuracy 0.978
test_accuracy 0.9496248660235799
-------------
For Epoch :  75
```

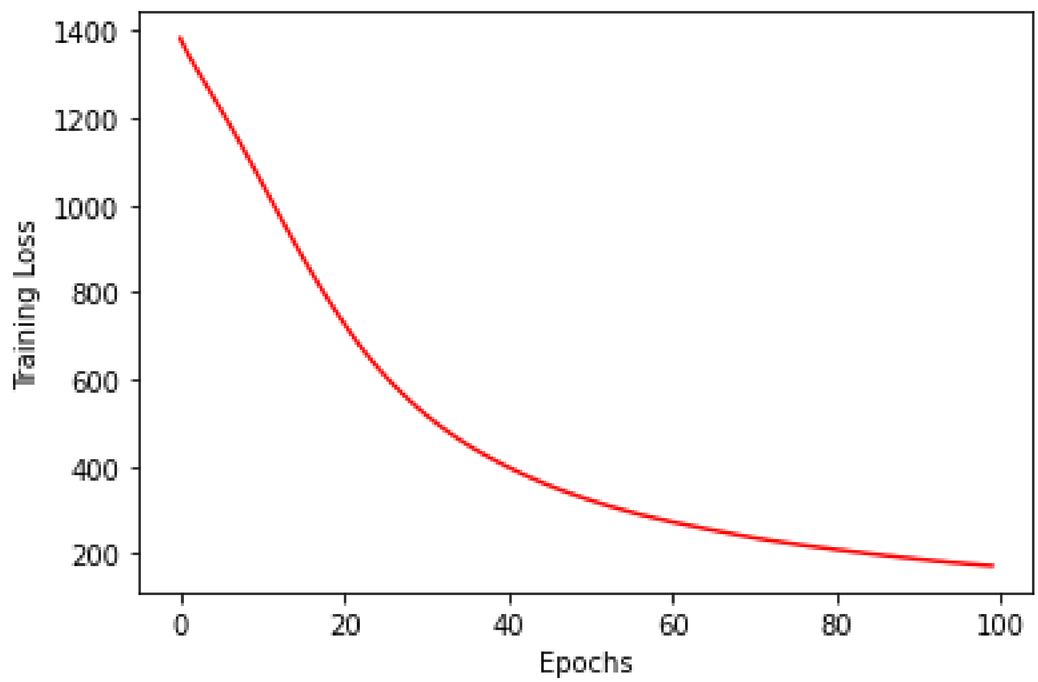
```
loss 221.61447361462677
train_accuracy 0.9785
test_accuracy 0.9512325830653805
-------------
For Epoch :  76
loss 218.79707518410154
train_accuracy 0.978
test_accuracy 0.9496248660235799
-------------
For Epoch :  77
loss 216.46582885639745
train_accuracy 0.979
test_accuracy 0.9506966773847803
-------------
For Epoch :  78
loss 213.7656832981032
train_accuracy 0.9785
test_accuracy 0.9496248660235799
-------------
For Epoch :  79
loss 211.51896219284143
train_accuracy 0.9795
test_accuracy 0.9506966773847803
-------------
For Epoch :  80
loss 208.85735739359754
train_accuracy 0.979
test_accuracy 0.9496248660235799
-------------
For Epoch :  81
loss 206.79491827445753
train_accuracy 0.981
test_accuracy 0.9506966773847803
-------------
For Epoch :  82
loss 204.14006675541643
train_accuracy 0.979
test_accuracy 0.9496248660235799
-------------
For Epoch :  83
loss 202.2052593845316
train_accuracy 0.9815
test_accuracy 0.9506966773847803
-------------
For Epoch :  84
loss 199.5561368809616
train_accuracy 0.979
test_accuracy 0.9490889603429796
```

```
-------------
For Epoch :  85
loss 197.71899577138498
train_accuracy 0.982
test_accuracy 0.9506966773847803
-------------
For Epoch :  86
loss 195.14735017403595
train_accuracy 0.979
test_accuracy 0.9490889603429796
-------------
For Epoch :  87
loss 193.41418262125447
train_accuracy 0.9825
test_accuracy 0.9506966773847803
-------------
For Epoch :  88
loss 190.9266269003494
train_accuracy 0.979
test_accuracy 0.9512325830653805
-------------
For Epoch :  89
loss 189.2729323683954
train_accuracy 0.9825
test_accuracy 0.9506966773847803
-------------
For Epoch :  90
loss 186.88529500257565
train_accuracy 0.9795
test_accuracy 0.9512325830653805
-------------
For Epoch :  91
loss 185.26135078765068
train_accuracy 0.9825
test_accuracy 0.9506966773847803
-------------
For Epoch :  92
loss 183.06718545245567
train_accuracy 0.981
test_accuracy 0.9512325830653805
-------------
For Epoch :  93
loss 181.5041770167333
train_accuracy 0.9825
test_accuracy 0.9501607717041801
-------------
For Epoch :  94
loss 179.50214930850638
```

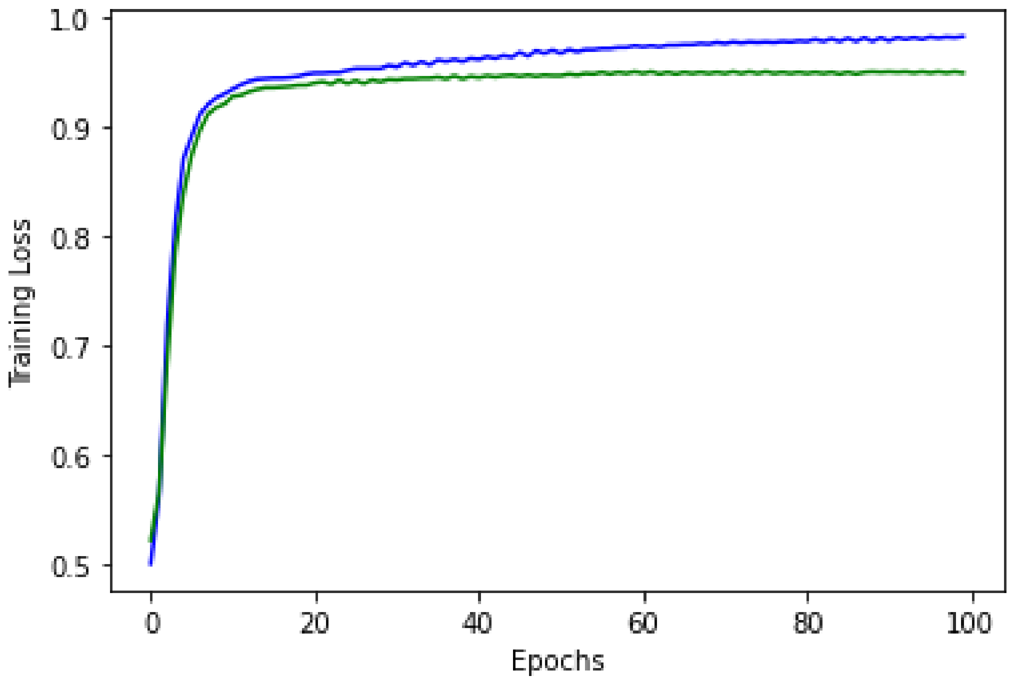
```
train_accuracy 0.981
test_accuracy 0.9512325830653805
-------------
For Epoch :  95
loss 178.0433922053425
train_accuracy 0.983
test_accuracy 0.9501607717041801
-------------
For Epoch :  96
loss 176.16560085132468
train_accuracy 0.982
test_accuracy 0.9512325830653805
-------------
For Epoch :  97
loss 174.81504470879568
train_accuracy 0.983
test_accuracy 0.9501607717041801
-------------
For Epoch :  98
loss 173.07481262019627
train_accuracy 0.9825
test_accuracy 0.9512325830653805
-------------
For Epoch :  99
loss 171.91108795963078
train_accuracy 0.9835
test_accuracy 0.9501607717041801
-------------
```

Now we will plot the recorded loss values vs epochs. We will observe the training loss decreasing with the epochs.

[15]:
```
plt.figure()
plt.plot(loss_history, color = 'red')
plt.xlabel("Epochs")
plt.ylabel("Training Loss")
plt.show()
```



[16]:
```
plt.figure()
# plt.plot(np.log(loss_history), color = 'red')
plt.plot(train_acc, color = 'blue')
plt.plot(test_acc, color = 'green')
plt.xlabel("Epochs")
plt.ylabel("Training Loss")
plt.show()
```



3.2.9 Evaluation on test data [5 pts]

Now we will be evaluating the accuracy we get from the trained model. We feed training data and test data to the forward model along with the trained parameters.

Note that, we need to covert the output probability of the forward pass to binary labels before evaluating accuracy. Since the model provides the posterior probability $p(y = 1|x)$ in range [0,1]. We can binarize them using 0.5 as a theshold (i.e. if $y_i^{(2)} \geq 0.5$, $y_i^{(2)} \leftarrow 1$ otherwise $y_i^{(2)} \leftarrow 0$).

[17]:
```
# TODO
y_pred = predict(train_x, params)
acc = accuracy(train_y, y_pred)
print("Training accuracy:", acc)

y_pred = predict(test_x, params)
acc = accuracy(test_y, y_pred)
print("Test accuracy:", acc)
```

```
Training accuracy: 0.9825
Test accuracy: 0.9512325830653805
```

3.2.10 Visualize some of the correct/miscalassified images [5 pts]

Now we will look at some images from training and test sets that were misclassified.

Training set. Pick 5 images from each class that are correcly and incorreclty classified. True/False Positive/Negatives

Test set. Pick 5 images from each class that are correcly and incorreclty classified. True/False Positive/Negatives

[18]:
```
# TODO
# Training set
print("Training set examples for true/false positive/negative")
Y_hat = predict(train_x, params)
Y_hat = Y_hat.reshape(-1)

positive = []
negative = []
false_positive = []
false_negative = []

idx = 0

for y_hat,y_true,x_i in zip(Y_hat, train_y, train_x.T):
  idx += 1
  if y_hat == y_true:
    if y_hat == 1:
      # print('positive')
      positive.append(x_i)
    else :
      # print('negative')
      negative.append(x_i)
  else :
    if y_hat == 0:
      # print('false_positive')
      false_positive.append(x_i)
    else:
      # print('false_negative')
      false_negative.append(x_i)


positive = np.array(positive)
negative = np.array(negative)
false_positive = np.array(false_positive)
false_negative = np.array(false_negative)

print ('positive', positive.shape)
print ('negative', negative.shape)
print ('false_positive', false_positive.shape)
print ('false_negative', false_negative.shape)
```

```
Training set examples for true/false positive/negative
```

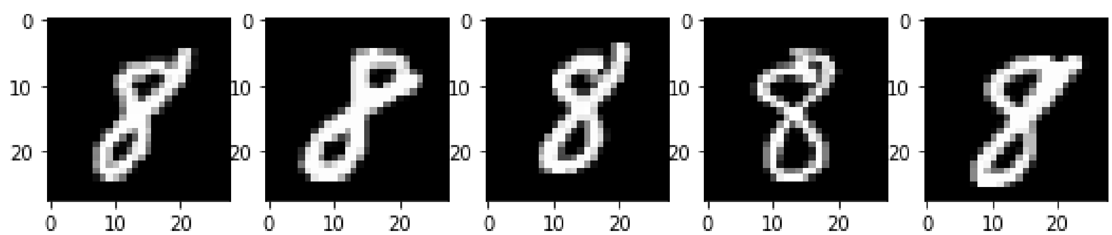
```
positive (977, 784)
negative (988, 784)
false_positive (23, 784)
false_negative (12, 784)
```

[19]:

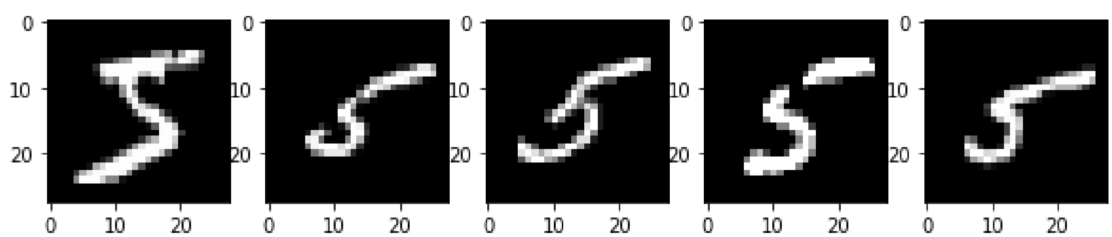
```
print (' --------- Positives -----------')
print (' --------- Label: 8, Predicted: 8 -----------')
print ()

n_img=5
plt.figure(figsize=(n_img*2,2))
plt.gray()
for i in range(n_img):
    plt.subplot(1,n_img,i+1)
    plt.imshow(positive[i].reshape(28,28))
plt.show()

print ()
print (' --------- Negatives -----------')
print (' --------- Label: 5, Predicted: 5 -----------')
print ()

n_img=5
plt.figure(figsize=(n_img*2,2))
plt.gray()
for i in range(n_img):
    plt.subplot(1,n_img,i+1)
    plt.imshow(negative[i].reshape(28,28))
plt.show()

print ()
print (' --------- False Positives -----------')
print (' --------- Label: 8, Predicted: 5 -----------')
print ()

n_img=5
plt.figure(figsize=(n_img*2,2))
plt.gray()
for i in range(n_img):
    plt.subplot(1,n_img,i+1)
    plt.imshow(false_positive[i].reshape(28,28))
plt.show()

print ()
print (' --------- False Negatives -----------')
print (' --------- Label: 5, Predicted: 8 -----------')
print ()
```

