# Deep Learning

## Assignment 2 Part 1

Pramil Panjawani (PhD19008)

Reshan Faraz (PhD19006)

Date : 21-Feb-2021

## Q1) For SGD optimizer find the best activation function

1) Accuracy

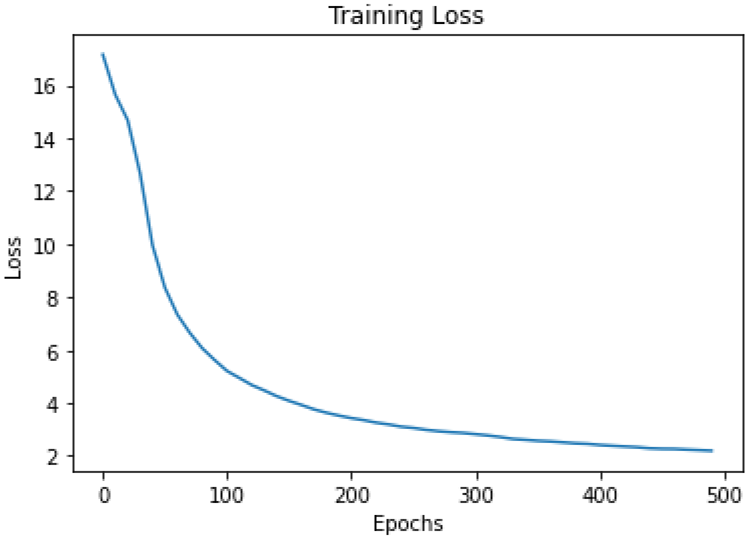
| **Activation Function** | **Train accuracy** | **Test accuracy** |
|---|---|---|
| | | |
| ReLU | 0.8616 | 0.8025 |
| Sigmoid | 0.7017 | 0.6685 |
| tanh | 0.8172 | 0.764 |

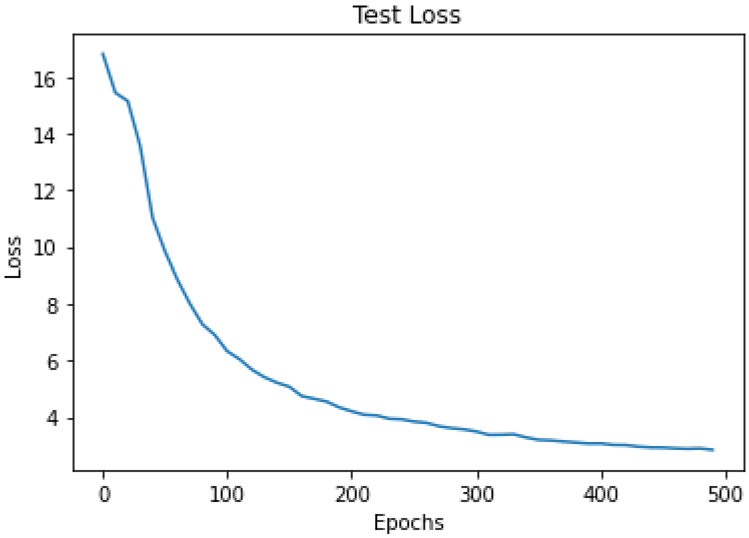
2) Loss after 500 Epochs

| Activation Function | Train Loss | Test Loss |
|---|---|---|
| | | |
| ReLU | 2.0416 | 3.1785 |
| Sigmoid | 4.51 | 5.245 |
| tanh | 3.0101 | 3.926 |

