Ozyegin University

Fall 2020

CS 554

Homework #3

Submitted by: M. Furkan Oruc

Student ID: S025464

Submission Date: December 7, 2020

Report Outline

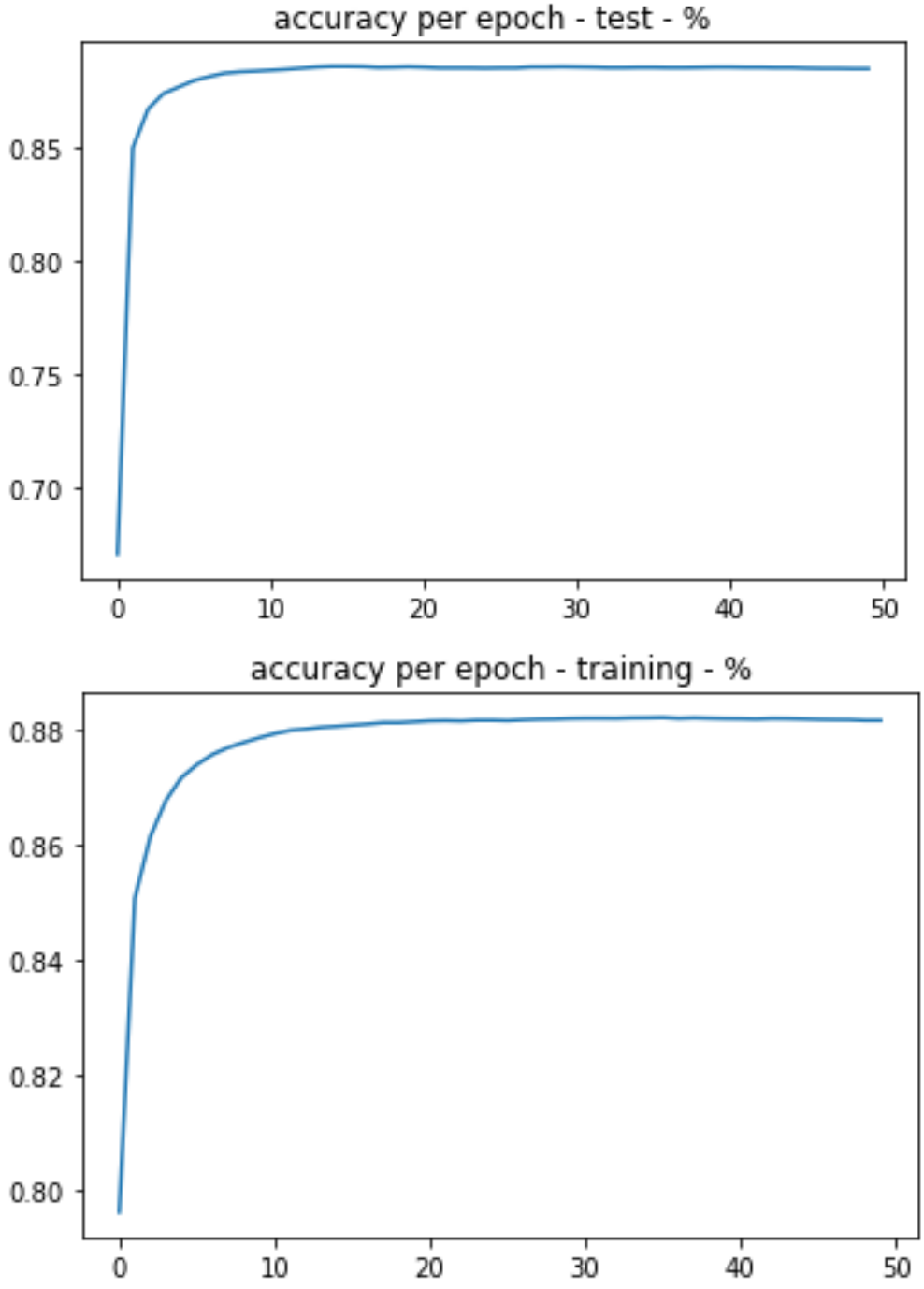
1. Single Layer Perceptron
    a. Accuracy Rates
    b. Confusion Matrices
    c. Weights' Visualization
2. Multi Layer Perceptron
    a. H=25 Case
        i. Accuracy Rates
        ii. Confusion Matrices
        iii. Weights' Visualization
    b. H=50 Case
        i. Accuracy Rates
        ii. Confusion Matrices
        iii. Weights' Visualization
    c. H=75 Case
        i. Accuracy Rates
        ii. Confusion Matrices
        iii. Weights' Visualization
3. Conclusion
4. Source Code

1. Single Layer Perceptron

Accuracy rates for both datasets are provided below.

Top Accuracy Rates (%)	
Training Set	0,88
Test Set	0,89

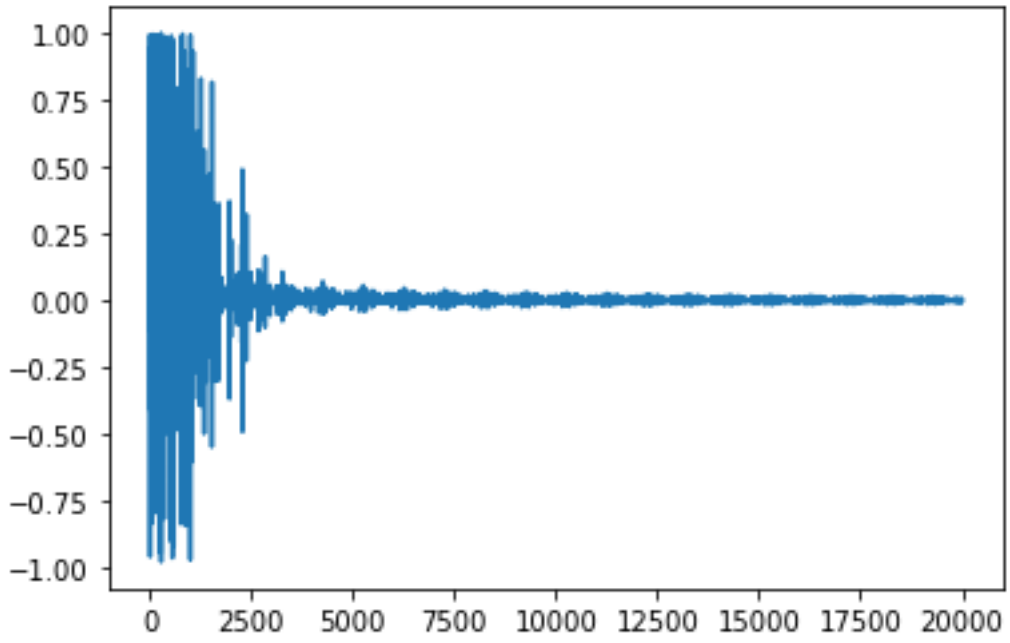
Change of accuracy rates for both datasets are provided below.



It can be clearly observed that as number of epochs increase, the increase in accuracy rate stagnates.

After 10 epoch, the accuracy rates changes slightly.

Error Plot for the dataset can be observed below. As trial number increases, error rate decreases.



Confusion matrix for training dataset can be observed below.
Rows represent correct label while columns represent assigned labels.

Training Dataset Confusion Matrix

Index	0	1	2	3	4	5	6	7	8	9
0	5616	0	32	17	10	70	79	4	82	13
1	0	6305	77	60	5	68	18	13	190	6
2	89	50	5116	132	101	21	152	99	161	37
3	59	20	165	5280	5	259	45	71	151	76
4	22	25	58	6	5245	10	107	20	71	278
5	142	53	65	322	89	4272	121	39	226	92
6	83	32	118	8	42	92	5473	4	66	0
7	85	84	123	21	57	11	5	5614	37	228
8	33	86	135	245	23	212	39	35	4911	132
9	64	35	53	110	215	43	5	246	101	5077

It can be expressed that the most mistaken instance is 5 with 3. On the other hand, classifier classifies number 1 with a very high accuracy.