```
n_img=5
plt.figure(figsize=(n_img*2,2))
plt.gray()
for i in range(n_img):
    plt.subplot(1,n_img,i+1)
    plt.imshow(false_negative[i].reshape(28,28))
plt.show()
```

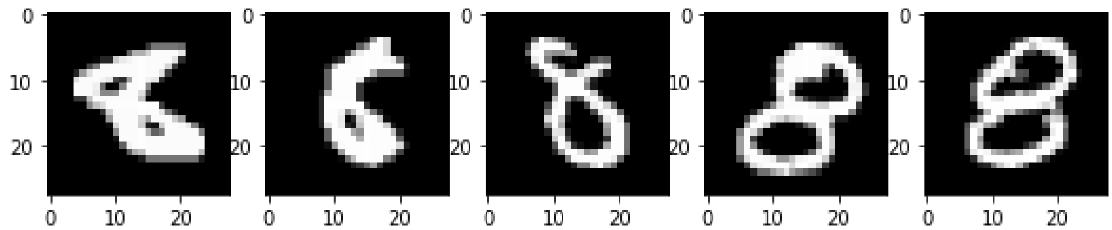
```
--------- Positives -----------
--------- Label: 8, Predicted: 8 -----------
```



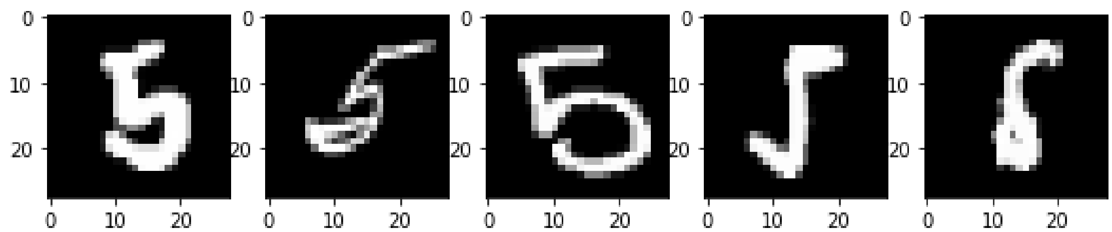
```
--------- Negatives -----------
--------- Label: 5, Predicted: 5 -----------
```



```
--------- False Positives -----------
--------- Label: 8, Predicted: 5 -----------
```



```
--------- False Negatives -----------
--------- Label: 5, Predicted: 8 -----------
```



[20]:
```python
# Test set
print("Test set examples for true/false positive/negative")
Y_hat = predict(test_x, params)
Y_hat = Y_hat.reshape(-1)

positive = []
negative = []
false_positive = []
false_negative = []

idx = 0

for y_hat,y_true,x_i in zip(Y_hat, test_y, test_x.T):
  idx += 1
  if y_hat == y_true:
    if y_hat == 1:
      # print('positive')
      positive.append(x_i)
    else :
      # print('negative')
      negative.append(x_i)
  else :
```

```
    if y_hat == 0:
      # print('false_positive')
      false_positive.append(x_i)
    else:
      # print('false_negative')
      false_negative.append(x_i)


positive = np.array(positive)
negative = np.array(negative)
false_positive = np.array(false_positive)
false_negative = np.array(false_negative)

print ('positive', positive.shape)
print ('negative', negative.shape)
print ('false_positive', false_positive.shape)
print ('false_negative', false_negative.shape)
```

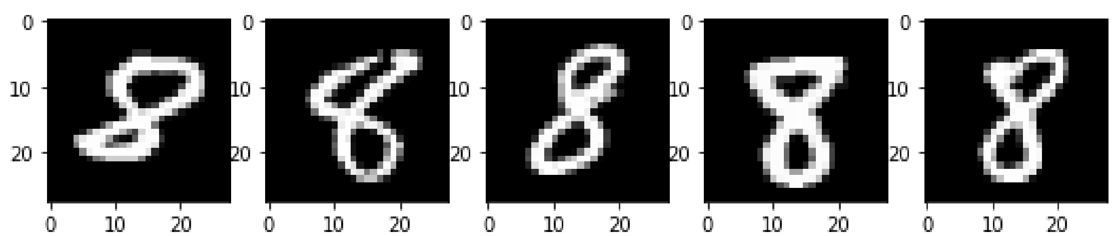
```
Test set examples for true/false positive/negative
positive (927, 784)
negative (848, 784)
false_positive (47, 784)
false_negative (44, 784)
```

[21]:

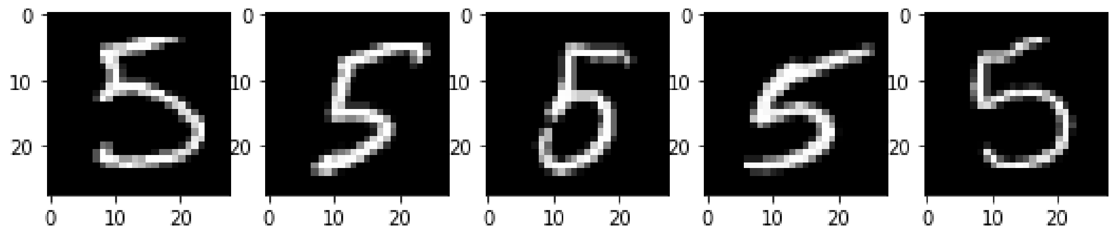
```
print (' --------- Positives -----------')
print (' --------- Label: 8, Predicted: 8 -----------')
print ()

n_img=5
plt.figure(figsize=(n_img*2,2))
plt.gray()
for i in range(n_img):
    plt.subplot(1,n_img,i+1)
    plt.imshow(positive[i].reshape(28,28))
plt.show()

print ()
print (' --------- Negatives -----------')
print (' --------- Label: 5, Predicted: 5 -----------')
print ()

n_img=5
plt.figure(figsize=(n_img*2,2))
plt.gray()
for i in range(n_img):
    plt.subplot(1,n_img,i+1)
    plt.imshow(negative[i].reshape(28,28))
```

```
plt.show()

print ()
print (' --------- False Positives -----------')
print (' --------- Label: 8, Predicted: 5 -----------')
print ()

n_img=5
plt.figure(figsize=(n_img*2,2))
plt.gray()
for i in range(n_img):
    plt.subplot(1,n_img,i+1)
    plt.imshow(false_positive[i].reshape(28,28))
plt.show()

print ()
print (' --------- False Negatives -----------')
print (' --------- Label: 5, Predicted: 8 -----------')
print ()

n_img=5
plt.figure(figsize=(n_img*2,2))
plt.gray()
for i in range(n_img):
    plt.subplot(1,n_img,i+1)
    plt.imshow(false_negative[i].reshape(28,28))
plt.show()
```

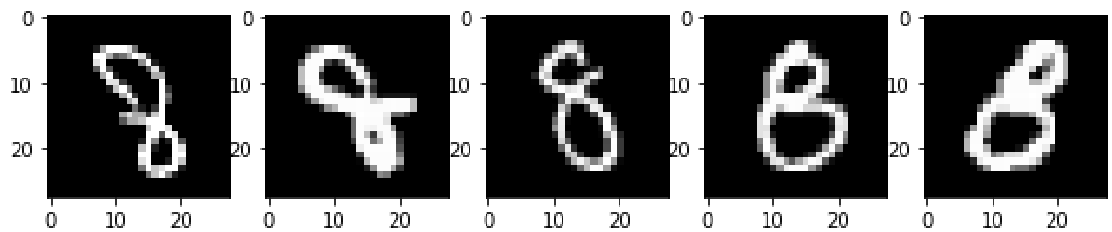
```
--------- Positives -----------
--------- Label: 8, Predicted: 8 -----------
```



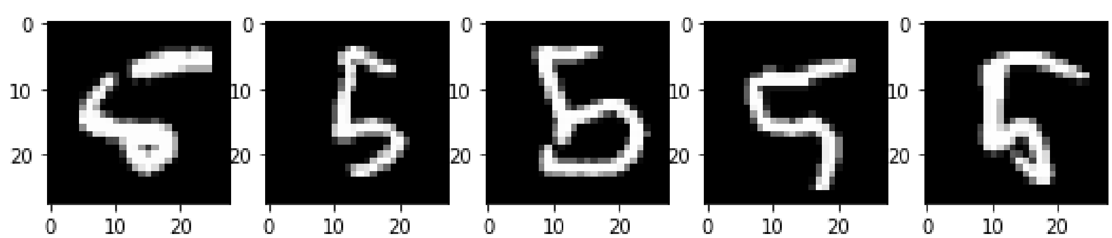

```
--------- Negatives -----------
--------- Label: 5, Predicted: 5 -----------
```



```
--------- False Positives -----------
--------- Label: 8, Predicted: 5 -----------
```



```
--------- False Negatives -----------
--------- Label: 5, Predicted: 8 -----------
```



3.3 Question 2. Multiclass classification [35 pts]

Now we will build a classifier to separate all the digits. For this purpose, we will only change the last layer and the loss.

Instead of using a single output, we will provide 10 outputs; and instead of using a binary cross entropy loss, we will use mutli-class cross entropy loss.

In multinomal logistic regression (aka softmax regression), we define the posterior probability of label $y \in \{0, \dots, K-1\}$ as

$$p(y = c|\mathbf{x}) = \frac{\exp(\mathbf{w}_c^T \mathbf{x})}{\sum_{k=1}^K \exp(\mathbf{w}_k^T \mathbf{x})} = \mathbf{p}_c.$$

In other words, last layer of the network provides a probability vector $\mathbf{p} \in \mathbb{R}^K$, such that each $0 \le \mathbf{p}_c \le 1$ and $\sum_c \mathbf{p}_c = 1$.

3.3.1 Softmax function [5 pts]

Let us first define the softmax function, which is a multinomal extension of the sigmoid function that maps a vector of length K to a probability vector.

We can define `softmax` function on a vector $\mathbf{z} \in \mathbb{R}^K$ as $\mathbf{p} = \text{softmax}(\mathbf{z})$:

$$\mathbf{p}_c(\mathbf{z}) = \frac{\exp(\mathbf{z}_c)}{\sum_{k=1}^K \exp(\mathbf{z}_k)}$$

[22]:
```
def softmax_2(Z):
    # Z -- K x N numpy.ndarray, K is the number of classes, N is the number of␣
 ↪samples
    # TODO
    # your code goes here...
    exp = np.exp(Z)
    exp_sum = exp.sum(0, keepdims=True)
    probs = exp/exp_sum

    return probs
```

We have to note that the numerical range of floating point numbers in numpy is limited. For `float64` the upper bound is 10^{308}. For exponential, its not difficult to overshoot that limit, in which case python returns `nan`.

To make our softmax function numerically stable, we simply normalize the values in the vector, by multiplying the numerator and denominator with a constant `C` as

$$\begin{aligned}
\mathbf{p}_c &= \frac{\exp(\mathbf{z}_c)}{\sum_{k=1}^K \exp(\mathbf{z}_k)} \\
&= \frac{C\exp(\mathbf{z}_c)}{C\sum_{k=1}^K \exp(\mathbf{z}_k)} \\
&= \frac{\exp(\mathbf{z}_c + \log C)}{C\sum_{k=1}^K \exp(\mathbf{z}_k + \log C)}.
\end{aligned}$$

We can choose an arbitrary value for `log(C)` term, but generally `log(C) = -max(z)` is chosen

[23]:
```
def stable_softmax_2(Z):
    # Z -- K x N numpy.ndarray, K is the number of classes, N is the number of␣
 ↪samples
    # TODO (this is optional)
    # your code goes here
    emax = -np.amax(Z)
    regularizer = math.e**emax
    exp = np.exp(Z + np.log(regularizer))
    exp_sum = exp.sum(0, keepdims=True)
    probs = exp/exp_sum
    return probs
```

3.3.2 Derivative of the softmax function

We can show that the derivative of the **softmax** function with respect to any input can be written as

$$\frac{\partial \mathbf{p}_i}{\partial \mathbf{z}_j} = \begin{cases} \mathbf{p}_i(1-\mathbf{p}_j) & i = j \\ \mathbf{p}_i(-\mathbf{p}_j) & i \neq j. \end{cases}$$

More info here

3.3.3 Multiclass cross entropy loss function [5 pts]

We will minimize the cross entropy loss. You will use the true labels and predicted labels of a batch of N samples.

The multi-class cross entropy loss for i^{th} sample can be written as

$$Loss_i = -\sum_c \mathbf{1}(y_i = c)\log \mathbf{p}_c$$

where y_i is the true label and

$$\mathbf{1}(y_i = c) = \begin{cases} 1 & y_i = c \\ 0 & \text{otherwise} \end{cases}$$

is an indicator function.

We can find the average loss for a batch of N samples as $Loss = \frac{1}{N}\sum_{i=1}^{N} Loss_i$.

[24]:
```
def one_hot(X):
  # X -- N x 1 array

  X_int = train_y.astype(int)
  X_one_hot = np.eye(np.max(X_int)+1)[X_int]
  return X_one_hot
```

[25]:
```python
def MultiClassCrossEntropyLoss_2(Y_true, probs):

  # TODO
  # Write your code here

  # probs -- K x N array
  # Y_true -- 1 x N array
  # loss --  sum Loss_i over N samples
  Y_true = Y_true.astype(int)
  logprobs = -np.log(probs.T[range(Y_true.shape[0]),Y_true])
  l = np.sum(logprobs)
  loss = (1.0/Y_true.shape[0]) * l

  return loss
```

3.3.4 Derivative of the cross entropy loss

Let us assume that $\mathbf{p} = \mathrm{softmax}(\mathbf{z})$.

Note that the derivative of the loss w.r.t. $\mathbf{p}_j$ can be written as

$$\frac{\partial Loss_i}{\partial \mathbf{p}_j} = \begin{cases} -1/\mathbf{p}_j & j = y_i \\ 0 & j \neq y_i \end{cases}.$$

Note that we can use *total derivative* to compute the derivative of the loss for ith sample w.r.t. jth entry in $\mathbf{z}$ as

$$\frac{\partial Loss_i}{\partial \mathbf{z}_j} = \sum_c \frac{\partial Loss_i}{\partial \mathbf{p}_c} \frac{\partial \mathbf{p}_c}{\partial \mathbf{z}_j}.$$

From our discussion above, we know that the $\frac{\partial Loss_i}{\partial \mathbf{p}_c} = 0$ if $c \neq y_i$.

$$\begin{aligned} \frac{\partial Loss_i}{\partial \mathbf{z}_j} &= -\frac{1}{\mathbf{p}_c} \frac{\partial \mathbf{p}_c}{\partial \mathbf{z}_j} \\ &= \begin{cases} \mathbf{p}_j - 1 & j = y_i \\ \mathbf{p}_j & j \neq y_i. \end{cases} \end{aligned}$$

Therefore,

$$\delta^{(2)} = \nabla_{\mathbf{z}^{(2)}} Loss_i = \mathbf{p} - \mathbf{1}_{y_i}.$$

where $\mathbf{1}_{y_i}$ is a **one-hot vector** that has length K and is zero everywhere except 1 at index same as y_i.

3.3.5 Training data

Let us pick training data for multi-class classification.

Pick same number of images from each class for training and create arrays for input and output.