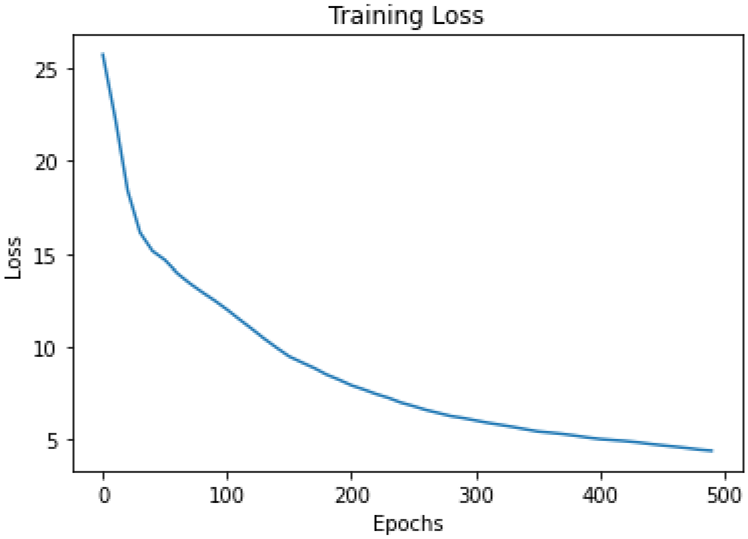
3) Train and Test loss vs Epochs

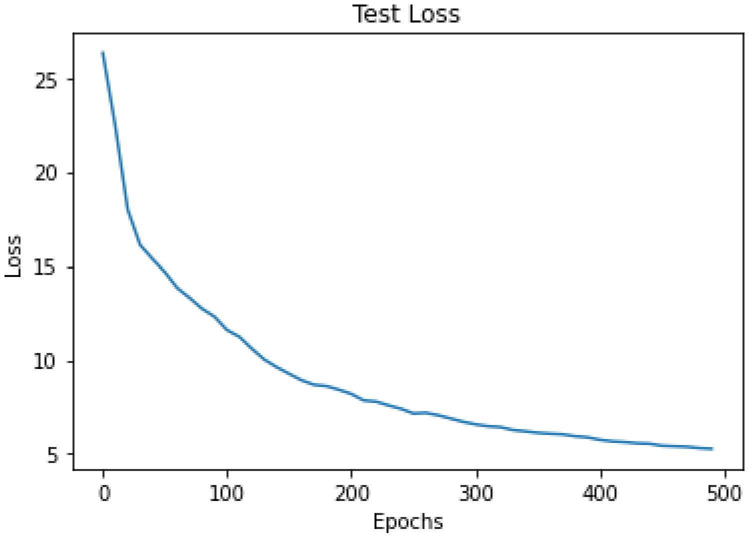
i) ReLU



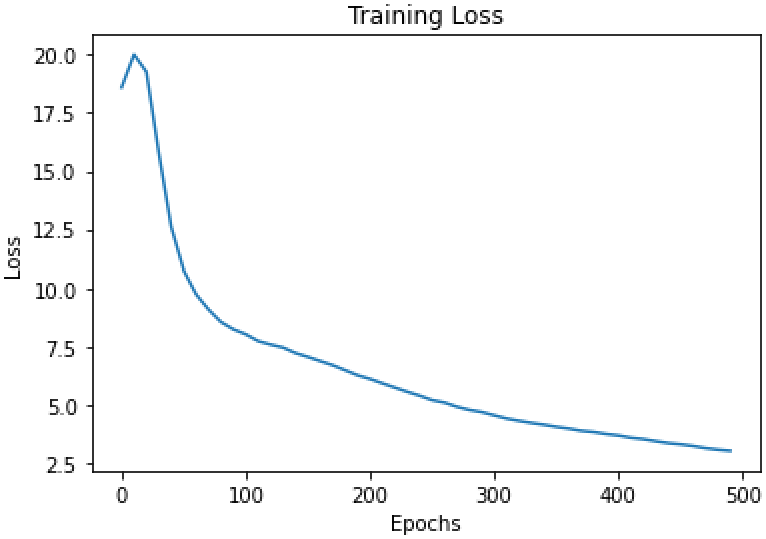


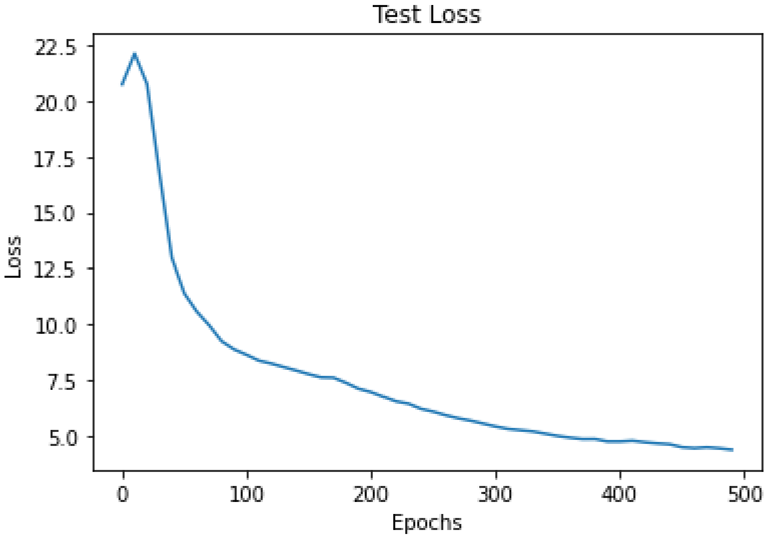
ii) Sigmoid activation





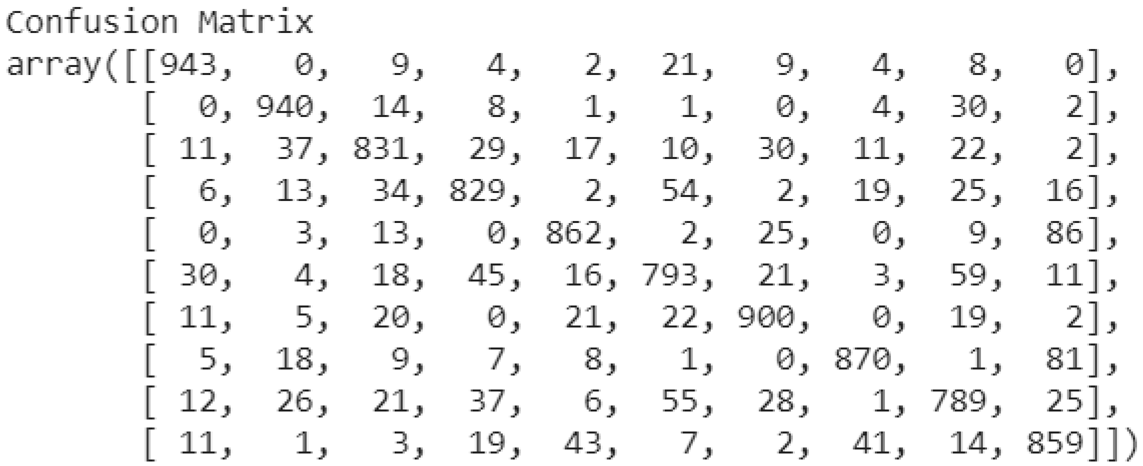
iii) tanh


Training Loss
Loss
Epochs
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0
100
200
300
400
500


Test Loss
Loss
Epochs
22.5
20.0
17.5
15.0
12.5
10.0
7.5
5.0
0
100
200
300
400
500

4) Confusion Matrix

i) ReLU



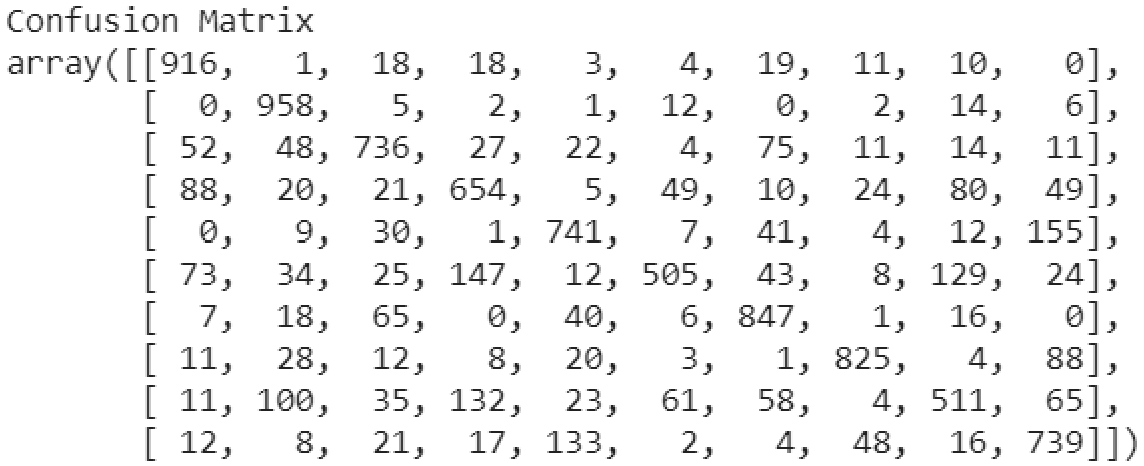
```
Confusion Matrix
array([[943,   0,   9,   4,   2,  21,   9,   4,   8,   0],
       [  0, 940,  14,   8,   1,   1,   0,   4,  30,   2],
       [ 11,  37, 831,  29,  17,  10,  30,  11,  22,   2],
       [  6,  13,  34, 829,   2,  54,   2,  19,  25,  16],
       [  0,   3,  13,   0, 862,   2,  25,   0,   9,  86],
       [ 30,   4,  18,  45,  16, 793,  21,   3,  59,  11],
       [ 11,   5,  20,   0,  21,  22, 900,   0,  19,   2],
       [  5,  18,   9,   7,   8,   1,   0, 870,   1,  81],
       [ 12,  26,  21,  37,   6,  55,  28,   1, 789,  25],
       [ 11,   1,   3,  19,  43,   7,   2,  41,  14, 859]])
```

