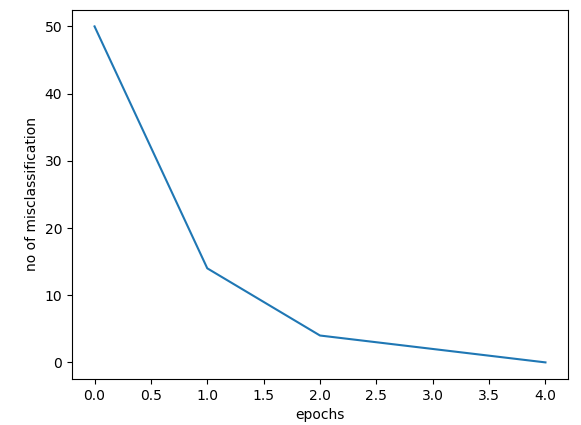
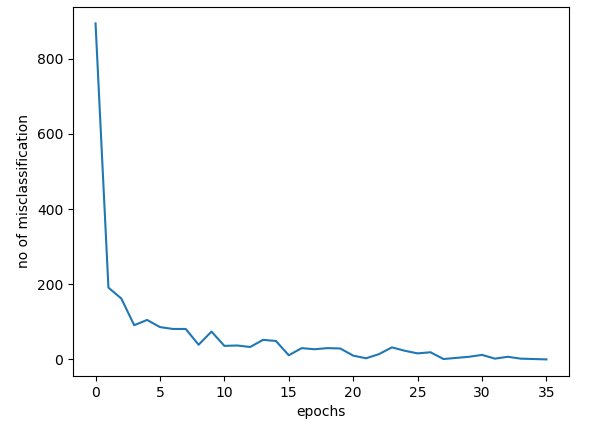
q2) Following are the answers for the report

f) 

Testing error (on all 10000 test images): 4603

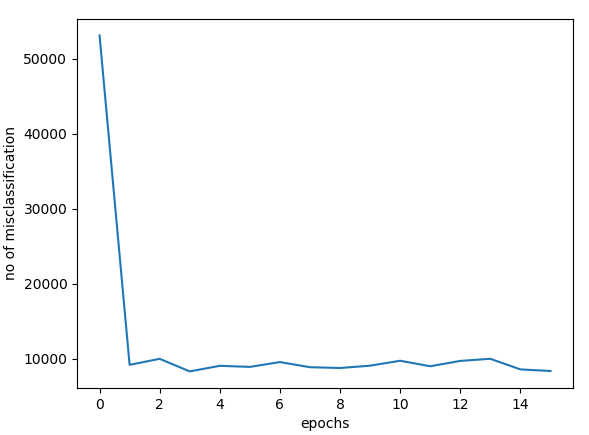
testing error: 0.46 (46%)

So the testing error is higher even when training error is 0 because we have trained only on limited sample(50) so the model can’t accurately predict the testing sample

g) 

testing error = 1794/10000 = 0.1794

So as compared to the previous part it takes more epochs (35 as compared to 3 before) to converge as we have increased the no of training samples in this case. The test error has also dropped as compared to before as more training samples were used to train the neural network, so it could learn the weights more accurately.

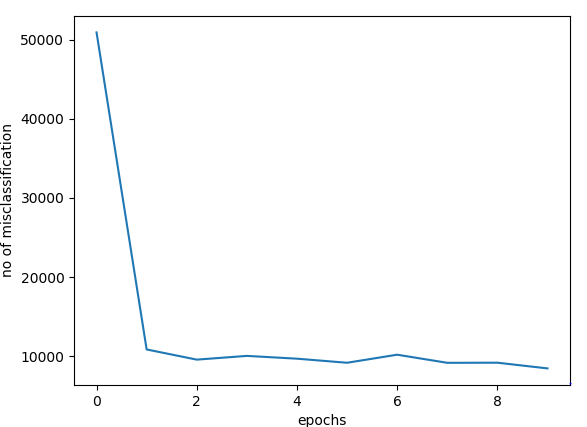
h) 

testing error = 0.1496

So as compared to before the algorithm doesn’t seem to converge, so had to stop after 15 epochs. As the data is not clearly linear separable so the algorithm is not able to converge so it comes close to 0.14 (approx) but it doesn’t seem to converge to zero. The testing error seems to come down as compared to previous part, as we have used all of the training samples.