```
# train_x -- N x 784 array of training input
# train_y -- N x 1 array of labels
```

If you use 1000 images from each class N = 10000. You can increase the number of training samples if you like. You may also use unequal number of images in each class.

We also need to transpose the dimension of the data so that their size becomes $784 \times N$. It will be helpful to feed it to our model based on our notations.

[25]:

```

```

[26]:

```python
# Pick training samples
num_samples = 1000

# Training data
x = np.zeros((0,784))
y = np.zeros((0))
for label in range(10):
  x1 = x_train[y_train == label]
  x1 = x1[:num_samples]
  y1 = y_train[y_train == label]
  y1 = y1[:num_samples]

  x = np.concatenate((x,x1),axis=0)
  y = np.concatenate((y,y1),axis=0)

train_x = x
train_y = y
print("Training data shape:", train_x.shape)


# Test data
test_x = x_test
test_y = y_test
print("Test data shape:", test_x.shape)

# reshape data
train_x = train_x.T
test_x = test_x.T
print("Training data shape:", train_x.shape)
print("Training label shape:", train_y.shape)
print ()
print("Test data shape:", test_x.shape)
```

```
print("Test label shape:", test_y.shape)
```

```
Training data shape: (10000, 784)
Test data shape: (10000, 784)
Training data shape: (784, 10000)
Training label shape: (10000,)

Test data shape: (784, 10000)
Test label shape: (10000,)
```

3.3.6 Network Architecture

We will be using a two layer neural network in our experiment. The input layer has 784 nodes, the hidden layer will have 256 nodes and the output layer will have 10 nodes. First layer will have **sigmoid** activation and second layer will have **softmax** activation.

The equations for feedforward operation will be as follows.

$$\mathbf{z}^{(1)} = W^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\mathbf{y}^{(1)} = \text{sigmoid}(\mathbf{z}^{(1)})\mathbf{z}^{(2)} = W^{(2)}\mathbf{y}^{(1)} + \mathbf{b}^{(2)}\mathbf{p} = \mathbf{y}^{(2)} = \text{softmax}(\mathbf{z}^{(2)})$$

where $\mathbf{x} \in \mathbb{R}^{784}$ is the input layer, $\mathbf{y}^{(1)} \in \mathbb{R}^{256}$ is the hidden layer, $\mathbf{y}^{(2)} \in \mathbb{R}$ is the output layer, $W^{(1)} \in \mathbb{R}^{256\times784}$ is the first layer weights, $W^{(2)} \in \mathbb{R}^{10\times256}$ is the second layer weights, $\mathbf{b}^{(1)} \in \mathbb{R}^{256}$ is the first layer bias, $\mathbf{b}^{(2)} \in \mathbb{R}^{10}$ is the second layer bias vector.

3.3.7 Network initialization [5 pts]

We initialize the weights for $W^{(1)}$ and $W^{(2)}$ with random values drawn from normal distribution with zero mean and 0.01 standard deviation. We will initialize bias vectors $\mathbf{b}^{(1)}$ and $\mathbf{b}^{(2)}$ with zero values.

We can fix the seed for random initialization for reproducibility.

[27]:
```
def sigmoid_2(Z):
    # Input: Z -- numpy.ndarray
    # TODO
    Y = (1 / (1 + np.exp(-Z)))
    return Y
```

[28]:
```
def TwoLayerNetwork_2(layer_dims=[784,256,10],random_state=3):
    # TODO
    # Your code goes here

    # Fix the seed
    np.random.seed(random_state)

    #Initialize the weights
```

```
    mean = 0
    std = 0.01

    params = {}
    params['w1'] = np.random.normal(mean, std,␣
↪size=(layer_dims[1],layer_dims[0]))
    params['b1'] = np.zeros(layer_dims[1])
    params['w2'] = np.random.normal(mean, std,␣
↪size=(layer_dims[2],layer_dims[1]))
    params['b2'] = np.zeros(layer_dims[2])

    return params
```

3.3.8 Forward propagation

Next, we will write the code for the forward pass for two layer network. Each layer consists of an affine function (fully-connected layer) followed by an activation function. You wil also return the intermediate results $(\mathbf{x}, \mathbf{z}^{(1)}, \mathbf{y}^{(1)}, \mathbf{z}^{(2)})$ in addition to final output $(\mathbf{y}^{(2)})$. You will need the intermediate outputs for the backpropagation step.

[29]:
```
def forward_2(X, params):

    # TODO
    # Write your codes here

    # X -- 784 x N array
    # params --
      # W1 -- 256 x 784 matrix
      # b1 -- 256 x 1 vector
      # W2 -- 10 x 256 matrix
      # b2 -- 10 x 1 scalar
    # probs -- 10 x N output

    intermediate = {}
    intermediate['X'] = X
    intermediate['z1'] = params['w1'].dot(intermediate['X'])+params['b1'].
↪reshape(params['b1'].shape[0], 1)
    intermediate['Y1'] = sigmoid_2(intermediate['z1'])
    intermediate['z2'] = params['w2'].dot(intermediate['Y1'])+params['b2'].
↪reshape(params['b2'].shape[0], 1)
    Y2 = softmax_2(intermediate['z2'])

    return Y2, intermediate
```

3.3.9 Backpropagation step [10 pts]

Now we will implement the backpropagation step for the two layer neural network using softmax layer and loss function.

You will need the gradient of the Loss w.r.t. $W^{(l)}, \mathbf{b}^{(l)}$ for $l = 1, 2$ for all the training samples.

We saw that we can write the gradient of Loss with respect to $W^{(l)}, \mathbf{b}^{(l)}$ for a single sample as

$$\nabla_{W^{(l)}} Loss_i = \delta^{(l)} \mathbf{y}^{(l-1)T},$$

$$\nabla_{\mathbf{b}^{(l)}} Loss_i = \delta^{(l)},$$

where

$$\delta^{(l)} = \nabla_{\mathbf{z}^{(l)}} Loss = \nabla_{\mathbf{y}^{(l)}} Loss \odot \varphi'(\mathbf{z}^{(l)}).$$

We saw above that for an ith sample, $\delta^{(2)} = \nabla_{\mathbf{z}^{(2)}} Loss_i = \mathbf{p} - \mathbf{1}_{y_i}$, where $\mathbf{1}_{y_i}$ is a **one-hot vector** that has length K and is zero everywhere except 1 at index same as y_i and $\mathbf{p}$ is the outpu probability vector for the ith sample.

Once we have the gradients $\nabla_{W^{(l)}} Loss_i, \nabla_{\mathbf{b}^{(l)}} Loss_i$ for all i. We can compute their average to compute the gradient of the total loss function as

$$\nabla_{W^{(l)}} Loss = \frac{1}{N} \sum_i \nabla_{W^{(l)}} Loss_i,$$

$$\nabla_{\mathbf{b}^{(l)}} Loss = \frac{1}{N} \sum_i \nabla_{\mathbf{b}^{(l)}} Loss_i.$$

Please refer to the slides and lectures for more details.

[30]:
```python
def backward_2(Y_true, probs, intermediate, params):

    # Inputs:
      # Y_true -- true labels
      # probs -- 10 x N output of the last layer
      # intermediate -- X, Z1, Y1, Z2
      # params -- W1, b1, W2, b2

    # Outputs:
      # grads -- [grad_W1, grad_b1, grad_W2, grad_b2]

    # TODO
    # Write your codes here

    grads = {}

    train_y_one_hot = one_hot(train_y)
```

```
    train_y_one_hot_transposed = train_y_one_hot.T

    delta_2 = probs - train_y_one_hot_transposed
    grads['w2'] = delta_2.dot(intermediate['Y1'].T)
    grads['b2'] = np.sum(delta_2,axis=1)

    delta_1 = grads['w2'].T.dot(delta_2) * (sigmoid_2(intermediate['z1'])*(1 -␣
↪sigmoid_2(intermediate['z1'])))
    grads['w1'] = delta_1.dot(intermediate['X'].T)
    grads['b1'] = np.sum(delta_1,axis=1)

    return grads
```

[31]:
```
def GD_2(params, grads, learning_rate):

    # updated params = old params - learning rate * gradient of Loss computed␣
↪at old params
    # TODO
    # Write your codes here

    params['w2'] = params['w2'] - learning_rate*grads['w2']
    params['b2'] = params['b2'] - learning_rate*grads['b2']
    params['w1'] = params['w1'] - learning_rate*grads['w1']
    params['b1'] = params['b1'] - learning_rate*grads['b1']


    return params
```

[32]:
```
def predict_2(train_x,params):
  y_pred,_ = forward_2(train_x,params)
  return np.argmax(y_pred,axis=0)

def accuracy_2(y_true, y_pred):
  aa = np.sum(y_pred == y_true)
  return aa / len(y_true)
```

3.3.10 Train the model [5 pts]

We will use the forward and backward functions defined above with the same optimizer defined in the previous question to train our multi-class classificaiton model.

We will specify the number of nodes in the layers, number of epochs and learning rate and initialize the network

[33]:
```
layer_dims = [train_x.shape[0],256,10]
epochs = 200
lr = 0.000017
```

```
params = TwoLayerNetwork_2(layer_dims,29)
```

Then we train the network for the number of epochs specified above. In every epoch, we will do the following: 1. Calculate the forward pass to get estimated labels. 2. Use the estimated labels calculate loss. We will be recording loss for every epoch. 3. Use backpropagation to calculate gradients. 4. Use gradient descent to update the weights and biases.

You should store the loss value after every epoch in an array `loss_history` and print the loss value after every few epochs (say 20).

[34]:
```
# TODO
# Write your codes here

loss_history = []
train_acc = []
test_acc = []
for i in range(epochs):

  print ('epoch ==> ', i)

  probs, intermediate = forward_2(train_x, params)
  loss = MultiClassCrossEntropyLoss_2(train_y, probs)
  loss_history.append(loss)

  train_pred = predict_2(train_x, params)
  test_pred = predict_2(test_x, params)

  train_accuracy = accuracy_2(train_y, train_pred)
  test_accuracy = accuracy_2(test_y,test_pred)

  print (' loss ', loss)
  print (' train_accuracy ', train_accuracy)
  print (' test_accuracy ', test_accuracy)

  test_acc.append(test_accuracy)
  train_acc.append(train_accuracy)

  grads = backward_2(train_y, probs, intermediate, params)
  params = GD_2(params, grads, lr)
```

```
epoch ==>  0
 loss  2.316381955048051
 train_accuracy  0.1103
 test_accuracy  0.1054
epoch ==>  1
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:4: RuntimeWarning:
overflow encountered in exp
```

```
  after removing the cwd from sys.path.
 loss  2.33358934713162
 train_accuracy  0.0824
 test_accuracy  0.0719
epoch ==>  2
 loss  2.268950454719662
 train_accuracy  0.2122
 test_accuracy  0.2197
epoch ==>  3
 loss  2.2089639722978704
 train_accuracy  0.4401
 test_accuracy  0.4392
epoch ==>  4
 loss  2.1529603012856353
 train_accuracy  0.4372
 test_accuracy  0.4456
epoch ==>  5
 loss  2.0996889967800647
 train_accuracy  0.5708
 test_accuracy  0.5791
epoch ==>  6
 loss  2.049430664836007
 train_accuracy  0.5683
 test_accuracy  0.5755
epoch ==>  7
 loss  2.002746173593738
 train_accuracy  0.5987
 test_accuracy  0.6076
epoch ==>  8
 loss  1.9581468402876112
 train_accuracy  0.6172
 test_accuracy  0.6247
epoch ==>  9
 loss  1.9161409935903577
 train_accuracy  0.6163
 test_accuracy  0.6245
epoch ==>  10
 loss  1.876342929349308
 train_accuracy  0.6301
 test_accuracy  0.6391
epoch ==>  11
 loss  1.838876707473756
 train_accuracy  0.6308
 test_accuracy  0.6381
epoch ==>  12
 loss  1.8032444869591369
 train_accuracy  0.6401
```

```
 test_accuracy  0.65
epoch ==>  13
 loss  1.7693691518868269
 train_accuracy  0.641
 test_accuracy  0.6453
epoch ==>  14
 loss  1.737370070181499
 train_accuracy  0.6491
 test_accuracy  0.6562
epoch ==>  15
 loss  1.7070734148228996
 train_accuracy  0.6497
 test_accuracy  0.6544
epoch ==>  16
 loss  1.6782778388313615
 train_accuracy  0.6552
 test_accuracy  0.6626
epoch ==>  17
 loss  1.6515525250806764
 train_accuracy  0.6589
 test_accuracy  0.6616
epoch ==>  18
 loss  1.6255815469703403
 train_accuracy  0.6618
 test_accuracy  0.6659
epoch ==>  19
 loss  1.6008622681593978
 train_accuracy  0.6652
 test_accuracy  0.6661
epoch ==>  20
 loss  1.5773126503686152
 train_accuracy  0.6671
 test_accuracy  0.6715
epoch ==>  21
 loss  1.5548573635666114
 train_accuracy  0.6706
 test_accuracy  0.6712
epoch ==>  22
 loss  1.5334260944652593
 train_accuracy  0.6717
 test_accuracy  0.6757
epoch ==>  23
 loss  1.512954511155078
 train_accuracy  0.6731
 test_accuracy  0.6753
epoch ==>  24
 loss  1.4933825205031965
 train_accuracy  0.6752
```