Training dataset contains 60,000 instances, which means, if we assume an approximately equal distribution, each instance is contained around 6000. So, each row may sum up to a value around 6000.

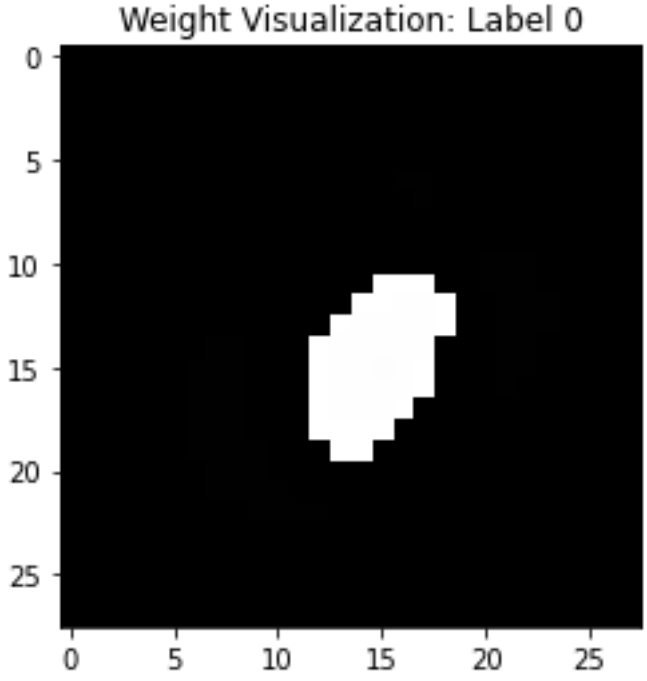
Test Dataset Confusion Matrix

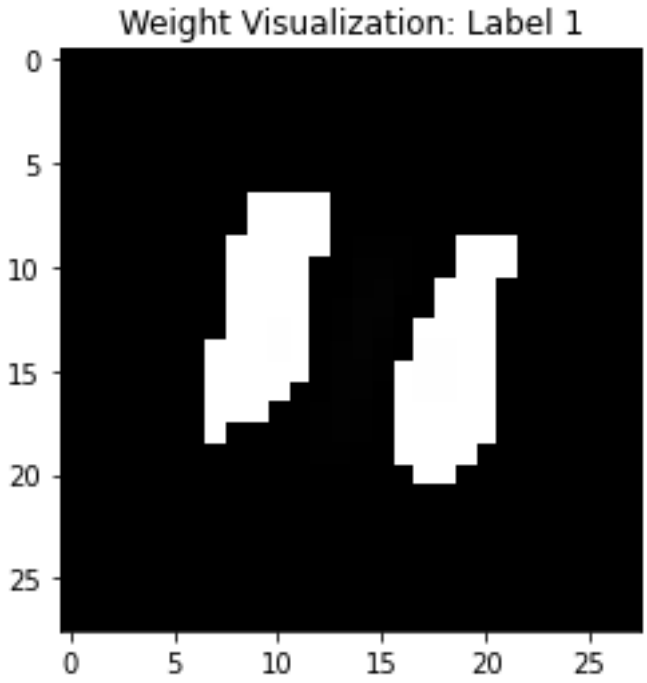
Index	0	1	2	3	4	5	6	7	8	9
0	949	0	2	3	0	9	11	1	5	0
1	0	1060	15	15	1	4	5	0	35	0
2	20	4	880	24	15	3	23	16	43	4
3	8	1	19	889	1	38	7	17	20	10
4	4	3	6	0	893	2	18	2	14	40
5	22	4	5	60	18	707	19	9	36	12
6	22	2	14	2	14	21	873	0	10	0
7	6	16	39	2	11	0	0	911	8	35
8	8	3	15	34	8	37	13	15	825	16
9	10	4	10	11	49	17	1	29	17	861

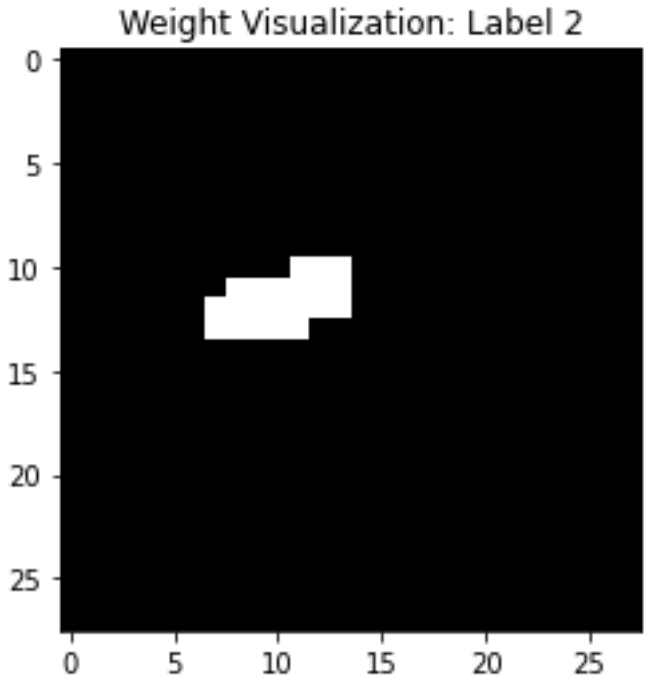
In test dataset, the most mistaken instance is 5 again, with 3.
Each row index represents correct label and each column index represents the assigned label.

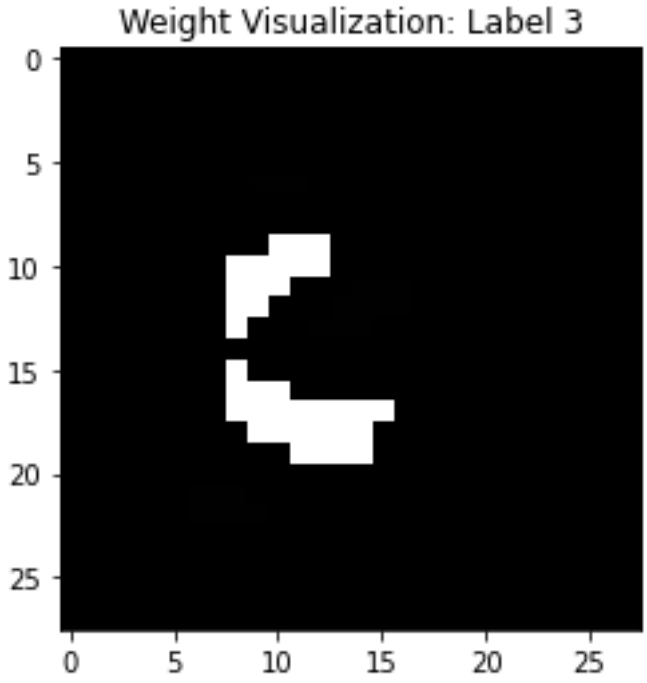
SLP – Weight Visualization

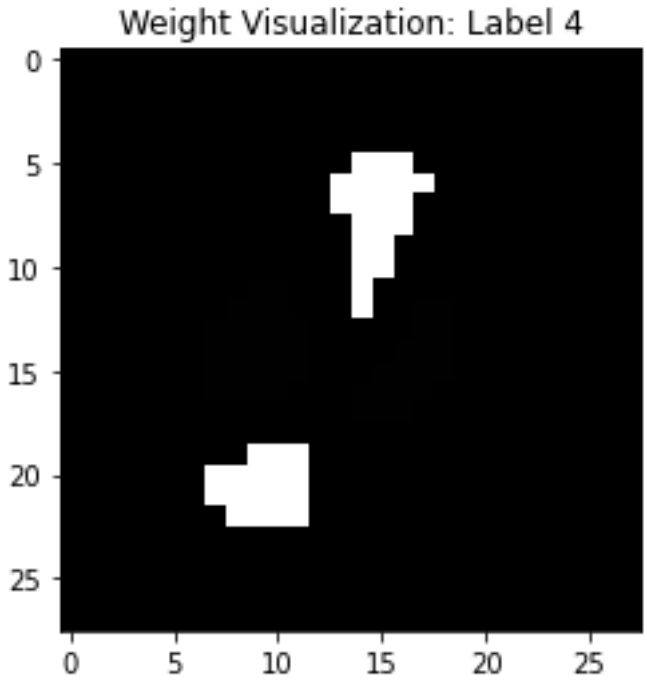
SLP did not provide clear visuals for weights to observe though some patterns may be easily discovered with a check.