We can see most classes are correctly classified

4 is the least  precise

0, 1, 6 are mostly precise

ii) Sigmoid



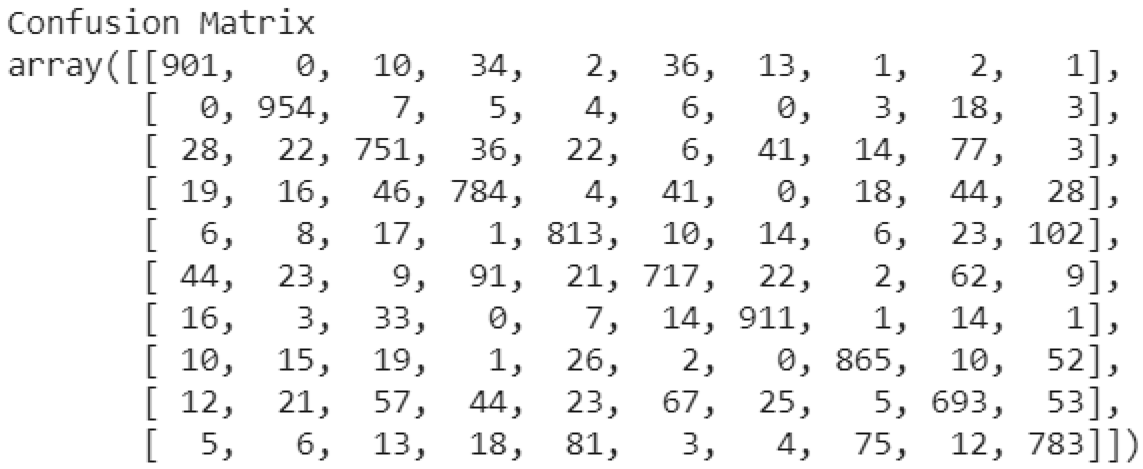
```
Confusion Matrix
array([[916,   1,  18,  18,   3,   4,  19,  11,  10,   0],
       [  0, 958,   5,   2,   1,  12,   0,   2,  14,   6],
       [ 52,  48, 736,  27,  22,   4,  75,  11,  14,  11],
       [ 88,  20,  21, 654,   5,  49,  10,  24,  80,  49],
       [  0,   9,  30,   1, 741,   7,  41,   4,  12, 155],
       [ 73,  34,  25, 147,  12, 505,  43,   8, 129,  24],
       [  7,  18,  65,   0,  40,   6, 847,   1,  16,   0],
       [ 11,  28,  12,   8,  20,   3,   1, 825,   4,  88],
       [ 11, 100,  35, 132,  23,  61,  58,   4, 511,  65],
       [ 12,   8,  21,  17, 133,   2,   4,  48,  16, 739]])
```

Classes are less well classified

5,8 is the least precise

0, 1 are mostly precise

iii) tanh



```
Confusion Matrix
array([[901,   0,  10,  34,   2,  36,  13,   1,   2,   1],
       [  0, 954,   7,   5,   4,   6,   0,   3,  18,   3],
       [ 28,  22, 751,  36,  22,   6,  41,  14,  77,   3],
       [ 19,  16,  46, 784,   4,  41,   0,  18,  44,  28],
       [  6,   8,  17,   1, 813,  10,  14,   6,  23, 102],
       [ 44,  23,   9,  91,  21, 717,  22,   2,  62,   9],
       [ 16,   3,  33,   0,   7,  14, 911,   1,  14,   1],
       [ 10,  15,  19,   1,  26,   2,   0, 865,  10,  52],
       [ 12,  21,  57,  44,  23,  67,  25,   5, 693,  53],
       [  5,   6,  13,  18,  81,   3,   4,  75,  12, 783]])
```