```
 test_accuracy  0.678
epoch ==>  25
 loss  1.474670540497931
 train_accuracy  0.6778
 test_accuracy  0.679
epoch ==>  26
 loss  1.4567398504272249
 train_accuracy  0.6817
 test_accuracy  0.6815
epoch ==>  27
 loss  1.4396923468665745
 train_accuracy  0.6822
 test_accuracy  0.6836
epoch ==>  28
 loss  1.4232657082344708
 train_accuracy  0.6845
 test_accuracy  0.6875
epoch ==>  29
 loss  1.407440580954413
 train_accuracy  0.6861
 test_accuracy  0.6886
epoch ==>  30
 loss  1.392237727416691
 train_accuracy  0.6873
 test_accuracy  0.6909
epoch ==>  31
 loss  1.377622074151682
 train_accuracy  0.6886
 test_accuracy  0.6923
epoch ==>  32
 loss  1.3635608013571154
 train_accuracy  0.6907
 test_accuracy  0.6934
epoch ==>  33
 loss  1.3500235281511406
 train_accuracy  0.6916
 test_accuracy  0.694
epoch ==>  34
 loss  1.3369818201719998
 train_accuracy  0.6936
 test_accuracy  0.6946
epoch ==>  35
 loss  1.324409277415533
 train_accuracy  0.6949
 test_accuracy  0.6957
epoch ==>  36
 loss  1.3122811664748997
 train_accuracy  0.6965
```

```
 test_accuracy  0.6969
epoch ==>  37
 loss  1.3005749851361479
 train_accuracy  0.6971
 test_accuracy  0.6986
epoch ==>  38
 loss  1.2892688883782688
 train_accuracy  0.6988
 test_accuracy  0.7007
epoch ==>  39
 loss  1.2783226581020772
 train_accuracy  0.6994
 test_accuracy  0.7011
epoch ==>  40
 loss  1.2677569706580478
 train_accuracy  0.701
 test_accuracy  0.7014
epoch ==>  41
 loss  1.2575346160053216
 train_accuracy  0.7023
 test_accuracy  0.703
epoch ==>  42
 loss  1.2475841923669062
 train_accuracy  0.7032
 test_accuracy  0.7045
epoch ==>  43
 loss  1.238446135004203
 train_accuracy  0.7036
 test_accuracy  0.7059
epoch ==>  44
 loss  1.2291591681725227
 train_accuracy  0.7042
 test_accuracy  0.7062
epoch ==>  45
 loss  1.2200739426921263
 train_accuracy  0.7051
 test_accuracy  0.7075
epoch ==>  46
 loss  1.2110080532450562
 train_accuracy  0.7058
 test_accuracy  0.7082
epoch ==>  47
 loss  1.2029848347892935
 train_accuracy  0.7062
 test_accuracy  0.7081
epoch ==>  48
 loss  1.1947960747220223
 train_accuracy  0.7071
```

```
 test_accuracy  0.7095
epoch ==>  49
 loss  1.1868006549584782
 train_accuracy  0.708
 test_accuracy  0.7098
epoch ==>  50
 loss  1.1790305642860208
 train_accuracy  0.7089
 test_accuracy  0.7101
epoch ==>  51
 loss  1.1714761366887363
 train_accuracy  0.7102
 test_accuracy  0.7111
epoch ==>  52
 loss  1.164128284335001
 train_accuracy  0.7106
 test_accuracy  0.7121
epoch ==>  53
 loss  1.1569784288584675
 train_accuracy  0.7114
 test_accuracy  0.712
epoch ==>  54
 loss  1.1500184613116178
 train_accuracy  0.7126
 test_accuracy  0.7125
epoch ==>  55
 loss  1.1432407098576423
 train_accuracy  0.7139
 test_accuracy  0.7131
epoch ==>  56
 loss  1.1366382101936114
 train_accuracy  0.7144
 test_accuracy  0.7145
epoch ==>  57
 loss  1.1301849516492355
 train_accuracy  0.7145
 test_accuracy  0.7151
epoch ==>  58
 loss  1.1239115439242005
 train_accuracy  0.7151
 test_accuracy  0.7156
epoch ==>  59
 loss  1.117793466005576
 train_accuracy  0.7162
 test_accuracy  0.7164
epoch ==>  60
 loss  1.1118248266053397
 train_accuracy  0.7173
```

```
 test_accuracy  0.717
epoch ==>  61
 loss  1.1060000287261953
 train_accuracy  0.7178
 test_accuracy  0.7177
epoch ==>  62
 loss  1.1003137505777942
 train_accuracy  0.7189
 test_accuracy  0.7179
epoch ==>  63
 loss  1.0947609329006858
 train_accuracy  0.7205
 test_accuracy  0.7192
epoch ==>  64
 loss  1.0893370085153173
 train_accuracy  0.7207
 test_accuracy  0.7199
epoch ==>  65
 loss  1.0840369262244265
 train_accuracy  0.7215
 test_accuracy  0.7204
epoch ==>  66
 loss  1.078856475658695
 train_accuracy  0.7219
 test_accuracy  0.7209
epoch ==>  67
 loss  1.073791524205537
 train_accuracy  0.7224
 test_accuracy  0.7214
epoch ==>  68
 loss  1.068838092178735
 train_accuracy  0.7229
 test_accuracy  0.7221
epoch ==>  69
 loss  1.0639923810082927
 train_accuracy  0.7235
 test_accuracy  0.7227
epoch ==>  70
 loss  1.0592507629686398
 train_accuracy  0.7238
 test_accuracy  0.7224
epoch ==>  71
 loss  1.0546097718848206
 train_accuracy  0.7238
 test_accuracy  0.7229
epoch ==>  72
 loss  1.050066094198404
 train_accuracy  0.7246
```

```
 test_accuracy  0.7231
epoch ==>  73
 loss  1.0456165607948367
 train_accuracy  0.7252
 test_accuracy  0.723
epoch ==>  74
 loss  1.0412581392025906
 train_accuracy  0.7251
 test_accuracy  0.7237
epoch ==>  75
 loss  1.0369879263970516
 train_accuracy  0.7258
 test_accuracy  0.7242
epoch ==>  76
 loss  1.0328031419648844
 train_accuracy  0.7263
 test_accuracy  0.7247
epoch ==>  77
 loss  1.0287011217583826
 train_accuracy  0.7279
 test_accuracy  0.7257
epoch ==>  78
 loss  1.0246793118864141
 train_accuracy  0.7282
 test_accuracy  0.7261
epoch ==>  79
 loss  1.0207352631101336
 train_accuracy  0.7288
 test_accuracy  0.7263
epoch ==>  80
 loss  1.0168666255470462
 train_accuracy  0.7289
 test_accuracy  0.7268
epoch ==>  81
 loss  1.0130711437197422
 train_accuracy  0.7292
 test_accuracy  0.7271
epoch ==>  82
 loss  1.0093466519053838
 train_accuracy  0.7293
 test_accuracy  0.7272
epoch ==>  83
 loss  1.0056910699330817
 train_accuracy  0.7293
 test_accuracy  0.7273
epoch ==>  84
 loss  1.0021024015486655
 train_accuracy  0.73
```

```
 test_accuracy  0.7276
epoch ==>  85
 loss  0.9985800896570486
 train_accuracy  0.7315
 test_accuracy  0.7279
epoch ==>  86
 loss  0.9952172276559208
 train_accuracy  0.7321
 test_accuracy  0.729
epoch ==>  87
 loss  0.9918186064819434
 train_accuracy  0.7323
 test_accuracy  0.7292
epoch ==>  88
 loss  0.9884795868231142
 train_accuracy  0.7327
 test_accuracy  0.7294
epoch ==>  89
 loss  0.9851985557712116
 train_accuracy  0.7333
 test_accuracy  0.7301
epoch ==>  90
 loss  0.9819739227981101
 train_accuracy  0.7339
 test_accuracy  0.7303
epoch ==>  91
 loss  0.9788041578689732
 train_accuracy  0.7343
 test_accuracy  0.7309
epoch ==>  92
 loss  0.9756877871541947
 train_accuracy  0.7344
 test_accuracy  0.7312
epoch ==>  93
 loss  0.9726233899902528
 train_accuracy  0.7351
 test_accuracy  0.7317
epoch ==>  94
 loss  0.9696095963770437
 train_accuracy  0.7361
 test_accuracy  0.7328
epoch ==>  95
 loss  0.9666450846724718
 train_accuracy  0.7365
 test_accuracy  0.7331
epoch ==>  96
 loss  0.9637285794276245
 train_accuracy  0.7369
```

```
 test_accuracy  0.733
epoch ==>  97
 loss  0.960858849394059
 train_accuracy  0.7374
 test_accuracy  0.7336
epoch ==>  98
 loss  0.9580347055921721
 train_accuracy  0.7378
 test_accuracy  0.7338
epoch ==>  99
 loss  0.9552549995309619
 train_accuracy  0.738
 test_accuracy  0.7339
epoch ==>  100
 loss  0.9525186214755342
 train_accuracy  0.7384
 test_accuracy  0.7342
epoch ==>  101
 loss  0.9498244988401111
 train_accuracy  0.7385
 test_accuracy  0.7341
epoch ==>  102
 loss  0.9471715946303616
 train_accuracy  0.7387
 test_accuracy  0.7347
epoch ==>  103
 loss  0.9445589059877525
 train_accuracy  0.7389
 test_accuracy  0.7347
epoch ==>  104
 loss  0.9419854627856487
 train_accuracy  0.7391
 test_accuracy  0.7347
epoch ==>  105
 loss  0.9394503263088855
 train_accuracy  0.7393
 test_accuracy  0.735
epoch ==>  106
 loss  0.9369525879853656
 train_accuracy  0.7395
 test_accuracy  0.7348
epoch ==>  107
 loss  0.934491368187221
 train_accuracy  0.74
 test_accuracy  0.7354
epoch ==>  108
 loss  0.9320658150823363
 train_accuracy  0.7402
```

```
 test_accuracy  0.7357
epoch ==>  109
 loss  0.9296751035452644
 train_accuracy  0.7403
 test_accuracy  0.7358
epoch ==>  110
 loss  0.9273184341163244
 train_accuracy  0.7406
 test_accuracy  0.7359
epoch ==>  111
 loss  0.9249950320152761
 train_accuracy  0.7411
 test_accuracy  0.736
epoch ==>  112
 loss  0.9227041462157187
 train_accuracy  0.7414
 test_accuracy  0.7363
epoch ==>  113
 loss  0.9204450487077666
 train_accuracy  0.7416
 test_accuracy  0.7367
epoch ==>  114
 loss  0.9182170386187063
 train_accuracy  0.7416
 test_accuracy  0.7371
epoch ==>  115
 loss  0.9160219707938301
 train_accuracy  0.7422
 test_accuracy  0.7371
epoch ==>  116
 loss  0.9138541857682715
 train_accuracy  0.7424
 test_accuracy  0.7372
epoch ==>  117
 loss  0.9117154874092712
 train_accuracy  0.7431
 test_accuracy  0.7391
epoch ==>  118
 loss  0.9096052490114622
 train_accuracy  0.7435
 test_accuracy  0.7395
epoch ==>  119
 loss  0.90752286239506
 train_accuracy  0.744
 test_accuracy  0.7397
epoch ==>  120
 loss  0.9054677369185667
 train_accuracy  0.7442
```

```
 test_accuracy  0.7403
epoch ==>  121
 loss  0.9034392988324155
 train_accuracy  0.7448
 test_accuracy  0.7406
epoch ==>  122
 loss  0.9014369906487643
 train_accuracy  0.7449
 test_accuracy  0.741
epoch ==>  123
 loss  0.8994602703145462
 train_accuracy  0.7448
 test_accuracy  0.7413
epoch ==>  124
 loss  0.8975086004610432
 train_accuracy  0.7453
 test_accuracy  0.7419
epoch ==>  125
 loss  0.8955816204391475
 train_accuracy  0.7455
 test_accuracy  0.7423
epoch ==>  126
 loss  0.8936785774635655
 train_accuracy  0.7457
 test_accuracy  0.7431
epoch ==>  127
 loss  0.8917991023311188
 train_accuracy  0.746
 test_accuracy  0.7432
epoch ==>  128
 loss  0.8899427240891604
 train_accuracy  0.746
 test_accuracy  0.7433
epoch ==>  129
 loss  0.8881089845236649
 train_accuracy  0.7465
 test_accuracy  0.7439
epoch ==>  130
 loss  0.8862974377214538
 train_accuracy  0.7468
 test_accuracy  0.7447
epoch ==>  131
 loss  0.8845076496617019
 train_accuracy  0.7471
 test_accuracy  0.7451
epoch ==>  132
 loss  0.8827391978174054
 train_accuracy  0.7472
```

```
 test_accuracy  0.7456
epoch ==>  133
 loss  0.8809916707774401
 train_accuracy  0.7472
 test_accuracy  0.7455
epoch ==>  134
 loss  0.8792646678801109
 train_accuracy  0.7478
 test_accuracy  0.7457
epoch ==>  135
 loss  0.8775577988635022
 train_accuracy  0.7482
 test_accuracy  0.7461
epoch ==>  136
 loss  0.8758706835276877
 train_accuracy  0.7482
 test_accuracy  0.7463
epoch ==>  137
 loss  0.8742029514112557
 train_accuracy  0.7486
 test_accuracy  0.7462
epoch ==>  138
 loss  0.8725542414793689
 train_accuracy  0.7486
 test_accuracy  0.7464
epoch ==>  139
 loss  0.870924201824372
 train_accuracy  0.7486
 test_accuracy  0.7466
epoch ==>  140
 loss  0.8693124893773071
 train_accuracy  0.7488
 test_accuracy  0.7468
epoch ==>  141
 loss  0.8677187696306236
 train_accuracy  0.7492
 test_accuracy  0.7471
epoch ==>  142
 loss  0.8661427163710521
 train_accuracy  0.7495
 test_accuracy  0.7474
epoch ==>  143
 loss  0.8645840114225863
 train_accuracy  0.7496
 test_accuracy  0.7475
epoch ==>  144
 loss  0.8630423443988592
 train_accuracy  0.7497
```