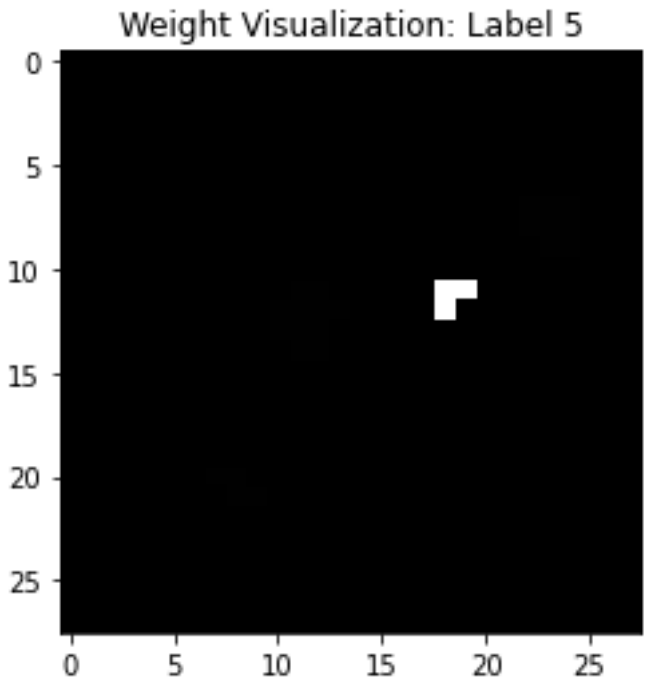


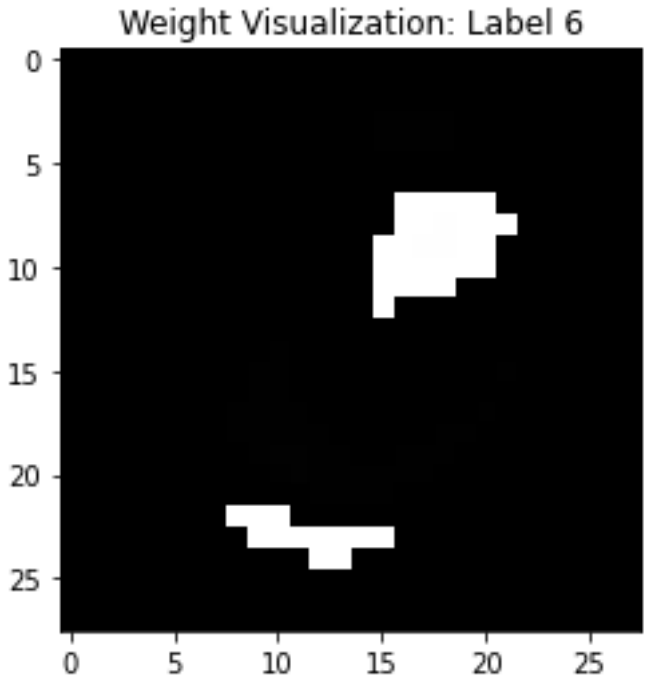


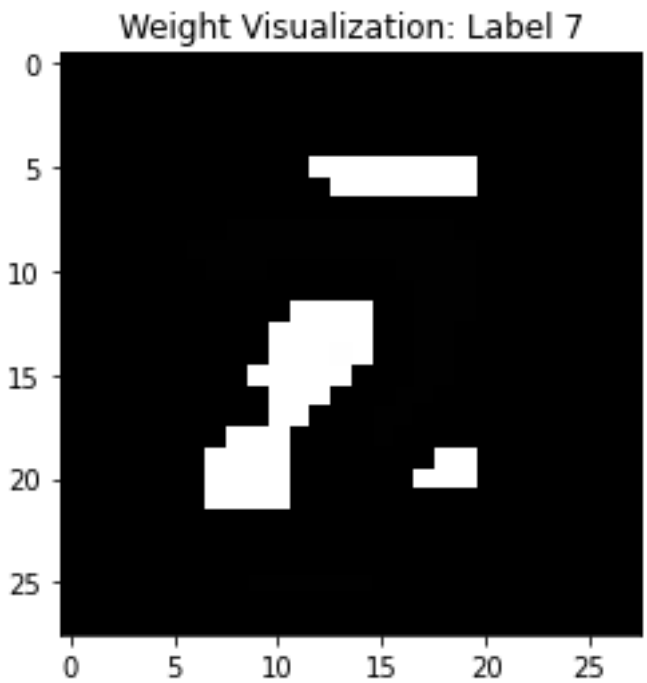

Weight Visualization: Label 2
0
5
10
15
20
25
0
5
10
15
20
25

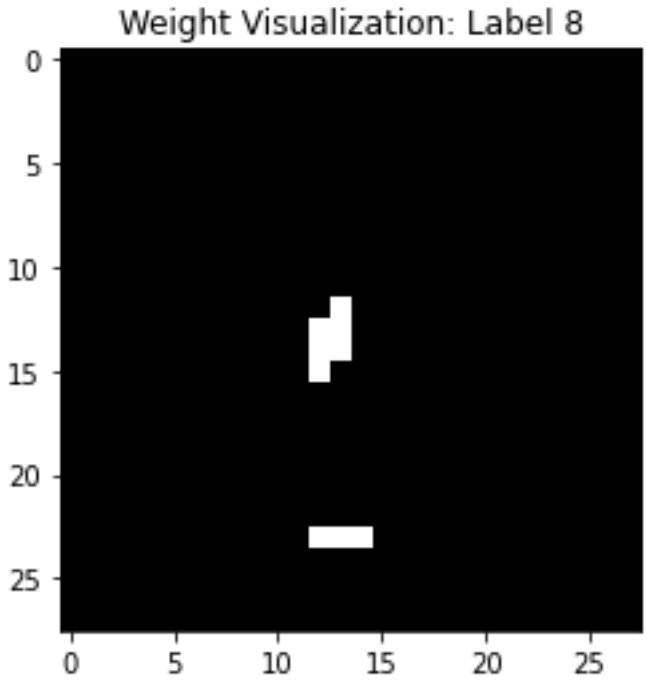

Weight Visualization: Label 3
0
5
10
15
20
25
0
5
10
15
20
25

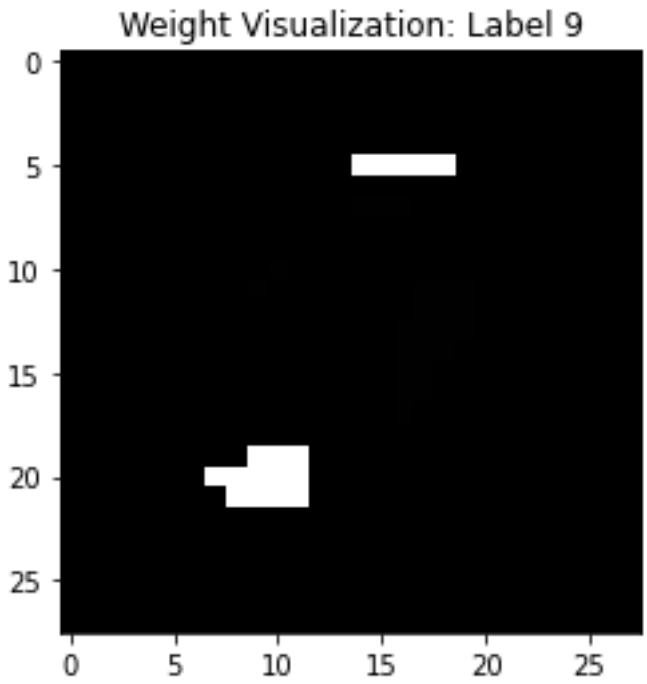

Weight Visualization: Label 4
0
5
10
15
20
25
0
5
10
15
20
25


Weight Visualization: Label 5
0
5
10
15
20
25
0
5
10
15
20
25


Weight Visualization: Label 6
0
5
10
15
20
25
0
5
10
15
20
25

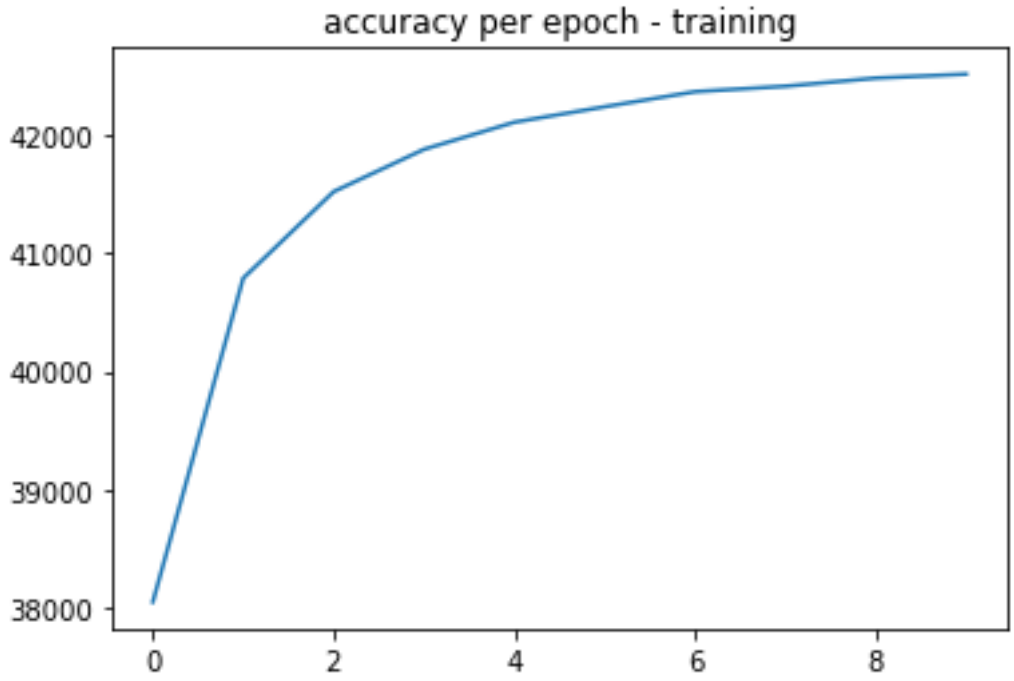

Weight Visualization: Label 7
0
5
10
15
20
25
0
5
10
15
20
25

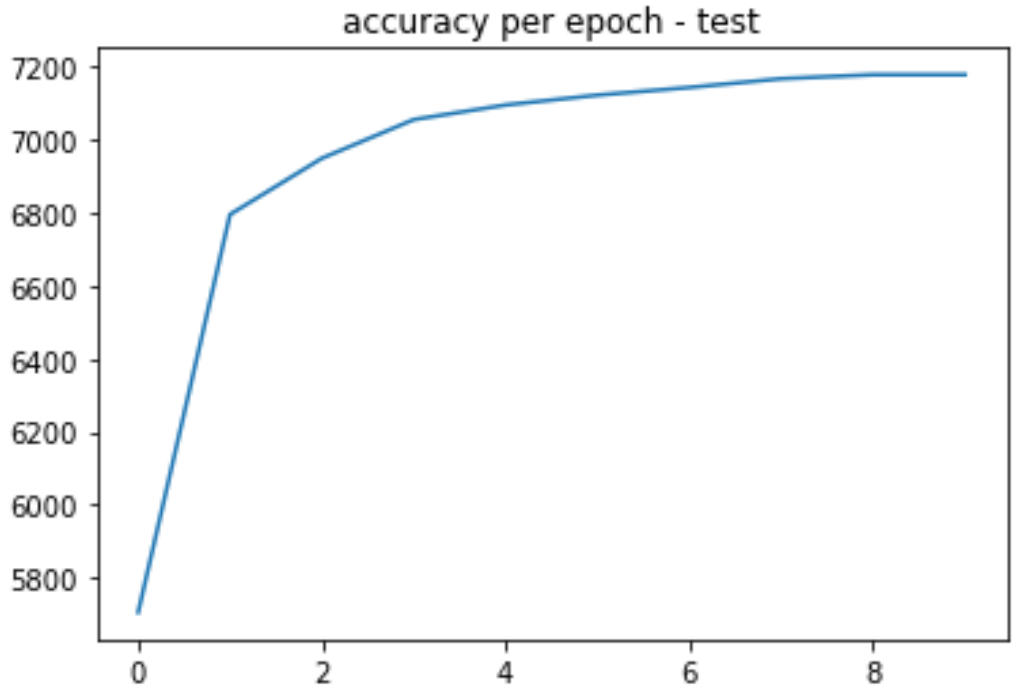

Weight Visualization: Label 8
0
5
10
15
20
25
0
5
10
15
20
25


Weight Visualization: Label 9
0
5
10
15
20
25
0
5
10
15
20
25

2. A. MLP – H=25 Case

Below are plots of accuracy change per epoch, based on 10 total epoch case. X axis represents number of epochs and y axis represents the accurate prediction made by model for each epoch.





For H=25 Units case, below are number of accurate predictions and percentages.

Top Accuracy (10th/10 Epoch)		
Training Set	42520	0,708666667
Test Set	7179	0,7179
	Total Accurate Prediction	Percentage of Accuracy

Training Set Accurate Prediction list for H=25 Case.

	0
1	40789
2	41523
3	41882
4	42112
5	42243
6	42371
7	42417
8	42486
9	42520

Test Set Accurate Prediction list for H=25 Case.

	0
1	6796
2	6950
3	7056
4	7096
5	7123
6	7144
7	7168
8	7179
9	7179

Training Set Confusion Matrix.