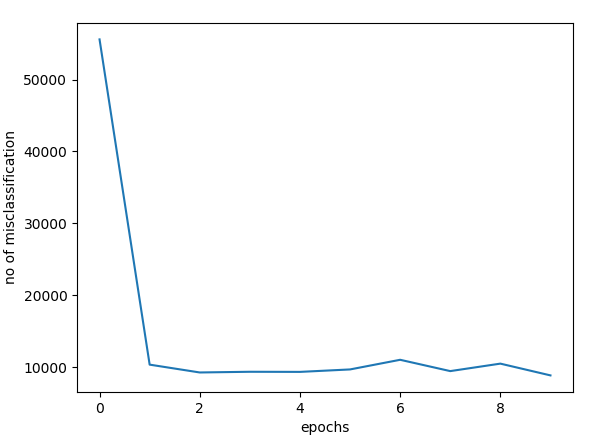
i) Choosing e = 0.15 as it seem to converge along that point

run 1 (n=60000,rate=0.5,e=0.15)



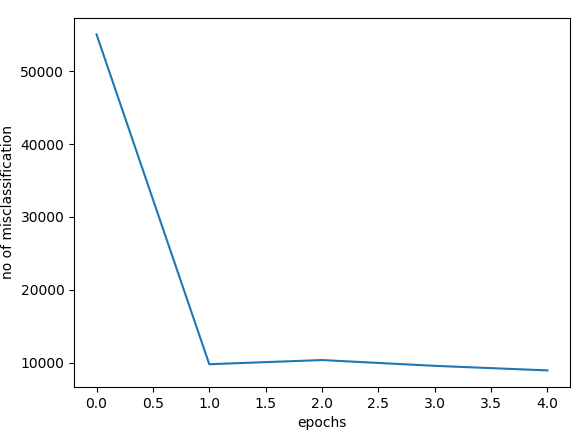
testing error = 0.1677

run 2 (n=60000,rate=0.5,e=0.15)



testing error = 0.1567

run 3 (n=60000,rate=0.5,e=0.15)



testing error = 0.1785

Even with the same value of learning rate and no of samples, the above runs had different convergence. Different set of initialisations of weights lead to different convergence. Some converge faster, and some takes more epochs to converge. The input sample are not clearly linearly separable so the PTA algo is doing its best to get to the minimum error but sometimes after changing weights it might lead to more error and might take longer to converge. Testing error of almost similar but not identical as the three started with different set of initial weights hence they will have different set of convergence and different W vector, as there is not one correct set of weights but rather its dependent of the initial weight initialisation.

Source Code:

**import** numpy **as** np  
**import** random  
**import** matplotlib.pyplot **as** plt  
**import** gzip  
**import** struct  
  
**class** Mnist:  
 **def** \_\_init\_\_(self):  
 self.files = {**'imagefile'**:**'image files//train-images-idx3-ubyte.gz'**,**'imagelabel'**:**'image files//train-labels-idx1-ubyte.gz'** ,**'testimage'**:**'image files//t10k-images-idx3-ubyte.gz'**,**'testlabel'**:**'image files//t10k-labels-idx1-ubyte.gz'**}  
 self.trainingImgs,self.trainNoImages,self.rows,self.columns = self.getImages(self.files[**'imagefile'**])  
 self.trainingLabels = self.getLabels(self.files[**'imagelabel'**])  
 self.testImgs,self.testNoImages,\_,\_ = self.getImages(self.files[**'testimage'**])  
 self.testLabels = self.getLabels(self.files[**'testlabel'**])  
 self.W = self.getRandomWeights(-1,1) *#initial weight vector* **def** getImages(self,filename):  
 f = gzip.open(**'../'**+filename)  
 **try**:  
 f.seek(4)  
 images = struct.unpack(**'>I'**, f.read(4))[0]  
 rows = struct.unpack(**'>I'**, f.read(4))[0]  
 columns = struct.unpack(**'>I'**, f.read(4))[0]  
 start = f.seek(16)  
 size = rows \* columns  
 pixel\_im = []  
 **for** i **in** range(images):  
 f.seek(start + (i \* size))  
 pixel = np.array(struct.unpack(**'>'** + **'B'** \* size, f.read(size)))  
 pixel\_im.append(pixel)  
 **return** (np.array(pixel\_im), images, rows, columns)  
 **except** struct.error **as** e:  
 print(e)  
  