```
 test_accuracy  0.7478
epoch ==>  145
 loss  0.8615174124647096
 train_accuracy  0.7498
 test_accuracy  0.7485
epoch ==>  146
 loss  0.8600089201064083
 train_accuracy  0.7499
 test_accuracy  0.7488
epoch ==>  147
 loss  0.8585165789102701
 train_accuracy  0.7503
 test_accuracy  0.7491
epoch ==>  148
 loss  0.8570401073492371
 train_accuracy  0.7505
 test_accuracy  0.7495
epoch ==>  149
 loss  0.8555792305771401
 train_accuracy  0.7512
 test_accuracy  0.7496
epoch ==>  150
 loss  0.8541336802302906
 train_accuracy  0.7512
 test_accuracy  0.7502
epoch ==>  151
 loss  0.8527031942361195
 train_accuracy  0.7512
 test_accuracy  0.7508
epoch ==>  152
 loss  0.8512875166285607
 train_accuracy  0.7517
 test_accuracy  0.7512
epoch ==>  153
 loss  0.8498863973699079
 train_accuracy  0.7519
 test_accuracy  0.7519
epoch ==>  154
 loss  0.8484995921788825
 train_accuracy  0.7521
 test_accuracy  0.7523
epoch ==>  155
 loss  0.8471268623646604
 train_accuracy  0.7528
 test_accuracy  0.7529
epoch ==>  156
 loss  0.8457679746666152
 train_accuracy  0.7529
```

```
 test_accuracy  0.7529
epoch ==>  157
 loss  0.8444227010995569
 train_accuracy  0.7529
 test_accuracy  0.7531
epoch ==>  158
 loss  0.8430908188042382
 train_accuracy  0.7532
 test_accuracy  0.7532
epoch ==>  159
 loss  0.8417721099029292
 train_accuracy  0.7533
 test_accuracy  0.7534
epoch ==>  160
 loss  0.8404663613598549
 train_accuracy  0.7534
 test_accuracy  0.7535
epoch ==>  161
 loss  0.8391733648463108
 train_accuracy  0.7531
 test_accuracy  0.7536
epoch ==>  162
 loss  0.8378929166102708
 train_accuracy  0.7532
 test_accuracy  0.7539
epoch ==>  163
 loss  0.8366248173503189
 train_accuracy  0.7534
 test_accuracy  0.7541
epoch ==>  164
 loss  0.8353688720937359
 train_accuracy  0.7536
 test_accuracy  0.7545
epoch ==>  165
 loss  0.8341248900785843
 train_accuracy  0.7539
 test_accuracy  0.7545
epoch ==>  166
 loss  0.8328926846396452
 train_accuracy  0.754
 test_accuracy  0.7547
epoch ==>  167
 loss  0.8316720730980603
 train_accuracy  0.7545
 test_accuracy  0.755
epoch ==>  168
 loss  0.8304628766545495
 train_accuracy  0.7546
```

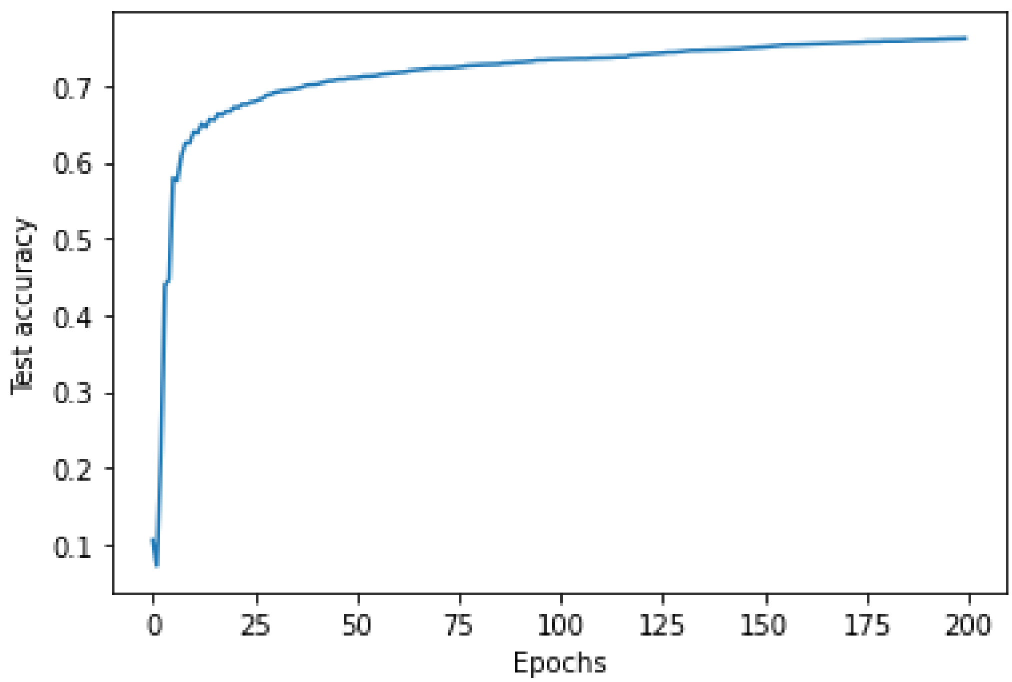
```
 test_accuracy  0.7552
epoch ==>  169
 loss  0.829264920286075
 train_accuracy  0.7548
 test_accuracy  0.7553
epoch ==>  170
 loss  0.8280780326458502
 train_accuracy  0.7549
 test_accuracy  0.7555
epoch ==>  171
 loss  0.8269020459665857
 train_accuracy  0.7553
 test_accuracy  0.7556
epoch ==>  172
 loss  0.8257367959669235
 train_accuracy  0.7554
 test_accuracy  0.756
epoch ==>  173
 loss  0.8245821217610417
 train_accuracy  0.7558
 test_accuracy  0.7561
epoch ==>  174
 loss  0.8234378657715578
 train_accuracy  0.7564
 test_accuracy  0.7565
epoch ==>  175
 loss  0.8223038736461314
 train_accuracy  0.7566
 test_accuracy  0.7571
epoch ==>  176
 loss  0.8211799941789776
 train_accuracy  0.7568
 test_accuracy  0.7572
epoch ==>  177
 loss  0.8200660792408728
 train_accuracy  0.7571
 test_accuracy  0.7573
epoch ==>  178
 loss  0.8189619837299575
 train_accuracy  0.7571
 test_accuracy  0.7573
epoch ==>  179
 loss  0.8178675655954754
 train_accuracy  0.7576
 test_accuracy  0.7579
epoch ==>  180
 loss  0.8167826862487859
 train_accuracy  0.7576
```

```
 test_accuracy  0.7579
epoch ==>  181
 loss  0.8157072150156606
 train_accuracy  0.7578
 test_accuracy  0.7578
epoch ==>  182
 loss  0.8146413500892287
 train_accuracy  0.7578
 test_accuracy  0.758
epoch ==>  183
 loss  0.8135886300250562
 train_accuracy  0.7578
 test_accuracy  0.7581
epoch ==>  184
 loss  0.8125406344572812
 train_accuracy  0.7578
 test_accuracy  0.7583
epoch ==>  185
 loss  0.8115015192120401
 train_accuracy  0.7579
 test_accuracy  0.7587
epoch ==>  186
 loss  0.8104711603979031
 train_accuracy  0.758
 test_accuracy  0.7589
epoch ==>  187
 loss  0.8094494365126617
 train_accuracy  0.7582
 test_accuracy  0.7593
epoch ==>  188
 loss  0.8084362283751011
 train_accuracy  0.7582
 test_accuracy  0.7594
epoch ==>  189
 loss  0.8074314190687225
 train_accuracy  0.7583
 test_accuracy  0.7594
epoch ==>  190
 loss  0.8064348938873295
 train_accuracy  0.7586
 test_accuracy  0.7596
epoch ==>  191
 loss  0.8054465402823959
 train_accuracy  0.7586
 test_accuracy  0.7597
epoch ==>  192
 loss  0.8044662478118809
 train_accuracy  0.7586
```

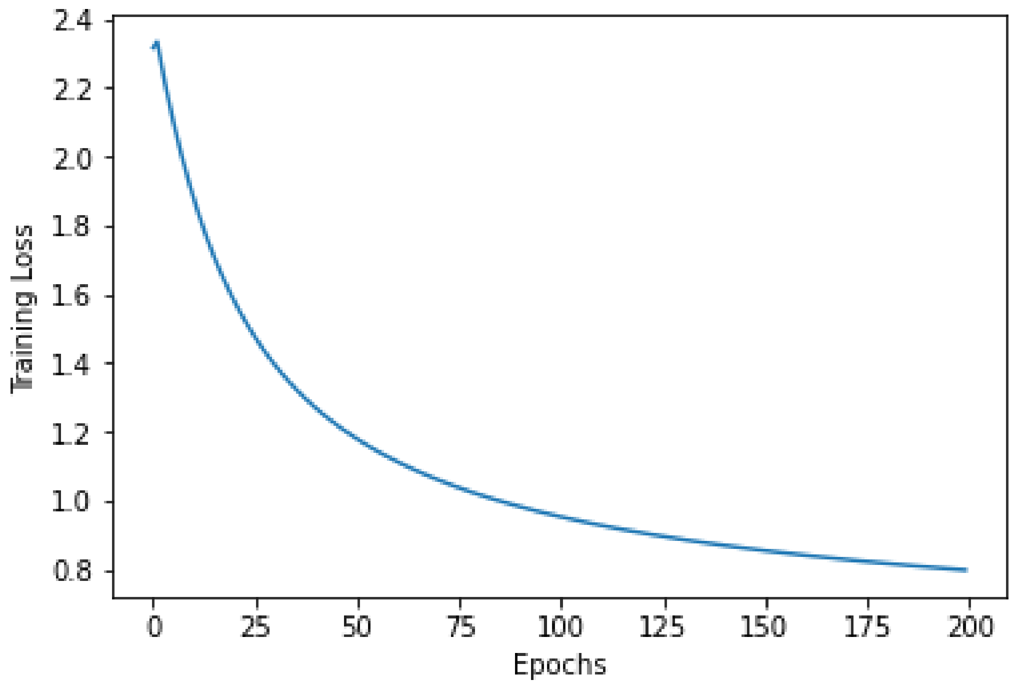
```
 test_accuracy  0.7598
epoch ==>  193
 loss  0.8034939080905552
 train_accuracy  0.7587
 test_accuracy  0.7604
epoch ==>  194
 loss  0.802529414741708
 train_accuracy  0.7588
 test_accuracy  0.7604
epoch ==>  195
 loss  0.801572663350232
 train_accuracy  0.7593
 test_accuracy  0.7608
epoch ==>  196
 loss  0.8006235514170134
 train_accuracy  0.7595
 test_accuracy  0.7608
epoch ==>  197
 loss  0.7996819783145992
 train_accuracy  0.7595
 test_accuracy  0.761
epoch ==>  198
 loss  0.7987478452440916
 train_accuracy  0.7599
 test_accuracy  0.7611
epoch ==>  199
 loss  0.7978210551932348
 train_accuracy  0.7601
 test_accuracy  0.7615
```

Now we will plot the recorded loss values vs epochs. We will observe the training loss decreasing with the epochs.

[35]:
```
plt.figure()
# plt.plot(train_acc)
plt.plot(test_acc)
plt.xlabel("Epochs")
plt.ylabel("Test accuracy")
plt.show()
```



[36]:
```
plt.figure()
plt.plot(loss_history)
plt.xlabel("Epochs")
plt.ylabel("Training Loss")
plt.show()
```



3.3.11 Evaluation on test data [5 pts]

Now we will be evaluating the accuracy we get from the trained model. We feed training data and test data to the forward model along with the trained parameters.

Note that, we need to convert the (probability) output of the forward pass into labels before evaluating accuracy. We can assign label based on the maximum probability.

We assign estimated labels

$$\hat{y}_i = \arg\max_c \mathbf{p}_c$$

for every probility vector.

[37]:
```
y_train_pred = predict_2(train_x, params)
train_accuracy = accuracy_2(train_y, y_train_pred)


print("Training accuracy:",train_accuracy)

y_test_pred = predict_2(test_x, params)
test_accuracy = accuracy_2(test_y, y_test_pred)

print("Test accuracy:",test_accuracy)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:4: RuntimeWarning:
overflow encountered in exp
  after removing the cwd from sys.path.
```

```
Training accuracy: 0.7605
Test accuracy: 0.7617
```

3.3.12 Visualize some of the correct/miscalassified images [optional]

Now we will look at some images from training and test sets that were misclassified.

Training set. Pick example from each class that are correcly and incorreclty classified. True/False Positive/Negatives

Test set. Pick examples from each class that are correcly and incorreclty classified. True/False Positive/Negatives

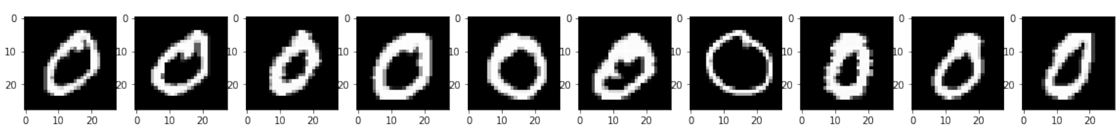
[38]:
```
# TODO
# Training set
print("Training set examples for true/false positive/negative")
Y_hat = predict_2(train_x, params)
Y_hat = Y_hat.reshape(-1)

positive = []
negative = []
false_positive = []
false_negative = []

idx = 0

for y_hat,y_true,x_i in zip(Y_hat, train_y, train_x.T):
  idx += 1
  if y_hat == y_true:
      positive.append(x_i)
  else :
    negative.append(x_i)


positive = np.array(positive)
negative = np.array(negative)

print ('positive', positive.shape)
print ('negative', negative.shape)


print (' --------- Positives -----------')
print ()
```

```
n_img=10
plt.figure(figsize=(n_img*2,2))
plt.gray()
for i in range(n_img):
    plt.subplot(1,n_img,i+1)
    plt.imshow(positive[i].reshape(28,28))
plt.show()

print ()
print (' --------- Negatives -----------')
print ()


plt.figure(figsize=(n_img*2,2))
plt.gray()
for i in range(n_img):
    plt.subplot(1,n_img,i+1)
    plt.imshow(negative[i+300].reshape(28,28))
plt.show()
```

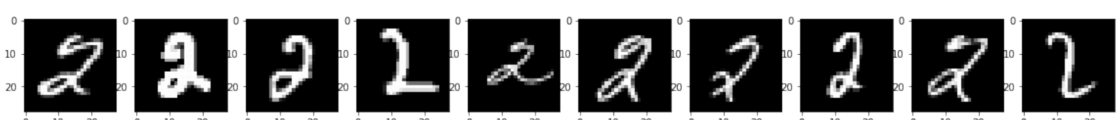
```
Training set examples for true/false positive/negative
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:4: RuntimeWarning:
overflow encountered in exp
  after removing the cwd from sys.path.
```