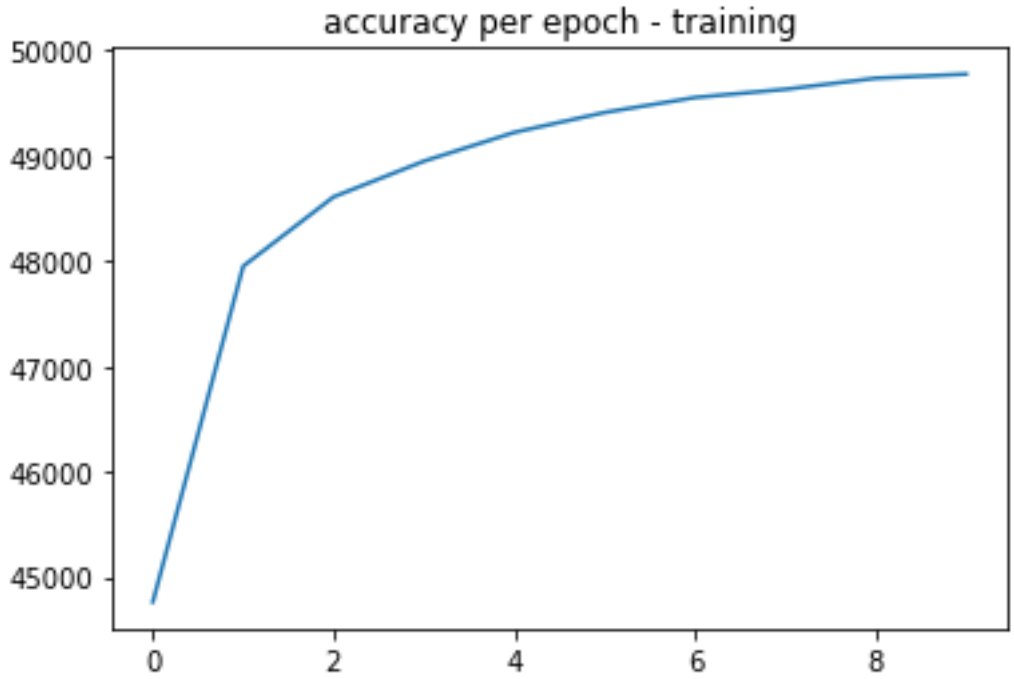
	0	1	2	3	4	5	6	7	8	9
0	4188	3	290	443	65	345	281	97	142	69
1	10	6213	89	16	14	83	8	20	248	41
2	378	163	3974	302	161	105	254	125	391	105
3	455	104	444	3949	55	349	112	197	354	112
4	139	103	102	32	4283	93	302	99	149	540
5	545	238	210	504	363	2438	311	141	476	195
6	250	52	173	87	213	262	4638	14	180	49
7	127	151	261	49	193	72	25	4807	112	468
8	147	250	298	392	68	360	136	82	3857	261
9	108	108	99	126	522	131	101	448	133	4173

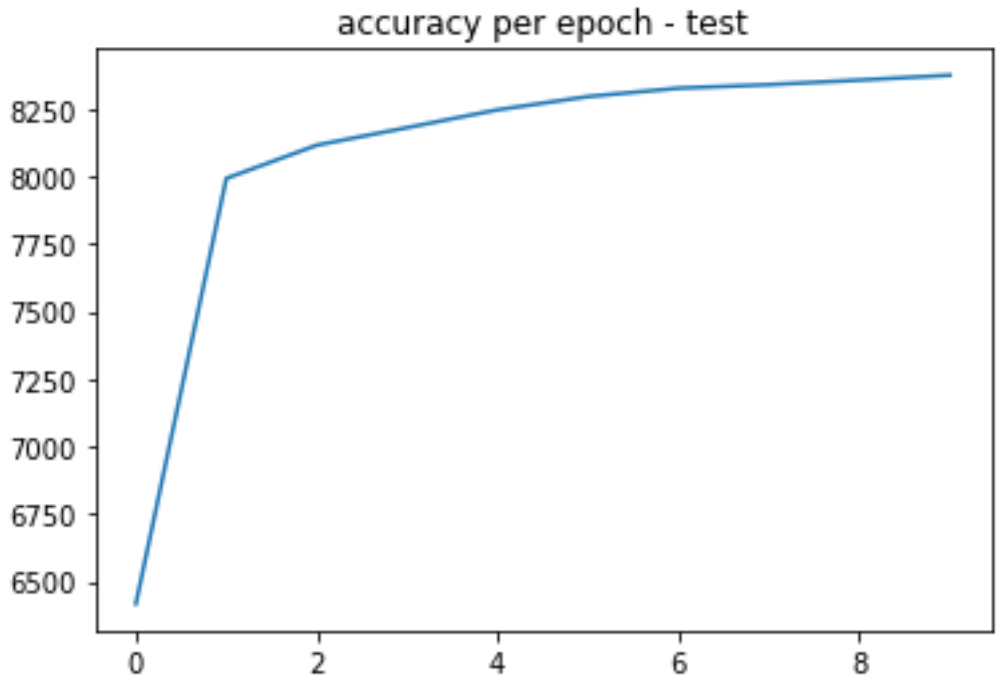
Test Set Confusion Matrix. 25H

	0	1	2	3	4	5	6	7	8	9
0	720	0	27	73	7	61	46	14	24	8
1	1	1064	11	4	3	13	2	4	31	2
2	69	29	696	57	25	19	28	19	72	18
3	82	10	45	677	10	64	7	43	56	16
4	22	18	8	6	748	7	43	14	28	88
5	101	33	19	96	44	391	48	28	98	34
6	51	10	35	12	42	37	745	6	15	5
7	14	19	57	7	31	9	4	788	22	77
8	33	31	41	53	16	70	19	21	638	52
9	24	13	15	14	90	25	20	68	28	712

2. B. MLP – H=50 Case

Below are plots of accuracy change per epoch, based on 10 total epoch case. X axis represents number of epochs and y axis represents the accurate prediction made by model for each epoch.





For H=25 Units case, below are number of accurate predictions and percentages.

Top Accuracy (10th/10 Epoch)		
Training Set	49773	0,82955
Test Set	8375	0,8375
	Total Accurate Prediction	Percentage of Accuracy

Training Set Accurate Prediction list for H=50 Case.

	0
0	44764
1	47950
2	48608
3	48947
4	49220
5	49409
6	49551
7	49629
8	49734
9	49773

Test Set Accurate Prediction list for H=50 Case.

	0
0	6418
1	7993
2	8116
3	8181
4	8247
5	8296
6	8327
7	8340
8	8357
9	8375

Training Set Confusion Matrix.

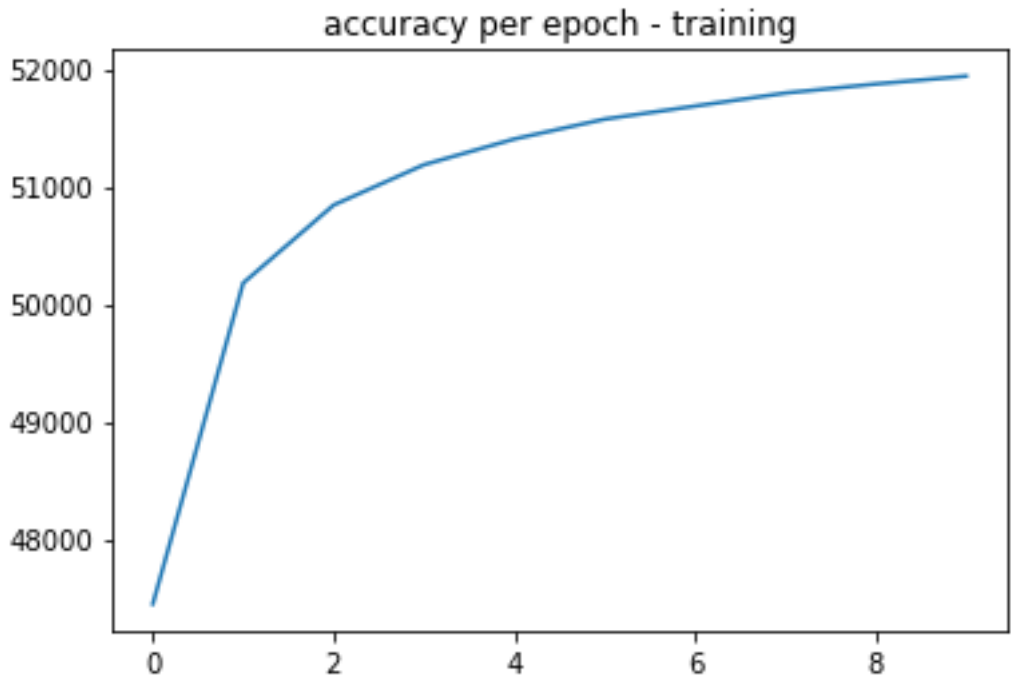
	0	1	2	3	4	5	6	7	8	9
0	5504	1	39	54	16	120	54	12	92	31
1	4	6414	55	23	17	42	17	21	132	17
2	89	152	4676	237	111	47	208	140	254	44
3	63	70	238	4886	25	366	69	98	181	135
4	34	40	93	45	4806	55	102	54	134	479
5	196	75	91	350	168	3867	212	92	291	79
6	57	68	88	25	75	163	5314	4	104	20
7	47	99	94	66	116	36	10	5438	61	298
8	53	243	196	267	105	249	87	86	4343	222
9	57	74	75	157	458	106	18	328	151	4525