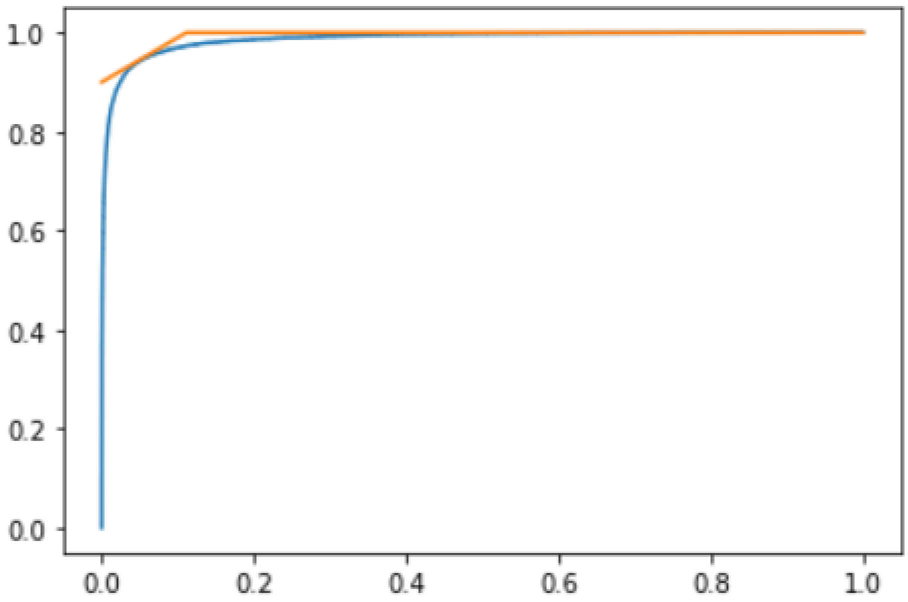
Classes are well classified

0, 1,6 are mostly precise

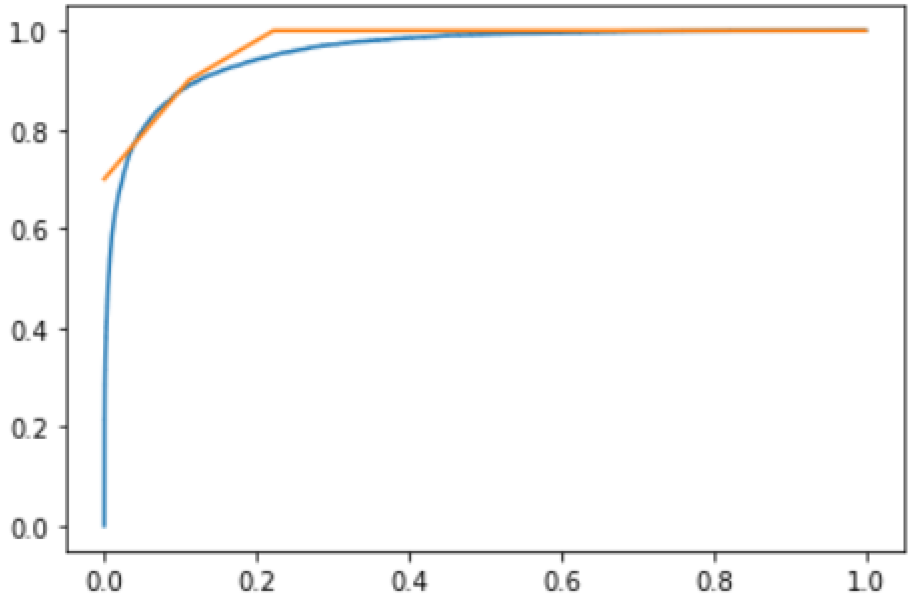
No classes are particularly imprecise

5) ROC curve

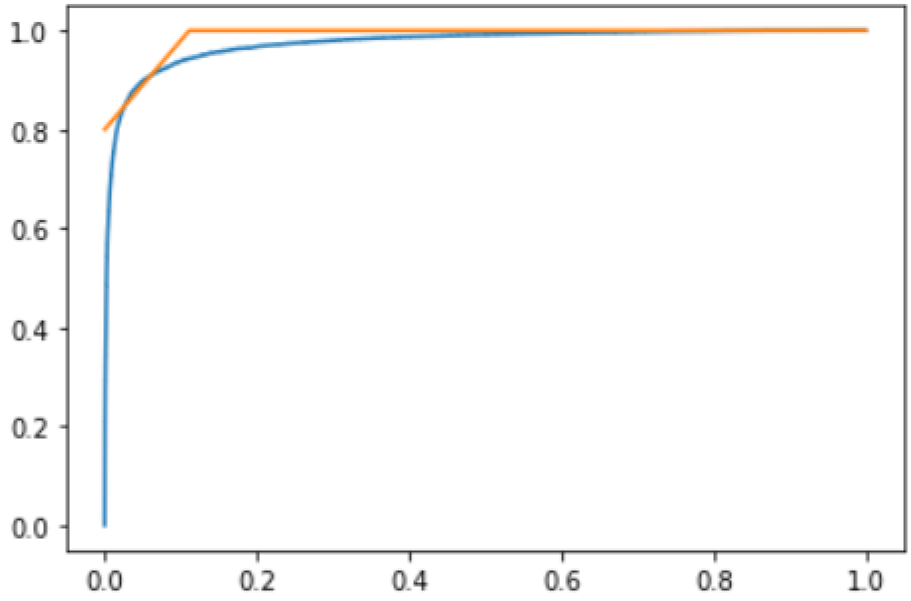
i) ReLU



ii) Sigmoid

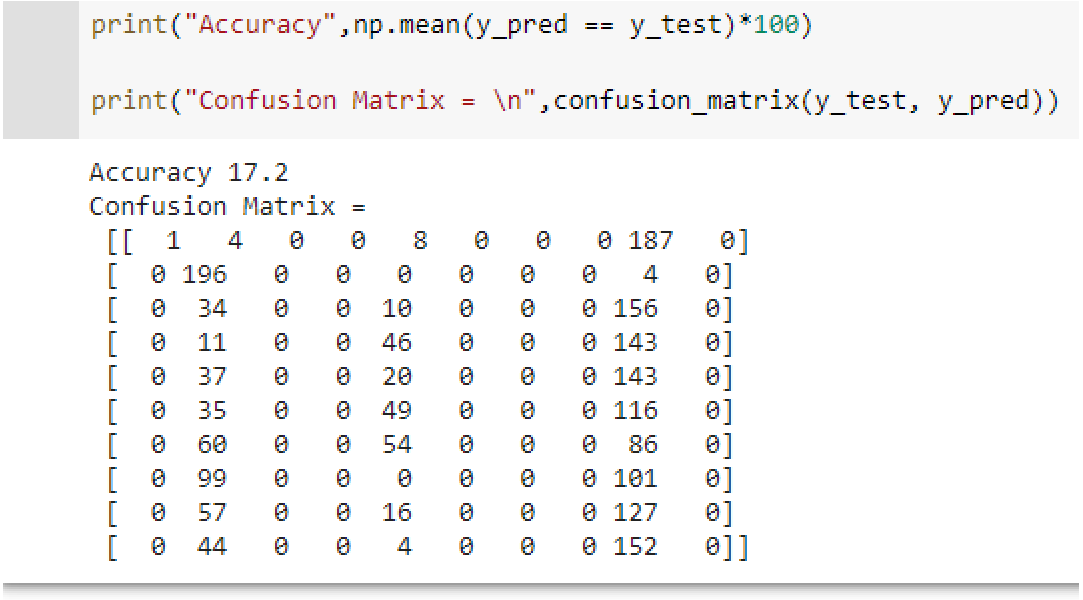


iii) tanh



## Q2 ) Results of Different Optimizers :

### Gradient Descent with Momentum :

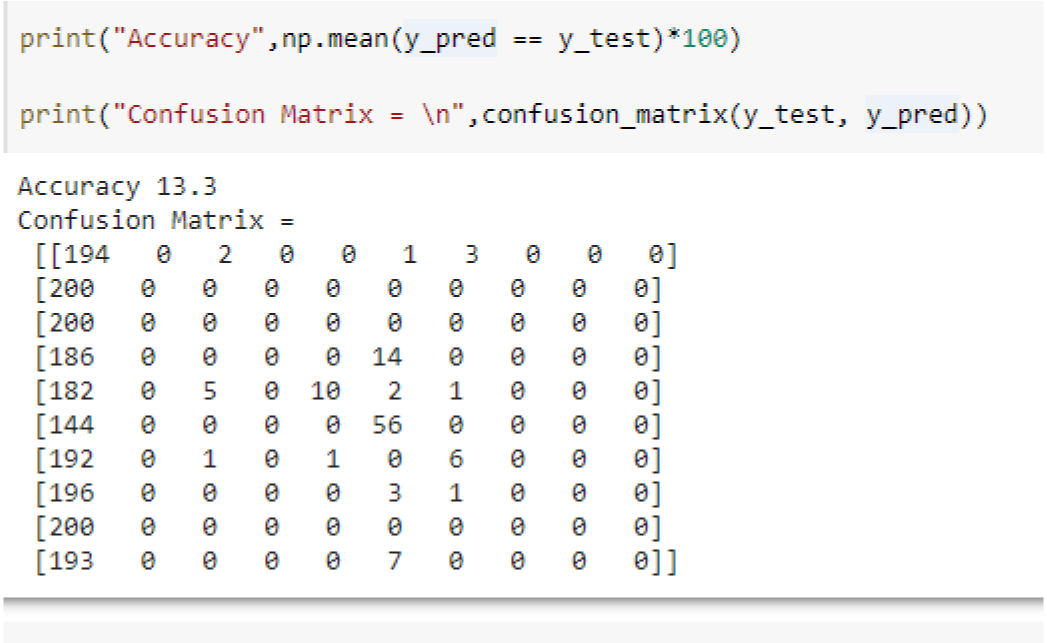


```
print("Accuracy",np.mean(y_pred == y_test)*100)

print("Confusion Matrix = \n",confusion_matrix(y_test, y_pred))
```

```
Accuracy 17.2
Confusion Matrix =
 [[  1   4   0   0   8   0   0   0 187   0]
 [  0 196   0   0   0   0   0   0   4   0]
 [  0  34   0   0  10   0   0   0 156   0]
 [  0  11   0   0  46   0   0   0 143   0]
 [  0  37   0   0  20   0   0   0 143   0]
 [  0  35   0   0  49   0   0   0 116   0]
 [  0  60   0   0  54   0   0   0  86   0]
 [  0  99   0   0   0   0   0   0 101   0]
 [  0  57   0   0  16   0   0   0 127   0]
 [  0  44   0   0   4   0   0   0 152   0]]
```