```
positive (7605, 784)
negative (2395, 784)
 --------- Positives -----------
```



```
 --------- Negatives -----------
```



[39]:

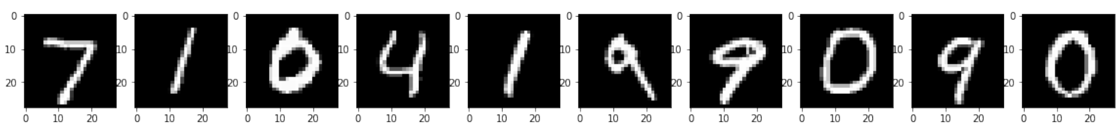
```python
# TODO
# Training set
print("Training set examples for true/false positive/negative")
Y_hat = predict_2(test_x, params)
Y_hat = Y_hat.reshape(-1)

positive = []
negative = []
false_positive = []
false_negative = []

idx = 0

for y_hat,y_true,x_i in zip(Y_hat, test_y, test_x.T):
  idx += 1
  if y_hat == y_true:
      positive.append(x_i)
  else :
    negative.append(x_i)


positive = np.array(positive)
negative = np.array(negative)

print ('positive', positive.shape)
print ('negative', negative.shape)


print (' --------- Positives -----------')
print ()

n_img=10
plt.figure(figsize=(n_img*2,2))
plt.gray()
for i in range(n_img):
    plt.subplot(1,n_img,i+1)
    plt.imshow(positive[i].reshape(28,28))
plt.show()

print ()
print (' --------- Negatives -----------')
print ()


plt.figure(figsize=(n_img*2,2))
plt.gray()
for i in range(n_img):
```

```
    plt.subplot(1,n_img,i+1)
    plt.imshow(negative[i+300].reshape(28,28))
plt.show()
```

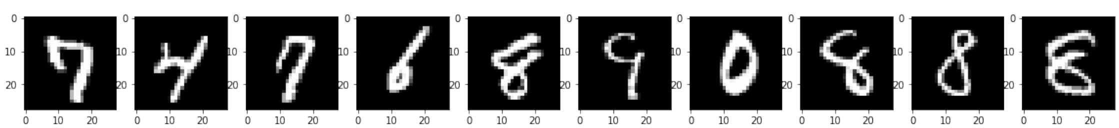
```
Training set examples for true/false positive/negative
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:4: RuntimeWarning:
overflow encountered in exp
  after removing the cwd from sys.path.
```

```
positive (7617, 784)
negative (2383, 784)
 --------- Positives -----------
```



```
 --------- Negatives -----------
```



3.3.13 Note about implementation

This is a note on two problems I have seen in the past and how they can be easily fixed.

1. Summation along different axes ?
2. Summation of gradients over samples ?

1. Summation to create probability vectors in the Softmax function

Suppose X is a d x N array, in our case, it is 784 x 10000.

`Z2 = W2 Y1 + b2  will be 10 x 10000 array`

`softmax(Z2)` will be a `10 x 10000` array in which we want to apply a softmax function on every column of `Z2` by first computing exponential and then normalizing the column to sum to 1, which is needed for it to be a probability vector.

We can do that as

```
probs = np.exp(Z2)

# now you want to sum up each column and divide the column by the sum so that each column is a

probs /= np.sum(probs,axis=0,keepdims=True) # this makes sum of each column to 1
```

The **WRONG** thing to do is

```
probs /= np.sum(probs)
# This is WRONG. np.sum() computes sum of the entire array.
```

2. Computing gradient for the entire loss function

(this involves summation of N rank-one matrices in our notation.)

Suppose you have computed delta1, delta2 properly

Let's assume you computed

```
# delta2 is a 10 x 10000 array
# Y1 is a 256 x 10000 array
# N is 10000
# grad_W2 should be a 10 x 256 array
```

We can expand the formula for the gradient of the overall loss.

$$\nabla_{W^{(2)}} Loss = \frac{1}{N} \sum_i \nabla_{W^{(2)}} Loss_i,$$

where

$$\nabla_{W^{(2)}} Loss_i = \delta^{(2)} y^{(1)T}$$

is the gradient of the loss for ith training sample, where $\delta^{(2)}$ is a column of length 10 and $y^{(1)T}$ is a row of length 256, corresponding to ith training sample. Matrix product of column and row gives a a rank-1 matrix of size 10 x 256.

To compute the gradient of loss over all the training samples, we need to average the rank-1 matrices for all N training samples.

We can write the code for that as

```
# Sum gradient of loss for each sample
for i in range(N):
    grad_W2 += (1/N)*delta2[:,i,None].dot(Y1[:,i,None].T)

# OR we can compute grad_W2 without for loop as
grad_W2 = 1/N*np.dot(delta2,Y1.T)
```

To see why this is true, you can convince yourself that matrix product of an `M x N` matrix with an `N x K` matrix can be written as a summation of N `M x K` rank-one matricess.

Suppose

$$A = [\mathbf{a}_1 \ \cdots \ \mathbf{a}_N] \text{ and } B \ = \begin{bmatrix} \mathbf{b}_1^T \\ \vdots \\ \mathbf{b}_N^T \end{bmatrix},$$

where $\mathbf{a}_i, \mathbf{b}_i$ are columns of length M, K, respectively.

We can write AB as

$$AB = \sum_{i=1}^{N} \mathbf{a}_i \mathbf{b}_i^T.$$

3.4 Submission instructions

1. Download this Colab to ipynb, and convert it to PDF. Follow similar steps as here but convert to PDF.

- Download your .ipynb file. You can do it using only Google Colab. `File -> Download -> Download .ipynb`
- Reupload it so Colab can see it. Click on the `Files` icon on the far left to expand the side bar. You can directly drag the downloaded .ipynb file to the area. Or click `Upload to session storage` icon and then select & upload your .ipynb file.
- Conversion using %%shell. `!sudo apt-get update !sudo apt-get install texlive-xetex texlive-fonts-recommended texlive-generic-recommended !jupyter nbconvert --log-level CRITICAL --to pdf name_of_hw.ipynb`
- Your PDF file is ready. Click 3 dots and `Download`.

2. Upload the PDF to Gradescope, select the correct pdf pages for each question. **Important!**

3. Upload the ipynb file to Gradescope

Notice: In case of errors in conversion, please check your LaTeX and debug. In Markdown, when you write in LaTeX math mode, do not leave any leading and trailing whitespaces inside the dollar signs ($). For example, write `(dollarSign)\mathbf(dollarSign)(dollarSign)` instead of `(dollarSign)(space)\mathbf{w}(dollarSign)`. Otherwise, nbconvert will throw an error and the generated pdf will be incomplete. This is a bug of nbconvert.

[40]:
```
!sudo apt-get update
!sudo apt-get install texlive-xetex texlive-fonts-recommended
 ↪texlive-generic-recommended
```

```
Get:1 https://cloud.r-project.org/bin/linux/ubuntu bionic-cran40/ InRelease
[3,626 B]
Ign:2 https://developer.download.nvidia.com/compute/machine-
learning/repos/ubuntu1804/x86_64  InRelease
```

```
Get:3 http://ppa.launchpad.net/c2d4u.team/c2d4u4.0+/ubuntu bionic InRelease
[15.9 kB]
Hit:4 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64
InRelease
Hit:5 https://developer.download.nvidia.com/compute/machine-
learning/repos/ubuntu1804/x86_64  Release
Get:6 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]
Hit:7 http://archive.ubuntu.com/ubuntu bionic InRelease
Get:8 http://archive.ubuntu.com/ubuntu bionic-updates InRelease [88.7 kB]
Hit:9 http://ppa.launchpad.net/cran/libgit2/ubuntu bionic InRelease
Hit:11 http://ppa.launchpad.net/deadsnakes/ppa/ubuntu bionic InRelease
Get:12 http://archive.ubuntu.com/ubuntu bionic-backports InRelease [83.3 kB]
Hit:13 http://ppa.launchpad.net/graphics-drivers/ppa/ubuntu bionic InRelease
Get:14 http://security.ubuntu.com/ubuntu bionic-security/main amd64 Packages
[3,040 kB]
Get:15 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 Packages
[3,472 kB]
Get:16 http://ppa.launchpad.net/c2d4u.team/c2d4u4.0+/ubuntu bionic/main Sources
[2,215 kB]
Get:17 http://security.ubuntu.com/ubuntu bionic-security/universe amd64 Packages
[1,554 kB]
Get:18 http://archive.ubuntu.com/ubuntu bionic-updates/universe amd64 Packages
[2,332 kB]
Get:19 http://ppa.launchpad.net/c2d4u.team/c2d4u4.0+/ubuntu bionic/main amd64
Packages [1,133 kB]
Fetched 14.0 MB in 3s (5,074 kB/s)
Reading package lists… Done
Reading package lists… Done
Building dependency tree
Reading state information… Done
The following package was automatically installed and is no longer required:
  libnvidia-common-460
Use 'sudo apt autoremove' to remove it.
The following additional packages will be installed:
  fonts-droid-fallback fonts-lato fonts-lmodern fonts-noto-mono fonts-texgyre
  javascript-common libcupsfilters1 libcupsimage2 libgs9 libgs9-common
  libijs-0.35 libjbig2dec0 libjs-jquery libkpathsea6 libpotrace0 libptexenc1
  libruby2.5 libsynctex1 libtexlua52 libtexluajit2 libzzip-0-13 lmodern
  poppler-data preview-latex-style rake ruby ruby-did-you-mean ruby-minitest
  ruby-net-telnet ruby-power-assert ruby-test-unit ruby2.5
  rubygems-integration t1utils tex-common tex-gyre texlive-base
  texlive-binaries texlive-latex-base texlive-latex-extra
  texlive-latex-recommended texlive-pictures texlive-plain-generic tipa
Suggested packages:
  fonts-noto apache2 | lighttpd | httpd poppler-utils ghostscript
  fonts-japanese-mincho | fonts-ipafont-mincho fonts-japanese-gothic
  | fonts-ipafont-gothic fonts-arphic-ukai fonts-arphic-uming fonts-nanum ri
  ruby-dev bundler debhelper gv | postscript-viewer perl-tk xpdf-reader
```

```
  | pdf-viewer texlive-fonts-recommended-doc texlive-latex-base-doc
  python-pygments icc-profiles libfile-which-perl
  libspreadsheet-parseexcel-perl texlive-latex-extra-doc
  texlive-latex-recommended-doc texlive-pstricks dot2tex prerex ruby-tcltk
  | libtcltk-ruby texlive-pictures-doc vprerex
The following NEW packages will be installed:
  fonts-droid-fallback fonts-lato fonts-lmodern fonts-noto-mono fonts-texgyre
  javascript-common libcupsfilters1 libcupsimage2 libgs9 libgs9-common
  libijs-0.35 libjbig2dec0 libjs-jquery libkpathsea6 libpotrace0 libptexenc1
  libruby2.5 libsynctex1 libtexlua52 libtexluajit2 libzzip-0-13 lmodern
  poppler-data preview-latex-style rake ruby ruby-did-you-mean ruby-minitest
  ruby-net-telnet ruby-power-assert ruby-test-unit ruby2.5
  rubygems-integration t1utils tex-common tex-gyre texlive-base
  texlive-binaries texlive-fonts-recommended texlive-generic-recommended
  texlive-latex-base texlive-latex-extra texlive-latex-recommended
  texlive-pictures texlive-plain-generic texlive-xetex tipa
0 upgraded, 47 newly installed, 0 to remove and 6 not upgraded.
Need to get 146 MB of archives.
After this operation, 460 MB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu bionic/main amd64 fonts-droid-fallback
all 1:6.0.1r16-1.1 [1,805 kB]
Get:2 http://archive.ubuntu.com/ubuntu bionic/main amd64 fonts-lato all 2.0-2
[2,698 kB]
Get:3 http://archive.ubuntu.com/ubuntu bionic/main amd64 poppler-data all
0.4.8-2 [1,479 kB]
Get:4 http://archive.ubuntu.com/ubuntu bionic/main amd64 tex-common all 6.09
[33.0 kB]
Get:5 http://archive.ubuntu.com/ubuntu bionic/main amd64 fonts-lmodern all
2.004.5-3 [4,551 kB]
Get:6 http://archive.ubuntu.com/ubuntu bionic/main amd64 fonts-noto-mono all
20171026-2 [75.5 kB]
Get:7 http://archive.ubuntu.com/ubuntu bionic/universe amd64 fonts-texgyre all
20160520-1 [8,761 kB]
Get:8 http://archive.ubuntu.com/ubuntu bionic/main amd64 javascript-common all
11 [6,066 B]
Get:9 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 libcupsfilters1
amd64 1.20.2-0ubuntu3.1 [108 kB]
Get:10 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 libcupsimage2
amd64 2.2.7-1ubuntu2.9 [18.6 kB]
Get:11 http://archive.ubuntu.com/ubuntu bionic/main amd64 libijs-0.35 amd64
0.35-13 [15.5 kB]
Get:12 http://archive.ubuntu.com/ubuntu bionic/main amd64 libjbig2dec0 amd64
0.13-6 [55.9 kB]
Get:13 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 libgs9-common
all 9.26~dfsg+0-0ubuntu0.18.04.17 [5,092 kB]
Get:14 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 libgs9 amd64
9.26~dfsg+0-0ubuntu0.18.04.17 [2,267 kB]
Get:15 http://archive.ubuntu.com/ubuntu bionic/main amd64 libjs-jquery all
```