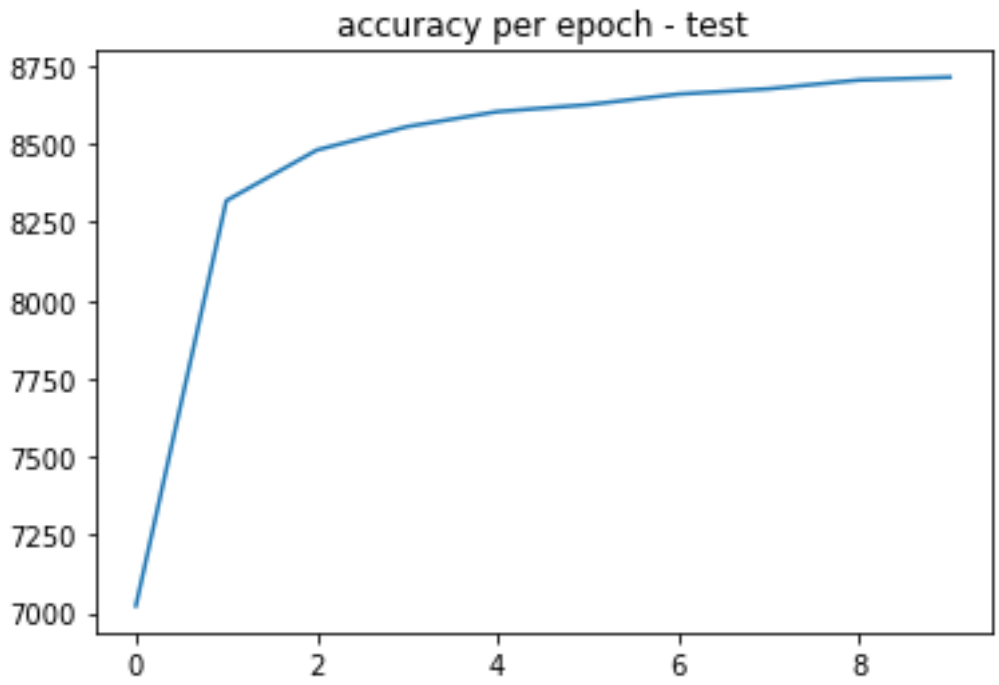
Test Set Confusion Matrix. H50

	0	1	2	3	4	5	6	7	8	9
0	917	0	8	11	2	17	9	3	7	6
1	0	1085	3	7	4	5	4	0	24	3
2	15	30	811	45	8	3	28	27	56	9
3	12	3	22	845	6	47	8	18	32	17
4	5	2	6	3	825	10	19	8	25	79
5	29	10	7	65	31	649	36	17	42	6
6	17	5	13	7	24	32	842	2	14	2
7	3	24	30	12	12	5	6	884	9	43
8	13	22	18	48	25	36	21	24	733	34
9	14	6	11	14	94	18	1	43	24	784

2. B MLP - H= 75 Case

Below are plots of accuracy change per epoch, based on 10 total epoch case. X axis represents number of epochs and y axis represents the accurate prediction made by model for each epoch.





For H=50 Units case, below are number of accurate predictions and percentages

Top Accuracy (10th/10 Epoch)		
Training Set	51939	0,86565
Test Set	8714	0,8714
	Total Accurate Prediction	Percentage of Accuracy

Training and Test Set Accurate Prediction list for H=75 Case can be followed below.

	0
0	47449
1	50179
2	50842
3	51185
4	51404
5	51573
6	51684
7	51795
8	51872
9	51939

Training Case

	0
0	7023
1	8319
2	8481
3	8556
4	8604
5	8626
6	8660
7	8677
8	8705
9	8714

Test Case

In the next two pages, confusion matrices are provided for H=75 Case.

Training Set Confusion Matrix. 75

	0	1	2	3	4	5	6	7	8	9
0	5599	0	33	15	22	64	85	17	69	19
1	1	6466	39	42	7	26	14	24	116	7
2	77	116	4858	173	154	28	172	88	234	58
3	37	39	208	5142	14	304	51	76	173	87
4	20	31	84	8	5124	18	90	46	83	338
5	101	76	55	360	114	4177	124	70	256	88
6	69	21	98	10	92	108	5437	12	54	17
7	51	86	126	44	130	15	8	5458	54	293
8	48	189	141	204	66	239	45	49	4724	146
9	43	42	53	70	303	59	15	311	99	4954

Test Set Confusion Matrix. 75=H

	0	1	2	3	4	5	6	7	8	9
0	936	0	5	4	3	7	12	2	10	1
1	0	1099	3	6	0	4	3	0	19	1
2	16	16	829	41	25	1	28	21	48	7
3	6	1	31	883	2	37	4	16	23	7
4	2	2	5	4	868	2	21	10	11	57
5	15	4	9	61	21	705	19	7	43	8
6	13	3	14	3	27	18	872	3	4	1
7	3	18	32	9	15	0	2	899	8	42
8	9	18	16	30	17	47	12	16	784	25
9	14	8	8	8	57	13	3	46	13	839

3. Conclusion

It can be observed that MLP with H=75 case carries the most accurate predictions.

Number of units in a layer is an important indicator for prediction accuracy.

4. Source Code

The loop for MLP, H=25 Case.

```
for q in range (0,10): #Number of Epochs?
    epoch[q,0] = 0
    epoch_test[q,0] = 0 #test
    for t in range (0,60000): #Instances for Training Dataset: Change Based on numbe
r of instances.
        #for i in range (0,25):
        z = (np.dot(w, tr_images[t])) + bias_1
        if t < 10000:
            z_test = (np.dot(w, tt_images[t])) + bias_1 #test
        z_softmax = (softmax_z(z))
        y_holder[:,t] = z_softmax[0,:]
        if t < 10000:
            z_test_softmax = (softmax_z(z_test)) #test
            y_holder_test[:,t] = z_test_softmax[0,:] #test
            #z is the new x
        tentative_y = (np.dot(v, z)) + bias_2
        if t < 10000:
            tentative_y_test = (np.dot(v, z_test)) + bias_2 #test
        tentative_y_softmax = (softmax(tentative_y))
        tentative_y_holder[t,:] = tentative_y_softmax[0,:] # dikkat
        if t < 10000:
            tentative_y_test_softmax = (softmax(tentative_y_test)) #test
            tentative_y_test_holder[t,:] = tentative_y_test_softmax[0,:]

        for ix in range (0,10):
            loss = desired_training[t,ix] - tentative_y_holder[t,ix] #z_softmax?
            loss_tracker.append(loss)
            v[ix] = v[ix] + 0.1*(loss)*z
        for h in range (0,75):
            loss_2 = np.sum(desired_training[t] - tentative_y_holder[t]) * np.sum(v[
:,h])
            ccc = np.dot(np.dot(loss_2, z[h]),(1-z[h]))
            w[h] = w[h] + 0.1*(np.dot(ccc,tr_images[t]))

        if tr_labels[t] == np.argmax(tentative_y_holder[t]):
            epoch[q,0] = epoch[q,0] + 1
        if t < 10000 and tt_labels[t] == np.argmax(tentative_y_test_holder[t]): #tes
t
            epoch_test[q,0] = epoch_test[q,0] + 1
    q = q + 1
```