### Nestrov’s Accelerated Gradient :

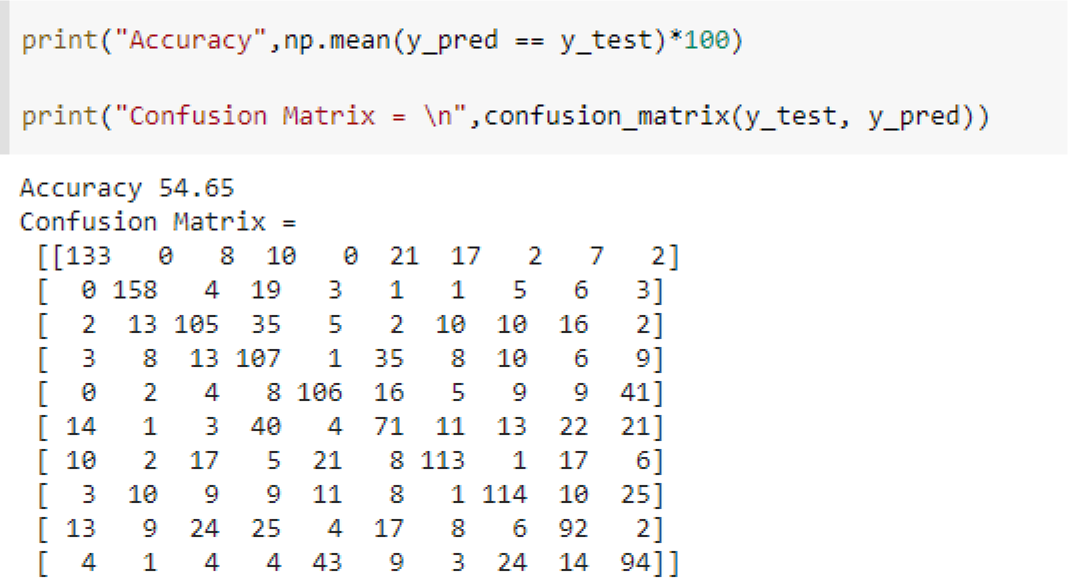


```
print("Accuracy",np.mean(y_pred == y_test)*100)

print("Confusion Matrix = \n",confusion_matrix(y_test, y_pred))
```

```
Accuracy 13.3
Confusion Matrix = 
 [[194   0   2   0   0   1   3   0   0   0]
 [200   0   0   0   0   0   0   0   0   0]
 [200   0   0   0   0   0   0   0   0   0]
 [186   0   0   0   0  14   0   0   0   0]
 [182   0   5   0  10   2   1   0   0   0]
 [144   0   0   0   0  56   0   0   0   0]
 [192   0   1   0   1   0   6   0   0   0]
 [196   0   0   0   0   3   1   0   0   0]
 [200   0   0   0   0   0   0   0   0   0]
 [193   0   0   0   0   7   0   0   0   0]]
```

**AdaGrad :**

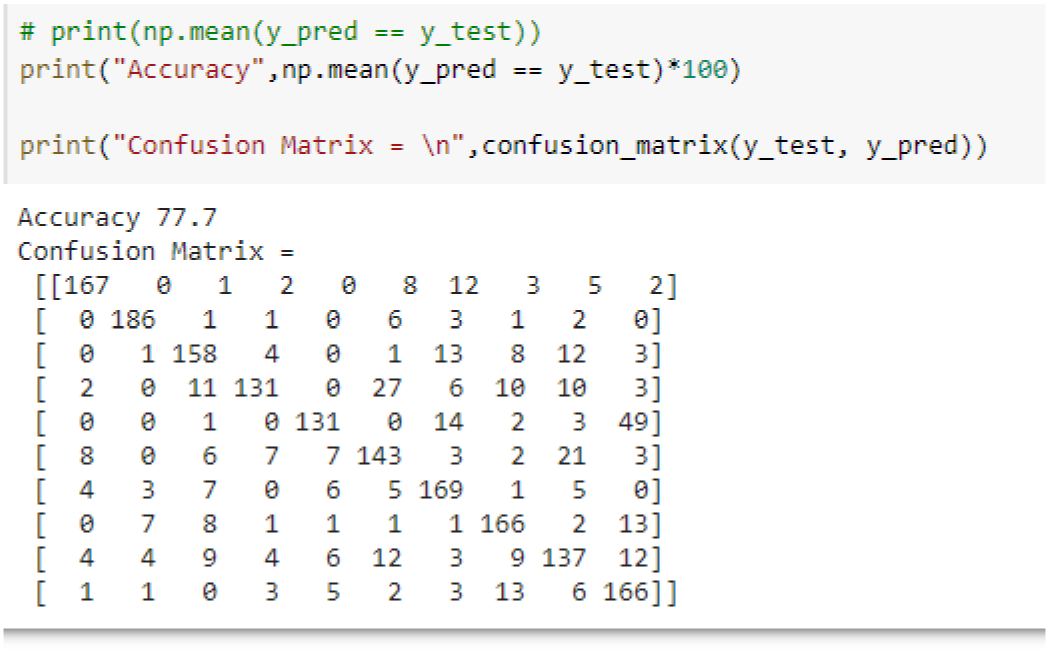


```
print("Accuracy",np.mean(y_pred == y_test)*100)

print("Confusion Matrix = \n",confusion_matrix(y_test, y_pred))
```

```
Accuracy 54.65
Confusion Matrix = 
 [[133   0   8  10   0  21  17   2   7   2]
 [  0 158   4  19   3   1   1   5   6   3]
 [  2  13 105  35   5   2  10  10  16   2]
 [  3   8  13 107   1  35   8  10   6   9]
 [  0   2   4   8 106  16   5   9   9  41]
 [ 14   1   3  40   4  71  11  13  22  21]
 [ 10   2  17   5  21   8 113   1  17   6]
 [  3  10   9   9  11   8   1 114  10  25]
 [ 13   9  24  25   4  17   8   6  92   2]
 [  4   1   4   4  43   9   3  24  14  94]]
```