```
3.2.1-1 [152 kB]
Get:16 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 libkpathsea6
amd64 2017.20170613.44572-8ubuntu0.1 [54.9 kB]
Get:17 http://archive.ubuntu.com/ubuntu bionic/main amd64 libpotrace0 amd64
1.14-2 [17.4 kB]
Get:18 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 libptexenc1
amd64 2017.20170613.44572-8ubuntu0.1 [34.5 kB]
Get:19 http://archive.ubuntu.com/ubuntu bionic/main amd64 rubygems-integration
all 1.11 [4,994 B]
Get:20 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 ruby2.5 amd64
2.5.1-1ubuntu1.12 [48.6 kB]
Get:21 http://archive.ubuntu.com/ubuntu bionic/main amd64 ruby amd64 1:2.5.1
[5,712 B]
Get:22 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 rake all
12.3.1-1ubuntu0.1 [44.9 kB]
Get:23 http://archive.ubuntu.com/ubuntu bionic/main amd64 ruby-did-you-mean all
1.2.0-2 [9,700 B]
Get:24 http://archive.ubuntu.com/ubuntu bionic/main amd64 ruby-minitest all
5.10.3-1 [38.6 kB]
Get:25 http://archive.ubuntu.com/ubuntu bionic/main amd64 ruby-net-telnet all
0.1.1-2 [12.6 kB]
Get:26 http://archive.ubuntu.com/ubuntu bionic/main amd64 ruby-power-assert all
0.3.0-1 [7,952 B]
Get:27 http://archive.ubuntu.com/ubuntu bionic/main amd64 ruby-test-unit all
3.2.5-1 [61.1 kB]
Get:28 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 libruby2.5
amd64 2.5.1-1ubuntu1.12 [3,073 kB]
Get:29 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 libsynctex1
amd64 2017.20170613.44572-8ubuntu0.1 [41.4 kB]
Get:30 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 libtexlua52
amd64 2017.20170613.44572-8ubuntu0.1 [91.2 kB]
Get:31 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 libtexluajit2
amd64 2017.20170613.44572-8ubuntu0.1 [230 kB]
Get:32 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 libzzip-0-13
amd64 0.13.62-3.1ubuntu0.18.04.1 [26.0 kB]
Get:33 http://archive.ubuntu.com/ubuntu bionic/main amd64 lmodern all 2.004.5-3
[9,631 kB]
Get:34 http://archive.ubuntu.com/ubuntu bionic/main amd64 preview-latex-style
all 11.91-1ubuntu1 [185 kB]
Get:35 http://archive.ubuntu.com/ubuntu bionic/main amd64 t1utils amd64 1.41-2
[56.0 kB]
Get:36 http://archive.ubuntu.com/ubuntu bionic/universe amd64 tex-gyre all
20160520-1 [4,998 kB]
Get:37 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 texlive-
binaries amd64 2017.20170613.44572-8ubuntu0.1 [8,179 kB]
Get:38 http://archive.ubuntu.com/ubuntu bionic/main amd64 texlive-base all
2017.20180305-1 [18.7 MB]
Get:39 http://archive.ubuntu.com/ubuntu bionic/universe amd64 texlive-fonts-
```

```
recommended all 2017.20180305-1 [5,262 kB]
Get:40 http://archive.ubuntu.com/ubuntu bionic/universe amd64 texlive-plain-
generic all 2017.20180305-2 [23.6 MB]
Get:41 http://archive.ubuntu.com/ubuntu bionic/universe amd64 texlive-generic-
recommended all 2017.20180305-1 [15.9 kB]
Get:42 http://archive.ubuntu.com/ubuntu bionic/main amd64 texlive-latex-base all
2017.20180305-1 [951 kB]
Get:43 http://archive.ubuntu.com/ubuntu bionic/main amd64 texlive-latex-
recommended all 2017.20180305-1 [14.9 MB]
Get:44 http://archive.ubuntu.com/ubuntu bionic/universe amd64 texlive-pictures
all 2017.20180305-1 [4,026 kB]
Get:45 http://archive.ubuntu.com/ubuntu bionic/universe amd64 texlive-latex-
extra all 2017.20180305-2 [10.6 MB]
Get:46 http://archive.ubuntu.com/ubuntu bionic/universe amd64 tipa all 2:1.3-20
[2,978 kB]
Get:47 http://archive.ubuntu.com/ubuntu bionic/universe amd64 texlive-xetex all
2017.20180305-1 [10.7 MB]
Fetched 146 MB in 5s (32.1 MB/s)
debconf: unable to initialize frontend: Dialog
debconf: (No usable dialog-like program is installed, so the dialog based
frontend cannot be used. at /usr/share/perl5/Debconf/FrontEnd/Dialog.pm line 76,
<> line 47.)
debconf: falling back to frontend: Readline
debconf: unable to initialize frontend: Readline
debconf: (This frontend requires a controlling tty.)
debconf: falling back to frontend: Teletype
dpkg-preconfigure: unable to re-open stdin:
Selecting previously unselected package fonts-droid-fallback.
(Reading database … 123942 files and directories currently installed.)
Preparing to unpack …/00-fonts-droid-fallback_1%3a6.0.1r16-1.1_all.deb …
Unpacking fonts-droid-fallback (1:6.0.1r16-1.1) …
Selecting previously unselected package fonts-lato.
Preparing to unpack …/01-fonts-lato_2.0-2_all.deb …
Unpacking fonts-lato (2.0-2) …
Selecting previously unselected package poppler-data.
Preparing to unpack …/02-poppler-data_0.4.8-2_all.deb …
Unpacking poppler-data (0.4.8-2) …
Selecting previously unselected package tex-common.
Preparing to unpack …/03-tex-common_6.09_all.deb …
Unpacking tex-common (6.09) …
Selecting previously unselected package fonts-lmodern.
Preparing to unpack …/04-fonts-lmodern_2.004.5-3_all.deb …
Unpacking fonts-lmodern (2.004.5-3) …
Selecting previously unselected package fonts-noto-mono.
Preparing to unpack …/05-fonts-noto-mono_20171026-2_all.deb …
Unpacking fonts-noto-mono (20171026-2) …
Selecting previously unselected package fonts-texgyre.
Preparing to unpack …/06-fonts-texgyre_20160520-1_all.deb …
```

```
Unpacking fonts-texgyre (20160520-1) …
Selecting previously unselected package javascript-common.
Preparing to unpack …/07-javascript-common_11_all.deb …
Unpacking javascript-common (11) …
Selecting previously unselected package libcupsfilters1:amd64.
Preparing to unpack …/08-libcupsfilters1_1.20.2-0ubuntu3.1_amd64.deb …
Unpacking libcupsfilters1:amd64 (1.20.2-0ubuntu3.1) …
Selecting previously unselected package libcupsimage2:amd64.
Preparing to unpack …/09-libcupsimage2_2.2.7-1ubuntu2.9_amd64.deb …
Unpacking libcupsimage2:amd64 (2.2.7-1ubuntu2.9) …
Selecting previously unselected package libijs-0.35:amd64.
Preparing to unpack …/10-libijs-0.35_0.35-13_amd64.deb …
Unpacking libijs-0.35:amd64 (0.35-13) …
Selecting previously unselected package libjbig2dec0:amd64.
Preparing to unpack …/11-libjbig2dec0_0.13-6_amd64.deb …
Unpacking libjbig2dec0:amd64 (0.13-6) …
Selecting previously unselected package libgs9-common.
Preparing to unpack …/12-libgs9-common_9.26~dfsg+0-0ubuntu0.18.04.17_all.deb
…
Unpacking libgs9-common (9.26~dfsg+0-0ubuntu0.18.04.17) …
Selecting previously unselected package libgs9:amd64.
Preparing to unpack …/13-libgs9_9.26~dfsg+0-0ubuntu0.18.04.17_amd64.deb …
Unpacking libgs9:amd64 (9.26~dfsg+0-0ubuntu0.18.04.17) …
Selecting previously unselected package libjs-jquery.
Preparing to unpack …/14-libjs-jquery_3.2.1-1_all.deb …
Unpacking libjs-jquery (3.2.1-1) …
Selecting previously unselected package libkpathsea6:amd64.
Preparing to unpack …/15-libkpathsea6_2017.20170613.44572-8ubuntu0.1_amd64.deb
…
Unpacking libkpathsea6:amd64 (2017.20170613.44572-8ubuntu0.1) …
Selecting previously unselected package libpotrace0.
Preparing to unpack …/16-libpotrace0_1.14-2_amd64.deb …
Unpacking libpotrace0 (1.14-2) …
Selecting previously unselected package libptexenc1:amd64.
Preparing to unpack …/17-libptexenc1_2017.20170613.44572-8ubuntu0.1_amd64.deb
…
Unpacking libptexenc1:amd64 (2017.20170613.44572-8ubuntu0.1) …
Selecting previously unselected package rubygems-integration.
Preparing to unpack …/18-rubygems-integration_1.11_all.deb …
Unpacking rubygems-integration (1.11) …
Selecting previously unselected package ruby2.5.
Preparing to unpack …/19-ruby2.5_2.5.1-1ubuntu1.12_amd64.deb …
Unpacking ruby2.5 (2.5.1-1ubuntu1.12) …
Selecting previously unselected package ruby.
Preparing to unpack …/20-ruby_1%3a2.5.1_amd64.deb …
Unpacking ruby (1:2.5.1) …
Selecting previously unselected package rake.
Preparing to unpack …/21-rake_12.3.1-1ubuntu0.1_all.deb …
```

```
Unpacking rake (12.3.1-1ubuntu0.1) …
Selecting previously unselected package ruby-did-you-mean.
Preparing to unpack …/22-ruby-did-you-mean_1.2.0-2_all.deb …
Unpacking ruby-did-you-mean (1.2.0-2) …
Selecting previously unselected package ruby-minitest.
Preparing to unpack …/23-ruby-minitest_5.10.3-1_all.deb …
Unpacking ruby-minitest (5.10.3-1) …
Selecting previously unselected package ruby-net-telnet.
Preparing to unpack …/24-ruby-net-telnet_0.1.1-2_all.deb …
Unpacking ruby-net-telnet (0.1.1-2) …
Selecting previously unselected package ruby-power-assert.
Preparing to unpack …/25-ruby-power-assert_0.3.0-1_all.deb …
Unpacking ruby-power-assert (0.3.0-1) …
Selecting previously unselected package ruby-test-unit.
Preparing to unpack …/26-ruby-test-unit_3.2.5-1_all.deb …
Unpacking ruby-test-unit (3.2.5-1) …
Selecting previously unselected package libruby2.5:amd64.
Preparing to unpack …/27-libruby2.5_2.5.1-1ubuntu1.12_amd64.deb …
Unpacking libruby2.5:amd64 (2.5.1-1ubuntu1.12) …
Selecting previously unselected package libsynctex1:amd64.
Preparing to unpack …/28-libsynctex1_2017.20170613.44572-8ubuntu0.1_amd64.deb
…
Unpacking libsynctex1:amd64 (2017.20170613.44572-8ubuntu0.1) …
Selecting previously unselected package libtexlua52:amd64.
Preparing to unpack …/29-libtexlua52_2017.20170613.44572-8ubuntu0.1_amd64.deb
…
Unpacking libtexlua52:amd64 (2017.20170613.44572-8ubuntu0.1) …
Selecting previously unselected package libtexluajit2:amd64.
Preparing to unpack
…/30-libtexluajit2_2017.20170613.44572-8ubuntu0.1_amd64.deb …
Unpacking libtexluajit2:amd64 (2017.20170613.44572-8ubuntu0.1) …
Selecting previously unselected package libzzip-0-13:amd64.
Preparing to unpack …/31-libzzip-0-13_0.13.62-3.1ubuntu0.18.04.1_amd64.deb …
Unpacking libzzip-0-13:amd64 (0.13.62-3.1ubuntu0.18.04.1) …
Selecting previously unselected package lmodern.
Preparing to unpack …/32-lmodern_2.004.5-3_all.deb …
Unpacking lmodern (2.004.5-3) …
Selecting previously unselected package preview-latex-style.
Preparing to unpack …/33-preview-latex-style_11.91-1ubuntu1_all.deb …
Unpacking preview-latex-style (11.91-1ubuntu1) …
Selecting previously unselected package t1utils.
Preparing to unpack …/34-t1utils_1.41-2_amd64.deb …
Unpacking t1utils (1.41-2) …
Selecting previously unselected package tex-gyre.
Preparing to unpack …/35-tex-gyre_20160520-1_all.deb …
Unpacking tex-gyre (20160520-1) …
Selecting previously unselected package texlive-binaries.
Preparing to unpack …/36-texlive-
```