SLP Code.

```
import pandas as pd
import matplotlib.pyplot as plt
from mnist import MNIST
import numpy as np
import random
import math

np.random.seed(60) # reproducability
mndata = MNIST('./mnist')

#Added in order to read the file successfully, to decompress file.
mndata.gz = True

# read training images and corresponding labels
tr_images, tr_labels = mndata.load_training()
# read test images and corresponding labels
tt_images, tt_labels = mndata.load_testing()

# convert lists into numpy format and apply normalization
tr_images = np.array(tr_images) / 255. # shape (60000, 784)
tr_labels = np.array(tr_labels)         # shape (60000,)
tt_images = np.array(tt_images) / 255. # shape (10000, 784)
tt_labels = np.array(tt_labels)         # shape (10000,)

"""Consider Randomness for xt & rt if needed, later on
#randomness of t
np.random.seed(60) # reproducability
r = list(range(10000))
random.shuffle(r)
print(r)"""

#Desired Training

f = 0
t = 0
holder = 0

desired_training = pd.DataFrame(index=range(60000),columns=range(10))
desired_training = desired_training.values

for f in range (0,60000):
    holder = tr_labels[f]
    for t in range (10):
        if t != holder:
            desired_training[f][t] = 0
        elif t == holder:
            desired_training[f][holder] = 1


ts = 0
i = 0
t = 0
p = 0
j = 0
g = 0
k = 0
v = 0
r = 0
loss = 0
holder_list = []
error = []
accuracy = []
```

```
w = []
o = []
o = pd.DataFrame(index=range(1),columns=range(10))
o = o.values
o_test = []
o_test = pd.DataFrame(index=range(1),columns=range(10))
o_test = o_test.values
o_exp_sum = 0
y = []
y = pd.DataFrame(index=range(1),columns=range(10))
y=y.values
y_function = []
y_function = pd.DataFrame(index=range(1),columns=range(10))
y_function = y_function.values
y_test = []
y_test = pd.DataFrame(index=range(1),columns=range(10))
y_test = y_test.values
y_holder = []
y_holder = pd.DataFrame(index=range(60000),columns=range(10)) #change for training instance
y_holder = y_holder.values
y_holder_test = []
y_holder_test = pd.DataFrame(index=range(10000),columns=range(10)) #change for test instance
y_holder_test = y_holder_test.values
epoch = []
epoch = pd.DataFrame(index=range(60000),columns=range(1)) #change for training instance
epoch = epoch.values
epoch_test = []
epoch_test = pd.DataFrame(index=range(10000),columns=range(1)) #change for test instance
epoch_test = epoch_test.values
soft = []
soft = pd.DataFrame(index=range(1),columns=range(10))
soft=soft.values
        w = pd.DataFrame(index=range(10),columns=range(784))
        w=w.values

        #Softmax
        def softmax(t, r=0):
            for i in range (0,10):
                r = r + (math.exp(t[0][i]))
            for i in range (0,10):
                y_function[0][i] = math.exp(t[0][i]) / r
            return y_function


        l=(softmax(o))
        l.sum()

        for i in range (0,10):
            for j in range (0,784):
                w[i][j] = random.uniform(-0.01, 0.01)

        bias = np.random.uniform(-0.01, 0.01, 10)


        for q in range (0,50): #Number of Epochs?
            epoch[q,0] = 0
            epoch_test[q,0] = 0 #test
            for t in range (0,60000): #Instances for Training Dataset: Change Based on number of instances.
                o[0,:] = 0
                o_test[0,:] = 0 #test
                o = o + (np.dot(w, tr_images[t])) + bias
```

```
                if t < 10000:
                    o_test = o_test + (np.dot(w, tt_images[t])) + bias #test
                y = (softmax(o))
                y_holder[t,:] = y[0,:]
                if t < 10000:
                    y_test = (softmax(o_test)) #test
                    y_holder_test[t,:] = y_test[0,:] #test
                for i in range (0,10):
                    loss = desired_training[t,i] - y[0,i]
                    w[i] = w[i] + 0.0001*(loss)*tr_images[t]
                if tr_labels[t] == np.argmax(y_holder[t]):
                    epoch[q,0] = epoch[q,0] + 1
                if t < 10000 and tt_labels[t] == np.argmax(y_holder_test[t]): #test
                    epoch_test[q,0] = epoch_test[q,0] + 1
            q = q + 1


        plt.plot(epoch/60000)
        plt.title('accuracy per epoch - training - %')
        plt.show()

        plt.plot(epoch_test/10000)
        plt.title('accuracy per epoch - test - %')
        plt.show()

        #confusion matrix

        con_y_holder= y_holder.copy()
        con_y_holder_test = y_holder_test.copy()

        confusion_train = []
        confusion_train = pd.DataFrame(index=range(10),columns=range(10))
        confusion_train = confusion_train.values
        confusion_test = []
        confusion_test = pd.DataFrame(index=range(10),columns=range(10))
        confusion_test = confusion_test.values
        conf = 0
        conf_label = 0
        conf_assigned = 0

        #training confusion matrix prep

        for i in range (0,10):
            for j in range (0,10):
                confusion_train[i][j] = 0


        #training confusion matrix
        for i in range (0,60000):
            if np.argmax(y_holder[i]) == tr_labels[i]:
                conf_label = tr_labels[i]
                confusion_train[conf_label,conf_label] = confusion_train[conf_label,c
    onf_label] + 1
            elif np.argmax(y_holder[i]) != tr_labels[i]:
                conf_assigned = np.argmax(y_holder[i])
                conf_label = tr_labels[i]
                confusion_train[conf_label,conf_assigned] = confusion_train[conf_labe
    l,conf_assigned] + 1

        #test confusion matrix
        conf_label = 0
        conf_assigned = 0

        for i in range (0,10):
            for j in range (0,10):
                confusion_test[i][j] = 0
```

```
193.
194.       for i in range (0,10000):
195.           if np.argmax(y_holder_test[i]) == tt_labels[i]:
196.               conf_label = tt_labels[i]
197.               confusion_test[conf_label,conf_label] = confusion_test[conf_label,con
    f_label] + 1
198.           elif np.argmax(y_holder_test[i]) != tt_labels[i]:
199.               conf_assigned = np.argmax(y_holder_test[i])
200.               conf_label = tt_labels[i]
201.               confusion_test[conf_label,conf_assigned] = confusion_test[conf_label,
    conf_assigned] + 1
202.
203.
204.
205.       df_confusion_test = pd.DataFrame(confusion_test)
206.       df_confusion_train = pd.DataFrame(confusion_train)
207.
208.
209.
210.       w_transpose = w.transpose()
211.       #Visualization Test
212.
213.       #from matplotlib import pyplot
214.       import numpy as np
215.       #import csv
216.       import matplotlib.pyplot as plt
217.       k = 0
218.       for k in range (0,10):
219.           pixels_mean = w_transpose[0:]
220.           pixels_mean = np.array(pixels_mean, dtype='uint8')
221.           deneme = pixels_mean[:,k].reshape(28, 28) #CHANGE k based on numbers
222.           plt.title('Weight Visualization: Label ' + str(k))
223.           plt.imshow(deneme, cmap='gray')
224.           plt.show()
```