**RMSProp :**

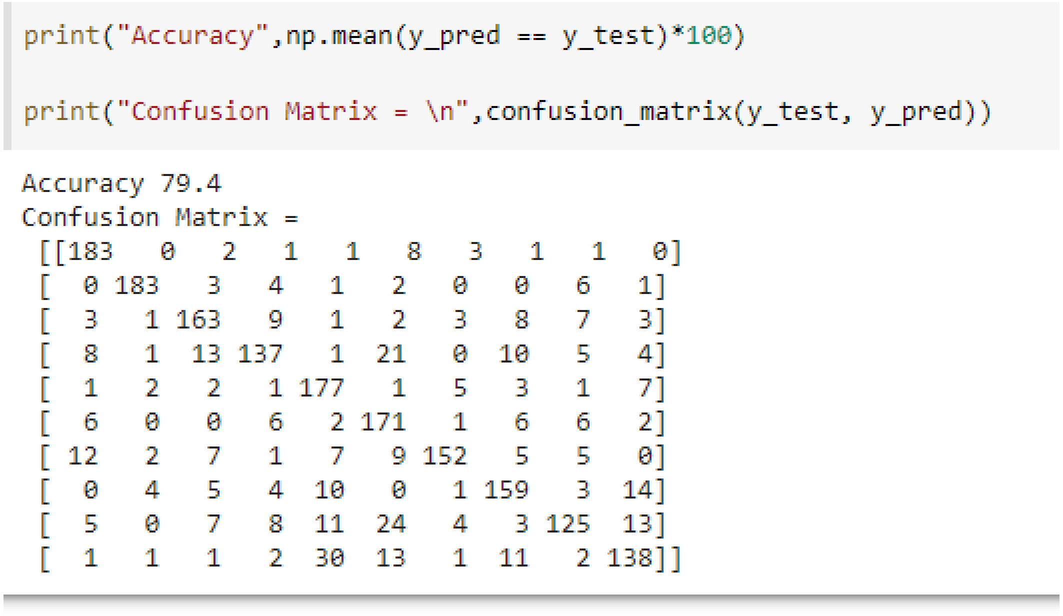


```
# print(np.mean(y_pred == y_test))
print("Accuracy",np.mean(y_pred == y_test)*100)

print("Confusion Matrix = \n",confusion_matrix(y_test, y_pred))
```

```
Accuracy 77.7
Confusion Matrix =
 [[167   0   1   2   0   8  12   3   5   2]
 [  0 186   1   1   0   6   3   1   2   0]
 [  0   1 158   4   0   1  13   8  12   3]
 [  2   0  11 131   0  27   6  10  10   3]
 [  0   0   1   0 131   0  14   2   3  49]
 [  8   0   6   7   7 143   3   2  21   3]
 [  4   3   7   0   6   5 169   1   5   0]
 [  0   7   8   1   1   1   1 166   2  13]
 [  4   4   9   4   6  12   3   9 137  12]
 [  1   1   0   3   5   2   3  13   6 166]]
```

**Adam :**



```
print("Accuracy",np.mean(y_pred == y_test)*100)

print("Confusion Matrix = \n",confusion_matrix(y_test, y_pred))
```

```
Accuracy 79.4
Confusion Matrix =
 [[183   0   2   1   1   8   3   1   1   0]
 [  0 183   3   4   1   2   0   0   6   1]
 [  3   1 163   9   1   2   3   8   7   3]
 [  8   1  13 137   1  21   0  10   5   4]
 [  1   2   2   1 177   1   5   3   1   7]
 [  6   0   0   6   2 171   1   6   6   2]
 [ 12   2   7   1   7   9 152   5   5   0]
 [  0   4   5   4  10   0   1 159   3  14]
 [  5   0   7   8  11  24   4   3 125  13]
 [  1   1   1   2  30  13   1  11   2 138]]
```