 **def** getLabels(self,filename):  
 f = gzip.open(**'../'** + filename)  
 **try**:  
 f.seek(4)  
 images = struct.unpack(**'>I'**, f.read(4))[0]  
 f.seek(8)  
 labels = np.array(struct.unpack(**'>'** + **'B'** \* images, f.read(images)))  
 **return** (labels)  
 **except** struct.error **as** e:  
 print(e)  
  
 **def** getRandomWeights(self,a,b):  
 size = self.rows \* self.columns  
 W = np.empty((0, size), int)  
 **for** j **in** range(10):  
 w = np.array([random.uniform(a, b) **for** i **in** range(size)])  
 W = np.vstack([w, W]) *# weight vector Ω* **return** W  
  
 **def** getLabel(self,idx,label\_type= **'train'**):  
 **if** (label\_type == **'train'**):  
 numImages = self.trainNoImages  
 labels = self.trainingLabels  
 **elif** (label\_type == **'test'**):  
 numImages = self.testNoImages  
 labels = self.testLabels  
 **if** (idx <= numImages):  
 **return** (labels[idx])  
  
 **def** getDesiredInput(self, idx,label\_type = **'train'**):  
 d = np.zeros(10)  
 label = self.getLabel(idx,label\_type)  
 d[label] = 1  
 **return** (d)  
  
 **def** PTA\_mnist(self,n,rate, e):  
 epoch = 0  
 epoch\_err = []  
 W = self.W  
 pixel\_im = self.trainingImgs  
 labels = self.trainingLabels  
 **while True**:  
 mis = 0  
 **for** idx **in** range(n):  
 v = W @ pixel\_im[idx]  
 **if** (np.argmax(v) != labels[idx]):  
 mis += 1  
 epoch\_err.append(mis)  
 epoch = epoch + 1  
 **for** idx **in** range(n):  
 W = W + rate\*(self.getDesiredInput(idx,label\_type=**'train'**) - self.stepFunction(W @ pixel\_im[idx])).reshape(-1, 1) @ (  
 pixel\_im[idx].reshape(-1, 1).T)  
 print(epoch\_err[epoch - 1] / n)  
 **if** ((epoch\_err[epoch - 1] / n <= e ) **or** (epoch>15)):  
 **break  
 return** ( W ,epoch\_err,epoch)  
  
 *# testing with the updates weights* **def** testing(self, W, n):  
 error = 0  
 testLabels = self.testLabels  
 testImages =self.testImgs  
 **for** idx **in** range(n):  
 v = W @ testImages[idx]  
 **if** (np.argmax(v) != testLabels[idx]):  
 error += 1  
 **return** error  
  
 **def** stepFunc(self,x):  
 **if** (x >= 0):  
 **return** 1  
 **else**:  
 **return** 0  
  
 **def** signFunc(self,x):  
 **if** (x < 0):  
 **return** -1  
 **elif** (x == 0):  
 **return** 0  
 **elif** (x > 0 ):  
 **return** 1  
  
 *# @param : aray,list or int* **def** stepFunction(self,X):  
 **if** (type(X) == np.ndarray **or** type(X) == list):  
 **for** idx, x **in** enumerate(X):  
 X[idx] = self.stepFunc(x)  
 **elif** (type(X) == int):  
 X = self.stepFunc(X)  
 **return** (X)  
  
 *# @param : aray,list or int* **def** signFunction(self,X):  
 **if** (type(X) == np.ndarray **or** type(X) == list):  
 **for** idx, x **in** enumerate(X):  
 X[idx] = self.signFunc(x)  
 **elif** (type(X) == int):  
 X = self.signFunc(X)  
 **return** X  
  
 **def** graphEpochList(self,epoch,misList):  
 plt.plot(np.array(range(epoch)),misList)  
 plt.xlabel(**'epochs'**)  
 plt.ylabel(**'no of misclassification'**)  
 plt.show()  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 ob = Mnist()  
 W\_upd,epoch\_erros,epoch = ob.PTA\_mnist(n=60000,rate=0.5,e=0.15)  
 ob.graphEpochList(epoch,epoch\_erros)  
 error = ob.testing(W\_upd, ob.testNoImages)  
 print(**"testing error = "**,error)