MLP Code.

```
1.  import pandas as pd
2.  import matplotlib.pyplot as plt
3.  from mnist import MNIST
4.  import numpy as np
5.  import random
6.  import math
7.
8.  np.random.seed(60) # reproducability
9.  mndata = MNIST('./mnist')
10.
11. #Added in order to read the file successfully, to decompress file.
12. mndata.gz = True
13.
14. # read training images and corresponding labels
15. tr_images, tr_labels = mndata.load_training()
16. # read test images and corresponding labels
17. tt_images, tt_labels = mndata.load_testing()
18.
19. # convert lists into numpy format and apply normalization
20. tr_images = np.array(tr_images) / 255. # shape (60000, 784)
21. tr_labels = np.array(tr_labels)         # shape (60000,)
22. tt_images = np.array(tt_images) / 255. # shape (10000, 784)
23.
24. f = 0
```

```
t = 0
holder = 0

desired_training = pd.DataFrame(index=range(60000),columns=range(10))
desired_training = desired_training.values

for f in range (0,60000):
    holder = tr_labels[f]
    for t in range (10):
        if t != holder:
            desired_training[f][t] = 0
        elif t == holder:
            desired_training[f][holder] = 1


ts = 0
i = 0
ix = 0
t = 0
p = 0
j = 0
g = 0
k = 0
v = 0
r = 0
h = 0
loss = 0
loss_2 = 0
holder_list = []
error = []
accuracy = []
w = []
o = []
o = pd.DataFrame(index=range(1),columns=range(10))
o = o.values
z = []
z = pd.DataFrame(index=range(75),columns=range(1)) #change based on h
z = z.values
tentative_y = []
tentative_y = pd.DataFrame(index=range(10),columns=range(1)) #change based on h
tentative_y = tentative_y.values
ccc = []
ccc = pd.DataFrame(index=range(75),columns=range(1)) #change based on h
ccc = ccc.values
tentative_y_softmax = []
tentative_y_softmax = pd.DataFrame(index=range(1),columns=range(10)) #change based o
n h
tentative_y_softmax = tentative_y_softmax.values
tentative_y_softmax_test = []
tentative_y_softmax_test = pd.DataFrame(index=range(1),columns=range(10)) #change ba
sed on h
tentative_y_softmax_test = tentative_y_softmax_test.values
tentative_y_test = []
tentative_y_test = pd.DataFrame(index=range(10),columns=range(1)) #change based on h

tentative_y_test = tentative_y_test.values
z_test = []
z_test = pd.DataFrame(index=range(75),columns=range(1)) #change based on h
z_test = z_test.values
o_test = []
o_test = pd.DataFrame(index=range(1),columns=range(10))
o_test = o_test.values
o_exp_sum = 0
y = []
y = pd.DataFrame(index=range(1),columns=range(10))
y=y.values
```

```
y_function = []
y_function = pd.DataFrame(index=range(1),columns=range(10))
y_function = y_function.values
y_function_z = []
y_function_z = pd.DataFrame(index=range(1),columns=range(75))
y_function_z = y_function_z.values
y_test = []
y_test = pd.DataFrame(index=range(1),columns=range(10))
y_test = y_test.values
z_softmax = []
z_softmax = pd.DataFrame(index=range(1),columns=range(10))
z_softmax = z_softmax.values
        z_test_softmax = []
        z_test_softmax = pd.DataFrame(index=range(1),columns=range(10))
        z_test_softmax = z_test_softmax.values
        y_holder = []
        y_holder = pd.DataFrame(index=range(75),columns=range(60000)) #change for tra
    ining instance
        y_holder = y_holder.values
        tentative_y_holder = []
        tentative_y_holder = pd.DataFrame(index=range(60000),columns=range(10)) #chan
    ge for training instance
        tentative_y_holder = tentative_y_holder.values
        tentative_y_test_holder = []
        tentative_y_test_holder = pd.DataFrame(index=range(10000),columns=range(10))
    #change for training instance
        tentative_y_test_holder = tentative_y_test_holder.values
        y_holder_test = []
        y_holder_test = pd.DataFrame(index=range(75),columns=range(10000)) #change fo
    r test instance
        y_holder_test = y_holder_test.values
        epoch = []
        epoch = pd.DataFrame(index=range(60000),columns=range(1)) #change for trainin
    g instance
        epoch = epoch.values
        epoch_test = []
        epoch_test = pd.DataFrame(index=range(10000),columns=range(1)) #change for te
    st instance
        epoch_test = epoch_test.values
        soft = []
        soft = pd.DataFrame(index=range(1),columns=range(10))
        soft=soft.values
        w = pd.DataFrame(index=range(75),columns=range(784))
        w=w.values
        v = []
        v = pd.DataFrame(index=range(10),columns=range(75))
        v=v.values
        loss_tracker = []

        #Softmax
        def softmax(t, r=0):
            for i in range (0,10):
                r = r + (math.exp(t[i]))
            for i in range (0,10):
                y_function[0,i] = math.exp(t[i]) / r
            return y_function

        def softmax_z(t, r=0):
            for i in range (0,75):
                r = r + (math.exp(t[i]))
            for i in range (0,75):
                y_function_z[0,i] = math.exp(t[i]) / r
            return y_function_z


        for i in range (0,75):
```

```
            for j in range (0,784):
                w[i][j] = random.uniform(-0.01, 0.01)

        for i in range (0,10):
            for j in range (0,75):
                v[i][j] = random.uniform(-0.01, 0.01)

        bias_1 = np.random.uniform(-0.01, 0.01, 75)
        bias_2 = np.random.uniform(-0.01, 0.01, 10)

        z[:,0] = 0
        z_test[:,0] = 0 #test
        tentative_y [:,0] = 0
        tentative_y_test[:,0] = 0

        for q in range (0,10): #Number of Epochs?
            epoch[q,0] = 0
            epoch_test[q,0] = 0 #test
            for t in range (0,60000): #Instances for Training Dataset: Change Based o
    n number of instances.
                #for i in range (0,25):
                z = (np.dot(w, tr_images[t])) + bias_1
                if t < 10000:
                    z_test = (np.dot(w, tt_images[t])) + bias_1 #test
                z_softmax = (softmax_z(z))
                y_holder[:,t] = z_softmax[0,:]
                if t < 10000:
                    z_test_softmax = (softmax_z(z_test)) #test
                    y_holder_test[:,t] = z_test_softmax[0,:] #test
                    #z is the new x
                tentative_y = (np.dot(v, z)) + bias_2
                if t < 10000:
                    tentative_y_test = (np.dot(v, z_test)) + bias_2 #test
                tentative_y_softmax = (softmax(tentative_y))
                tentative_y_holder[t,:] = tentative_y_softmax[0,:] # dikkat
                if t < 10000:
                    tentative_y_test_softmax = (softmax(tentative_y_test)) #test
                    tentative_y_test_holder[t,:] = tentative_y_test_softmax[0,:]

                for ix in range (0,10):
                    loss = desired_training[t,ix] - tentative_y_holder[t,ix] #z_softm
    ax?
                    loss_tracker.append(loss)
                    v[ix] = v[ix] + 0.1*(loss)*z
                for h in range (0,75):
                    loss_2 = np.sum(desired_training[t] - tentative_y_holder[t]) * np
    .sum(v[:,h])
                    ccc = np.dot(np.dot(loss_2, z[h]),(1-z[h]))
                    w[h] = w[h] + 0.1*(np.dot(ccc,tr_images[t]))

                if tr_labels[t] == np.argmax(tentative_y_holder[t]):
                    epoch[q,0] = epoch[q,0] + 1
                if t < 10000 and tt_labels[t] == np.argmax(tentative_y_test_holder[t]
    ): #test
                    epoch_test[q,0] = epoch_test[q,0] + 1
            q = q + 1

        plt.plot(epoch) #change to number of instances for rate
        plt.title('accuracy per epoch - training')
        plt.show()

        plt.plot(epoch_test) #change to number of instances for rate
        plt.title('accuracy per epoch - test')
        plt.show()


```

```
210.        plt.plot(loss_tracker)
211.        plt.show()
212.
213.        xxx = (np.dot(ccc,tr_images[3]))
```