```
binaries_2017.20170613.44572-8ubuntu0.1_amd64.deb …
Unpacking texlive-binaries (2017.20170613.44572-8ubuntu0.1) …
Selecting previously unselected package texlive-base.
Preparing to unpack …/37-texlive-base_2017.20180305-1_all.deb …
Unpacking texlive-base (2017.20180305-1) …
Selecting previously unselected package texlive-fonts-recommended.
Preparing to unpack …/38-texlive-fonts-recommended_2017.20180305-1_all.deb …
Unpacking texlive-fonts-recommended (2017.20180305-1) …
Selecting previously unselected package texlive-plain-generic.
Preparing to unpack …/39-texlive-plain-generic_2017.20180305-2_all.deb …
Unpacking texlive-plain-generic (2017.20180305-2) …
Selecting previously unselected package texlive-generic-recommended.
Preparing to unpack …/40-texlive-generic-recommended_2017.20180305-1_all.deb
…
Unpacking texlive-generic-recommended (2017.20180305-1) …
Selecting previously unselected package texlive-latex-base.
Preparing to unpack …/41-texlive-latex-base_2017.20180305-1_all.deb …
Unpacking texlive-latex-base (2017.20180305-1) …
Selecting previously unselected package texlive-latex-recommended.
Preparing to unpack …/42-texlive-latex-recommended_2017.20180305-1_all.deb …
Unpacking texlive-latex-recommended (2017.20180305-1) …
Selecting previously unselected package texlive-pictures.
Preparing to unpack …/43-texlive-pictures_2017.20180305-1_all.deb …
Unpacking texlive-pictures (2017.20180305-1) …
Selecting previously unselected package texlive-latex-extra.
Preparing to unpack …/44-texlive-latex-extra_2017.20180305-2_all.deb …
Unpacking texlive-latex-extra (2017.20180305-2) …
Selecting previously unselected package tipa.
Preparing to unpack …/45-tipa_2%3a1.3-20_all.deb …
Unpacking tipa (2:1.3-20) …
Selecting previously unselected package texlive-xetex.
Preparing to unpack …/46-texlive-xetex_2017.20180305-1_all.deb …
Unpacking texlive-xetex (2017.20180305-1) …
Setting up libgs9-common (9.26~dfsg+0-0ubuntu0.18.04.17) …
Setting up libkpathsea6:amd64 (2017.20170613.44572-8ubuntu0.1) …
Setting up libjs-jquery (3.2.1-1) …
Setting up libtexlua52:amd64 (2017.20170613.44572-8ubuntu0.1) …
Setting up fonts-droid-fallback (1:6.0.1r16-1.1) …
Setting up libsynctex1:amd64 (2017.20170613.44572-8ubuntu0.1) …
Setting up libptexenc1:amd64 (2017.20170613.44572-8ubuntu0.1) …
Setting up tex-common (6.09) …
debconf: unable to initialize frontend: Dialog
debconf: (No usable dialog-like program is installed, so the dialog based
frontend cannot be used. at /usr/share/perl5/Debconf/FrontEnd/Dialog.pm line
76.)
debconf: falling back to frontend: Readline
update-language: texlive-base not installed and configured, doing nothing!
Setting up poppler-data (0.4.8-2) …
```

```
Setting up tex-gyre (20160520-1) …
Setting up preview-latex-style (11.91-1ubuntu1) …
Setting up fonts-texgyre (20160520-1) …
Setting up fonts-noto-mono (20171026-2) …
Setting up fonts-lato (2.0-2) …
Setting up libcupsfilters1:amd64 (1.20.2-0ubuntu3.1) …
Setting up libcupsimage2:amd64 (2.2.7-1ubuntu2.9) …
Setting up libjbig2dec0:amd64 (0.13-6) …
Setting up ruby-did-you-mean (1.2.0-2) …
Setting up t1utils (1.41-2) …
Setting up ruby-net-telnet (0.1.1-2) …
Setting up libijs-0.35:amd64 (0.35-13) …
Setting up rubygems-integration (1.11) …
Setting up libpotrace0 (1.14-2) …
Setting up javascript-common (11) …
Setting up ruby-minitest (5.10.3-1) …
Setting up libzzip-0-13:amd64 (0.13.62-3.1ubuntu0.18.04.1) …
Setting up libgs9:amd64 (9.26~dfsg+0-0ubuntu0.18.04.17) …
Setting up libtexluajit2:amd64 (2017.20170613.44572-8ubuntu0.1) …
Setting up fonts-lmodern (2.004.5-3) …
Setting up ruby-power-assert (0.3.0-1) …
Setting up texlive-binaries (2017.20170613.44572-8ubuntu0.1) …
update-alternatives: using /usr/bin/xdvi-xaw to provide /usr/bin/xdvi.bin
(xdvi.bin) in auto mode
update-alternatives: using /usr/bin/bibtex.original to provide /usr/bin/bibtex
(bibtex) in auto mode
Setting up texlive-base (2017.20180305-1) …
mktexlsr: Updating /var/lib/texmf/ls-R-TEXLIVEDIST…
mktexlsr: Updating /var/lib/texmf/ls-R-TEXMFMAIN…
mktexlsr: Updating /var/lib/texmf/ls-R…
mktexlsr: Done.
tl-paper: setting paper size for dvips to a4:
/var/lib/texmf/dvips/config/config-paper.ps
tl-paper: setting paper size for dvipdfmx to a4:
/var/lib/texmf/dvipdfmx/dvipdfmx-paper.cfg
tl-paper: setting paper size for xdvi to a4: /var/lib/texmf/xdvi/XDvi-paper
tl-paper: setting paper size for pdftex to a4:
/var/lib/texmf/tex/generic/config/pdftexconfig.tex
debconf: unable to initialize frontend: Dialog
debconf: (No usable dialog-like program is installed, so the dialog based
frontend cannot be used. at /usr/share/perl5/Debconf/FrontEnd/Dialog.pm line
76.)
debconf: falling back to frontend: Readline
Setting up texlive-fonts-recommended (2017.20180305-1) …
Setting up texlive-plain-generic (2017.20180305-2) …
Setting up texlive-generic-recommended (2017.20180305-1) …
Setting up texlive-latex-base (2017.20180305-1) …
Setting up lmodern (2.004.5-3) …
```

```
Setting up texlive-latex-recommended (2017.20180305-1) …
Setting up texlive-pictures (2017.20180305-1) …
Setting up tipa (2:1.3-20) …
Regenerating '/var/lib/texmf/fmtutil.cnf-DEBIAN'… done.
Regenerating '/var/lib/texmf/fmtutil.cnf-TEXLIVEDIST'… done.
update-fmtutil has updated the following file(s):
        /var/lib/texmf/fmtutil.cnf-DEBIAN
        /var/lib/texmf/fmtutil.cnf-TEXLIVEDIST
If you want to activate the changes in the above file(s),
you should run fmtutil-sys or fmtutil.
Setting up texlive-latex-extra (2017.20180305-2) …
Setting up texlive-xetex (2017.20180305-1) …
Setting up ruby2.5 (2.5.1-1ubuntu1.12) …
Setting up ruby (1:2.5.1) …
Setting up ruby-test-unit (3.2.5-1) …
Setting up rake (12.3.1-1ubuntu0.1) …
Setting up libruby2.5:amd64 (2.5.1-1ubuntu1.12) …
Processing triggers for mime-support (3.60ubuntu1) …
Processing triggers for libc-bin (2.27-3ubuntu1.6) …
Processing triggers for man-db (2.8.3-2ubuntu0.1) …
Processing triggers for fontconfig (2.12.6-0ubuntu2) …
Processing triggers for tex-common (6.09) …
debconf: unable to initialize frontend: Dialog
debconf: (No usable dialog-like program is installed, so the dialog based
frontend cannot be used. at /usr/share/perl5/Debconf/FrontEnd/Dialog.pm line
76.)
debconf: falling back to frontend: Readline
Running updmap-sys. This may take some time… done.
Running mktexlsr /var/lib/texmf … done.
Building format(s) --all.
        This may take some time… done.
```

[41]:
```
!jupyter nbconvert --log-level CRITICAL --to pdf fall2022_hw3.ipynb # make sure
↪the ipynb name is correct
```

```
This application is used to convert notebook files (*.ipynb)
        to various other formats.

        WARNING: THE COMMANDLINE INTERFACE MAY CHANGE IN FUTURE RELEASES.

Options
=======
The options below are convenience aliases to configurable class-options,
as listed in the "Equivalent to" description-line of the aliases.
To see all configurable class-options for some <cmd>, use:
    <cmd> --help-all

--debug
```

```
    set log level to logging.DEBUG (maximize logging output)
    Equivalent to: [--Application.log_level=10]
--show-config
    Show the application's configuration (human-readable format)
    Equivalent to: [--Application.show_config=True]
--show-config-json
    Show the application's configuration (json format)
    Equivalent to: [--Application.show_config_json=True]
--generate-config
    generate default config file
    Equivalent to: [--JupyterApp.generate_config=True]
-y
    Answer yes to any questions instead of prompting.
    Equivalent to: [--JupyterApp.answer_yes=True]
--execute
    Execute the notebook prior to export.
    Equivalent to: [--ExecutePreprocessor.enabled=True]
--allow-errors
    Continue notebook execution even if one of the cells throws an error and
include the error message in the cell output (the default behaviour is to abort
conversion). This flag is only relevant if '--execute' was specified, too.
    Equivalent to: [--ExecutePreprocessor.allow_errors=True]
--stdin
    read a single notebook file from stdin. Write the resulting notebook with
default basename 'notebook.*'
    Equivalent to: [--NbConvertApp.from_stdin=True]
--stdout
    Write notebook output to stdout instead of files.
    Equivalent to: [--NbConvertApp.writer_class=StdoutWriter]
--inplace
    Run nbconvert in place, overwriting the existing notebook (only
            relevant when converting to notebook format)
    Equivalent to: [--NbConvertApp.use_output_suffix=False
--NbConvertApp.export_format=notebook --FilesWriter.build_directory=]
--clear-output
    Clear output of current file and save in place,
            overwriting the existing notebook.
    Equivalent to: [--NbConvertApp.use_output_suffix=False
--NbConvertApp.export_format=notebook --FilesWriter.build_directory=
--ClearOutputPreprocessor.enabled=True]
--no-prompt
    Exclude input and output prompts from converted document.
    Equivalent to: [--TemplateExporter.exclude_input_prompt=True
--TemplateExporter.exclude_output_prompt=True]
--no-input
    Exclude input cells and output prompts from converted document.
            This mode is ideal for generating code-free reports.
    Equivalent to: [--TemplateExporter.exclude_output_prompt=True
```

```
--TemplateExporter.exclude_input=True]
--log-level=<Enum>
    Set the log level by value or name.
    Choices: any of [0, 10, 20, 30, 40, 50, 'DEBUG', 'INFO', 'WARN', 'ERROR',
'CRITICAL']
    Default: 30
    Equivalent to: [--Application.log_level]
--config=<Unicode>
    Full path of a config file.
    Default: ''
    Equivalent to: [--JupyterApp.config_file]
--to=<Unicode>
    The export format to be used, either one of the built-in formats
            ['asciidoc', 'custom', 'html', 'latex', 'markdown', 'notebook',
'pdf', 'python', 'rst', 'script', 'slides']
            or a dotted object name that represents the import path for an
            `Exporter` class
    Default: 'html'
    Equivalent to: [--NbConvertApp.export_format]
--template=<Unicode>
    Name of the template file to use
    Default: ''
    Equivalent to: [--TemplateExporter.template_file]
--writer=<DottedObjectName>
    Writer class used to write the
                                    results of the conversion
    Default: 'FilesWriter'
    Equivalent to: [--NbConvertApp.writer_class]
--post=<DottedOrNone>
    PostProcessor class used to write the
                                    results of the conversion
    Default: ''
    Equivalent to: [--NbConvertApp.postprocessor_class]
--output=<Unicode>
    overwrite base name use for output files.
                can only be used when converting one notebook at a time.
    Default: ''
    Equivalent to: [--NbConvertApp.output_base]
--output-dir=<Unicode>
    Directory to write output(s) to. Defaults
                                  to output to the directory of each notebook.
To recover
                                  previous default behaviour (outputting to the
current
                                  working directory) use . as the flag value.
    Default: ''
    Equivalent to: [--FilesWriter.build_directory]
--reveal-prefix=<Unicode>
```

```
    The URL prefix for reveal.js (version 3.x).
            This defaults to the reveal CDN, but can be any url pointing to a
copy
            of reveal.js.
            For speaker notes to work, this must be a relative path to a local
            copy of reveal.js: e.g., "reveal.js".
            If a relative path is given, it must be a subdirectory of the
            current directory (from which the server is run).
            See the usage documentation
            (https://nbconvert.readthedocs.io/en/latest/usage.html#reveal-js-
html-slideshow)
            for more details.
    Default: ''
    Equivalent to: [--SlidesExporter.reveal_url_prefix]
--nbformat=<Enum>
    The nbformat version to write.
            Use this to downgrade notebooks.
    Choices: any of [1, 2, 3, 4]
    Default: 4
    Equivalent to: [--NotebookExporter.nbformat_version]

Examples
--------

    The simplest way to use nbconvert is

            > jupyter nbconvert mynotebook.ipynb

            which will convert mynotebook.ipynb to the default format (probably
HTML).

            You can specify the export format with `--to`.
            Options include ['asciidoc', 'custom', 'html', 'latex', 'markdown',
'notebook', 'pdf', 'python', 'rst', 'script', 'slides'].

            > jupyter nbconvert --to latex mynotebook.ipynb

            Both HTML and LaTeX support multiple output templates. LaTeX
includes
            'base', 'article' and 'report'.  HTML includes 'basic' and 'full'.
You
            can specify the flavor of the format used.

            > jupyter nbconvert --to html --template basic mynotebook.ipynb

            You can also pipe the output to stdout, rather than a file

            > jupyter nbconvert mynotebook.ipynb --stdout
```

```
        PDF is generated via latex

        > jupyter nbconvert mynotebook.ipynb --to pdf

        You can get (and serve) a Reveal.js-powered slideshow

        > jupyter nbconvert myslides.ipynb --to slides --post serve

        Multiple notebooks can be given at the command line in a couple of
        different ways:

        > jupyter nbconvert notebook*.ipynb
        > jupyter nbconvert notebook1.ipynb notebook2.ipynb

        or you can specify the notebooks list in a config file, containing::

            c.NbConvertApp.notebooks = ["my_notebook.ipynb"]

        > jupyter nbconvert --config mycfg.py

To see all available configurables, use `--help-all`.
```

[41]:

```

```